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

Jacek Gondzio   Andreas Grothey

April 15th, 2003

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

[*]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

Issues of implementation of a library for parallel interior-point methods for quadratic programming are addressed. The solver can easily exploit *any* special structure of the underlying optimization problem. In particular, it allows a nested embedding of structures and by this means very complicated real-life optimization problems can be modeled. The efficiency of the solver is illustrated on several problems arising in the financial planning, namely in the asset and liability management. The problems are modeled as multistage decision processes and by nature lead to specially structured tree-sparse problems. By taking the variance of the random variables into account the problems become nonseparable 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 10 million decision variables is solved in less than 2 hours on a serial computer. The approach is by nature scalable and the parallel implementation achieves perfect speed-ups.

1 Introduction

We are concerned in this paper with a parallel structure exploiting interior-point solver to tackle the 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 decision 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 [2, 27] 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

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

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 [27] 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 approach presented in this paper incorporates in a solver a set of routines that can support *any* structure. These are used in an object-oriented implementation which is an extension of [15, 13]. We assume that the structured matrices A and Q are built of a nested set of blocks. The hierarchy of blocks can be naturally represented as a tree: its root is the whole matrix, intermediate nodes are block-structured submatrices and the leaves are the elementary blocks.

Our approach takes advantage of inclusion polymorphism: we define a general abstract `Matrix` interface for matrix operations of which all other classes (one for each *type* of block-structured matrix supported) are implementations. This abstract `Matrix` interface contains a set of virtual functions (methods in the object-oriented terminology) that:

- provide all the necessary linear algebraic operations for an IPM, and
- allow self-referencing.

Among derived classes we define elementary ones for simple sparse and dense matrices as well as classes for block-structured matrices such as block-diagonal, block-sparse, block-dense and block bordered diagonal (which includes as special cases primal block-angular and dual block-angular matrices). Definitions of block-structured matrices use only references to the abstract `Matrix` interface . This self-referential property of our block-structured matrix classes allows us to represent a variety of nested structures. 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.

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 modeled as multi-stage stochastic linear programs [7, 9, 10, 18, 20, 28, 29]. 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 [3, 12, 14, 19, 22] or by the specialized interior point implementations [4, 17, 26]. Recently a number of interesting specialized approaches have been proposed for multistage quadratic programs. Steinbach [23] 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 [5, 6] 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 [21] extend the nested Benders decomposition to handle stochastic quadratic programming problems and use an IPM-based solver to solve all subproblems.

The origin of our approach is different: we assume that a QP problem is given with *any* block-structure in it (the structure does not have to be the consequence of dynamics and/or uncertainty) and we extend OOPS[1], the object-oriented LP solver [15, 13] to tackle QP problems. Multistage stochastic QPs are only one class of problems (among many others) which can be efficiently dealt with the QP extension of OOPS.

The approach presented in this paper is an extension of that implemented in OOPS [15, 13]. The main difference is that in OOPS [15] linear problems were considered so the augmented system of linear equations (1) did not contain matrix Q and was always reduced to normal equations

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

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

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 $A(Q+\Theta^{-1})^{-1}A^T$ and would be likely to incur prohibitively expensive computations (unless Q is a diagonal matrix). Hence in this paper and more generally in the QP extension of OOPS we always deal with the augmented system (1).

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.

The paper is organized as follows. In Section 2 we briefly discuss the linear algebraic operations required to implement the interior-point 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 introduce the tree representation of the block-structured matrices and discuss certain features of the abstract `Matrix` interface , the crucial object used in our design to support the notion of block. 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 Linear Algebra Kernel in the QP Solver

2.1 Linear Algebra in the Interior Point Method for QP

The basis for our implementation is the primal-dual interior point algorithm with multiple centrality correctors [1]. The main computational effort of this algorithm consists in the computation of the primal-dual Newton direction. This requires solving the following linear system

$$\begin{bmatrix} -Q - X^{-1}S & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \xi_d - X^{-1}\xi_\mu \\ \xi_p \end{bmatrix}, \tag{3}$$

where

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

The matrix involved in (3) is symmetric but indefinite (even for a 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

$$H = \begin{bmatrix} -Q - X^{-1}S & A^T \\ A & 0 \end{bmatrix} + \begin{bmatrix} -R_p & 0 \\ 0 & R_d \end{bmatrix} \tag{4}$$

to obtain a quasidefinite matrix [25]. The use of primal-dual regularization (4) 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 [8, 11].

The matrix in the regularized augmented system (4) 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 (3) 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 quasidefinite matrix [25], the symbolic phase of the factorization (reordering for sparsity and preparing data structures for sparse triangular factor) can be done before the optimization starts.

2.2 Block-Structured Matrices

In this section we give an example of exploitable structures in matrices Q and A. We show how by rearranging linear algebra operations a large structured linear system can be reduced to a sequence of small linear systems.

Assume that matrices A and Q are of the structure displayed in Figures 1 and 2, 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 3, i.e. a nested bordered diagonal form (the tilde in $\tilde{C}_{31}$ for example denotes that this is a part of the original C_{31} matrix). This reordering is generic, i.e. not depending on the particular structure of A and Q. In fact it can be seen that the matrix Φ in Figure 3 can be obtained directly from A/A^T and Q in Figures 1/2 by interleaving the matrices. Since Θ is a diagonal matrix and it does not change the sparsity pattern, we do not display it in Figure 3.

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 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. Of course, due to this, no physical reordering of memory entries has to be done to obtain the reordered augmented system matrix.

2.3 Symmetric Bordered Block-Diagonal Structure

Suppose a block of the augmented system Φ has the symmetric bordered block-diagonal structure.

$$\Phi = \begin{pmatrix} \Phi_1 & & & & B_1^T \\ & \Phi_2 & & & B_2^T \\ & & \ddots & & \vdots \\ & & & \Phi_n & B_n^T \\ B_1 & B_2 & \cdots & B_n & \Phi_0 \end{pmatrix}, \tag{5}$$

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 3 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.

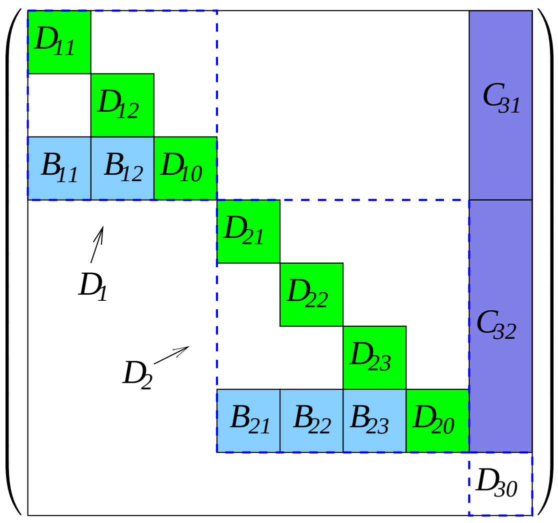

Figure 1: Example of Block-Structured Matrix A.

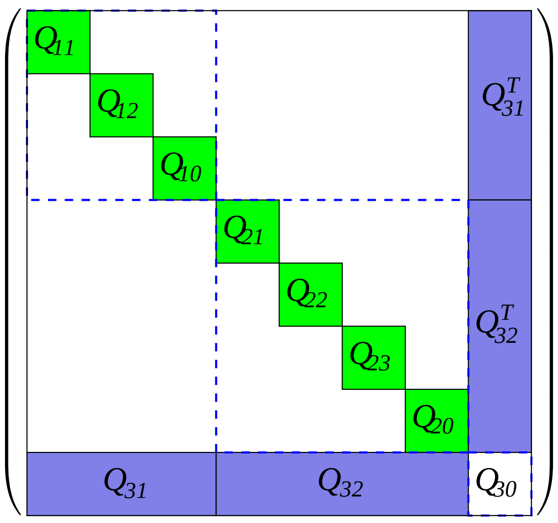

Figure 2: Example of Block-Structured Matrix Q.

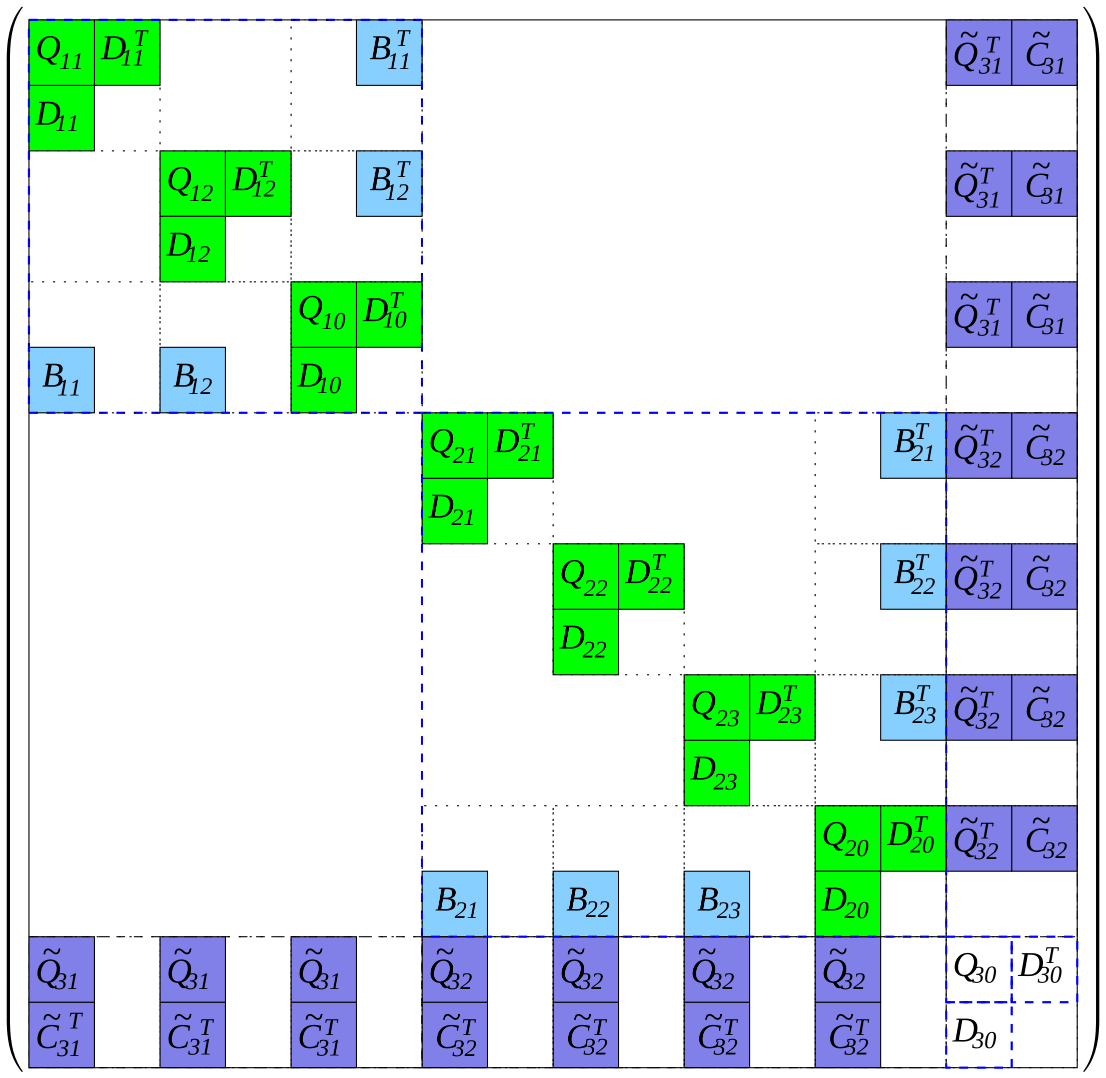
Q_{11} D_{11}^T B_{11}^T $\tilde{Q}_{31}^T$ $\tilde{C}_{31}$
D_{11}
Q_{12} D_{12}^T B_{12}^T $\tilde{Q}_{31}^T$ $\tilde{C}_{31}$
D_{12}
Q_{10} D_{10}^T $\tilde{Q}_{31}^T$ $\tilde{C}_{31}$
B_{11} B_{12} D_{10}
Q_{21} D_{21}^T B_{21}^T $\tilde{Q}_{32}^T$ $\tilde{C}_{32}$
D_{21}
Q_{22} D_{22}^T B_{22}^T $\tilde{Q}_{32}^T$ $\tilde{C}_{32}$
D_{22}
Q_{23} D_{23}^T B_{23}^T $\tilde{Q}_{32}^T$ $\tilde{C}_{32}$
D_{23}
Q_{20} D_{20}^T $\tilde{Q}_{32}^T$ $\tilde{C}_{32}$
B_{21} B_{22} B_{23} D_{20}
$\tilde{Q}_{31}$ $\tilde{Q}_{31}$ $\tilde{Q}_{31}$ $\tilde{Q}_{32}$ $\tilde{Q}_{32}$ $\tilde{Q}_{32}$ $\tilde{Q}_{32}$ Q_{30} D_{30}^T
$\tilde{C}_{31}^T$ $\tilde{C}_{31}^T$ $\tilde{C}_{31}^T$ $\tilde{C}_{32}^T$ $\tilde{C}_{32}^T$ $\tilde{C}_{32}^T$ $\tilde{C}_{32}^T$ D_{30}

Figure 3: Example of Block-Structured Augmented System Φ.

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

$$\begin{aligned} \Phi_i &= L_i D_i L_i^T && \text{(6a)} \\ L_{n,i} &= B_i L_i^{-T} D_i^{-1} && \text{(6b)} \\ C &= \Phi_0 - \sum_{i=1}^{n} B_i \Phi_i^{-1} B_i^T && \text{(6c)} \\ &= L_c D_c L_c^T && \text{(6d)} \end{aligned}$$

matrix Φ can be decomposed as

$$\Phi = LDL^T. \tag{7}$$

Because of this (7) 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 $x = (x_1, \dots, x_n, x_0)^T$, $b = (b_1, \dots, b_n, b_0)^T$ by the following operations:

$$\begin{aligned} z_i &= L_i^{-1} b_i, \quad i = 1, \dots, n && \text{(8a)} \\ z_0 &= L_c^{-1}(b_0 - \sum_{i=1}^{n} L_{n,i} z_i) && \text{(8b)} \\ y_i &= D_i^{-1} z_i, \quad i = 0, \dots, n && \text{(8c)} \\ x_0 &= L_c^{-T} y_0 && \text{(8d)} \\ x_i &= L_i^{-T}(y_i - L_{n,i}^T x_0), \quad i = 1, \dots, n && \text{(8e)} \end{aligned}$$

It is worth noting that the order in which matrix multiplications are performed in (6) and (8) may have a significant influence on the overall efficiency of the solution of equation $\Phi x = b$. The sum to compute C in (6c) 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 (8b) 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 (8e). For these reasons we refer to complex factorizations such as (6/8) as *implicit* factorizations.

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.

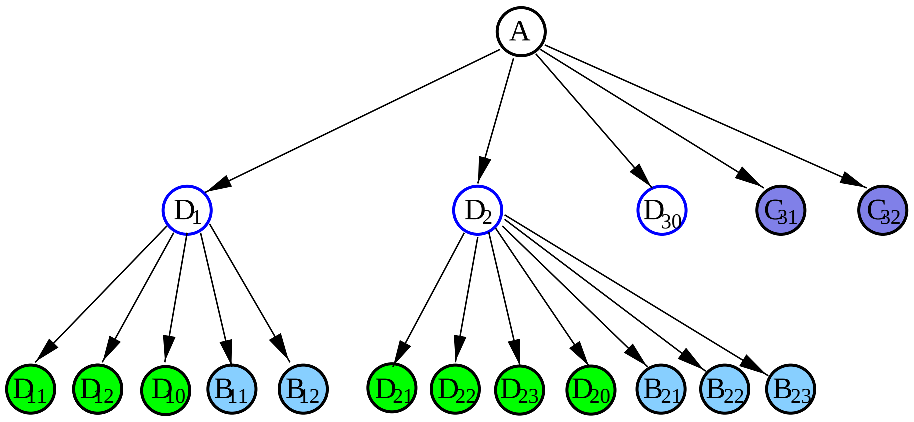

Figure 4: Tree Representation of Blocks.

3 Object-Oriented Design

The exploitation of a particular structure could be done with a traditional implementation of IPMs. Then however the code is not easily reusable for other structures. To achieve generality in our design we have followed [15].

There are many structures that can be exploited by an interior-point solver. We restrict our discussion to those in which the matrices A and Q are built of blocks and are compatible so that they imply a block structure for the reordered matrix Φ. There are two reasons for that. First, such structures are most often met in practice because they reflect the presence of dynamics, uncertainty, space distribution or other similar factors in the optimization problem. Secondly, block matrix structures can and should be exploited in any optimization algorithm that heavily involves linear algebraic operations, and the interior-point method is such an algorithm.

3.1 Tree Representation of Block-Structured Matrices

With a given block-structured matrix we associate a *tree* that defines the embedding of blocks. Every node in this tree corresponds to a block of the matrix. An arc indicates the embedding of blocks. The root of the tree is the whole matrix, and its leaves correspond to elementary blocks that are not partitioned into smaller entities. Any intermediate node of the tree corresponds to some submatrix that is further partitioned into blocks; the nodes corresponding to these blocks are children of a given node. Figure 4 shows the tree corresponding to the constraints matrix A in Figure 1.

The tree provides a complete description how the matrix is partitioned. In the example of Figures 1 and 4 the matrix is first partitioned into blocks D_1, D_2, D_{30}, C_{31} and C_{32} corresponding to dual block-angular structure. Blocks D_1 and D_2 have primal block-angular structure: they are partitioned into subblocks $D_{11}, D_{12}, D_{10}, B_{11}, B_{12}$ and $D_{21}, D_{22}, D_{23}, D_{20}, B_{21}, B_{22}, B_{23}$, respectively.

With every node of the tree we associate the *type* of the block-structure. We will say for example that A is a dual block-angular matrix with 2 diagonal blocks, D_1 and D_2 are primal block-angular matrices with 2 respectively 3 diagonal blocks, and all the remaining nodes are elementary matrices. Every *type* of the block structure corresponds in our layout to a particular implementation of the abstract `Matrix` interface .

Observe that the *type* of the node determines how the linear algebraic operations for the corresponding block of the matrix should be executed. Consider one of the simplest operations such as the matrix-vector product computed with A. The type of node A in the tree implies that the operation will be split into 5 subblocks D_1, D_2, D_{30}, C_{31} and C_{32}, corresponding to children of this node. In cases of nodes D_{30}, C_{31} and C_{32} that correspond to elementary blocks, this operation would invoke a routine appropriate for the given type of the node. For nodes D_1 and D_2 the same operation will have to be split further into subblocks — children of nodes D_1 and D_2. As seen in Section 2.2 the tree representation of matrices A, Q implies a tree decomposition of the augmented matrix Φ.

3.2 Matrix Interface for Augmented System Computations

We use object oriented principles to exploit the structure of matrices A, Q and Φ in the linear algebra operations needed within the interior point method for these matrices. The idea is that linear algebra operations are reduced to operations on nodes of the tree. Every operation for a particular type of matrix is reduced to a combination of operations on the children of this matrix. We have outlined above how this works for matrix-vector products and have given details of the more complex `ComputeCholesky` and `Solve` operations for bordered block-diagonal structure in Section 2.3.

Every type of matrix will correspond to a particular implementation of the abstract `Matrix` interface with its particular implementation of the linear algebra methods. In the rest of this section we will discuss the layout of the abstract `Matrix` interface .

The abstract `Matrix` interface provides a general interface to linear algebraic operations supported by any type of matrix. It ensures that every type of matrix supports the same linear algebraic operations and that they can be invoked in a generic way, i.e. without knowing the particular type of the matrix. Every particular type of matrix corresponds to an *implementation* of this abstract `Matrix` interface , i.e. it provides its specialized implementation of the abstract operations defined in the interface.

To implement an interior point method the following operations with the matrices A, Q and Φ are needed:

- Matrix-vector products Mz, M^Tz for A, Q and Φ.
- Given A, Q and Θ compute Φ.
- Calculate factors $\Phi = LDL^T$.
- Solve systems $Lz = g$, $Dz = g$ and $L^Tz = g$.

These operations are an integral part of the abstract `Matrix` interface used in our solver. It is worth noting that an interior-point algorithm does not need to know *how* these operations are executed; it needs to access only their *results*. However this set of routines is not sufficient. This is because the most efficient implementation of some of these *fundamental methods* might require further methods to be supported by its child node matrices. E.g. the formation of implicit Cholesky factors by Schur complement (see Section 2.3) requires a method that returns a row or column of the matrix in question.

We therefore introduce a set $\mathcal{F}_\mathcal{X}$ of methods supported by the abstract `Matrix` interface : it has to satisfy a closure condition: any method in $\mathcal{F}_\mathcal{X}$ can be implemented for any supported matrix structure by only using methods also in $\mathcal{F}_\mathcal{X}$ on its subblocks.

Below we list the set $\mathcal{F}_\mathcal{X}$ of methods supported by the abstract `Matrix` interface :

- compute the product $Mz = g$,
- compute the product $M^T z = g$,
- retrieve column j of M,
- retrieve row j of M,
- compute $\Phi = \text{reorder}\left(\begin{bmatrix} -Q - \Theta^{-1} & A^T \\ A & 0 \end{bmatrix}\right)$ for given A, Q and Θ,
- compute the inverse representation $LDL^T = \Phi$,
- compute the solution of equation $\Phi z = g$,
- compute the solution of equation $Lz = g$,
- compute the solution of equation $Dz = g$,
- compute the solution of equation $L^T z = g$.

Note that all these operations can be performed by only requiring the results of operations in $\mathcal{F}_\mathcal{X}$ from the subblocks Φ_i, B_i.

Not all these methods can be sensible used for a particular structure in all circumstances. The "compute Φ" method will only be used if the block is used as part of the constraint matrix A. Similarly the "Compute LDL^T" and "Solve" methods will only be applied for blocks used in the augmented system.

3.3 Parallel Implementation

One of the main advantages of an object-oriented linear algebra design is that it lends itself naturally to parallelization. Almost all linear algebra operations for the block-structured matrix types mentioned (e.g. block-bordered diagonal, primal/dual block-angular) can be efficiently computed in parallel, by distributing the blocks among different processors. The object oriented design provides a layout that can be adapted for parallel implementation with only minor changes.

Note that we only exploit this *coarse grain* parallelism: In block-structured matrices operations on child-matrices are distributed among processors. No attempt has been made to implement elementary matrices (sparse, dense) in parallel.

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/Liability Management (ALM) problems. This section will briefly summarize the structure of these problems. We will follow the problem description of [23, 24] and we refer the reader to [29] 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 [16].

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 on the null 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 [24]. The ALM problem can then be expressed as

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

If we denote $x_i = (x^s_{i,1}, x^b_{i,1}, x^h_{i,1}, \ldots, x^s_{i,J}, x^b_{i,J}, x^h_{i,J})$, and define matrices

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

and

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

where $c^b_j = (1+c_t)v_j, c^s_j = (1-c_t)v_j$, we can rewrite problem (9) as

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

Now if we 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 with a nested block-diagonal Q matrix of the form $Q = \mathrm{diag}(-1, Q_{\sigma(1)}, \ldots, Q_{\sigma(|L|-1)})$

and a constraint matrix A of the form

$$\left(\begin{array}{cc|cccc|c|cccc|}
-1 & & 0 & d_i & \cdots & d_i & \cdots & 0 & d_i & \cdots & d_i \\
 & A & & & & & & & & & \\
 & B_i & A & & & & & & & & \\
 & 0 & B_i & A & & & & & & & \\
 & \vdots & \vdots & & \ddots & & & & & & \\
 & 0 & B_i & & & A & & & & & \\
 & \vdots & & & & & \ddots & & & & \\
 & B_i & & & & & & A & & & \\
 & 0 & & & & & & B_i & A & & \\
 & \vdots & & & & & & \vdots & & \ddots & \\
 & 0 & & & & & & B_i & & & A
\end{array}\right) \qquad (11)$$

which is of nested primal/dual block-angular form, resulting in a QP of the form displayed in Figures 1/2/3. 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

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

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

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

As mentioned before this formulation of the problem is convex (Q now being a variance-covariance matrix) but dense.

Since our algorithm cannot currently exploit the dense Q in this second formulation, we have used the first formulation in our numerical tests. The non-convexity of the problem does not seem to adversely affect the performance of the algorithm.

5 Numerical Results

We shall discuss in this section 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 2-6 processors.

Problem	Stages	Blocks	Assets	Total Nodes	constraints	variables
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

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

Problem	iteration	Time (s)
ALM1	12	75.6
ALM2	19	1528
ALM3	29	7492
ALM4	31	5434
ALM5	16	6842
UNS1	26	5252

Table 2: Asset and Liability Management Problems: Serial Implementation.

All computations were done on the Sunfire 6800 machine at Edinburgh Parallel Computing Centre (EPCC). This machine features 24 750MHz UltraSparc-III processors and 48GB of shared memory.

In Table 1 we summarize problem statistics of our test-problems. Further we give solution times and statistics of the serial implementation in Table 2, and finally solution time and speed-up of the parallel implementation is reported in Table 3. While problems ALM1-ALM5 correspond to symmetric scenario trees, problem UNS1 is an example of an unsymmetric scenario tree. In this case the subblocks are distributed among processors by a greedy algorithm (assigning largest subblocks first).

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 is near perfect, due to the exact balance of the blocks treated by each processor and the by far major computational task being distributed amongst the processors. In fact we achieve superlinear speed-up on quite a few problems: we believe this to be the effect of more efficient use of cache when each processor only has to deal with a smaller part of the matrix and the corresponding vectors.

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* 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

Problem		1 proc		2 procs		k procs	
	k	time (s)	speed-up	time (s)	speed-up	time (s)	speed-up
ALM1	6	72.8	-	35.2	2.07	12.2	5.97
ALM2	5	1528	-	758	2.01	309	4.95
ALM3	5	7492	-	3661	2.04	1464	5.12
ALM4	6	5434	-	2717	2.00	905	6.00
ALM5	6	6842	-	3480	1.97	1150	5.95
UNS1	5	5252	-	2823	1.86	1108	4.74

Table 3: Asset and Liability Management Problems: Parallel Implementation.

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 10 million variables in about 20 minutes on 6 processors.

References

[1] A. Altman and J. Gondzio, *Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization*, Optimization Methods and Software, 11-12 (1999), pp. 275–302.

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

[3] J. R. Birge, *Decomposition and partitioning methods for multistage stochastic linear programs*, Operations Research, 33 (1985), pp. 989–1007.

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

[5] J. Blomvall and P. O. Lindberg, *A Riccati-based primal interior point solver for multistage stochastic programming*, European Journal of Operational Research, 143 (2002), pp. 452–461.

[6] ——, *A Riccati-based primal interior point solver for multistage stochastic programming - extensions*, Optimization Methods and Software, 17 (2002), pp. 383–407.

[7] S. Bradley and D. Crane, *A dynamic model for bond portfolio management*, Management Science, 19 (1972), pp. 139–151.

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

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

[10] G. Consigli and M. Dempster, *Dynamic stochastic programming for asset-liability management*, Annals of Operations Research, 81 (1998), pp. 131–162.

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

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

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

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

[15] J. Gondzio and R. Sarkissian, *Parallel interior point solver for structured linear programs*, Mathematical Programming, 96 (2003), pp. ??–??

[16] K. Høyland, M. Kaut, and S. W. Wallace, *A heuristic for moment-matching scenario generation*, Computational Optimization and Applications, 24 (2003), pp. 169–186.

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

[18] M. Kusy and W. Ziemba, *A bank asset and liability model*, Operations Research, 34 (1986), pp. 356–376.

[19] J. Linderoth and S. J. Wright, *Decomposition algorithms for stochastic programming on a computational grid*, Computational Optimization and Applications, 24 (2003), pp. 207–250.

[20] J. Mulvey and H. Vladimirou, *Stochastic network programming for financial planning problems*, Management Science, 38 (1992), pp. 1643–1664.

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

[22] A. Ruszczyński, *A regularized decomposition method for minimizing a sum of polyhedral functions*, Mathematical Programming, 33 (1985), pp. 309–333.

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

[24] ——, *Markowitz revisited: Mean variance models in financial portfolio analysis*, SIAM Review, 43 (2001), pp. 31–85.

[25] R. J. Vanderbei, *Symmetric quasidefinite matrices*, SIAM Journal on Optimization, 5 (1995), pp. 100–113.

[26] H. Vladimirou and S. A. Zenios, *Scalable parallel computations for large-scale stochastic programming*, Annals of Operations Research, 90 (1999), pp. 87–129.

[27] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, 1997.

[28] S. Zenios, *Asset/liability management under uncertainty for fixed-income securities*, Annals of Operations Research, 59 (1995), pp. 77–97.

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