Direct Solution of Linear Systems of Size 10^9 Arising in Optimization with Interior Point Methods*

Jacek Gondzio[1] and Andreas Grothey[1]

School of Mathematics, University of Edinburgh
JCMB, King's Buildings, Edinburgh, EH9 3JZ, UK
J.Gondzio@ed.ac.uk and A.Grothey@ed.ac.uk

Abstract. Solution methods for very large scale optimization problems are addressed in this paper. Interior point methods are demonstrated to provide unequalled efficiency in this context. They need a small (and predictable) number of iterations to solve a problem. A single iteration of interior point method requires the solution of indefinite system of equations. This system is regularized to guarantee the existence of triangular decomposition. Hence the well-understood parallel computing techniques developed for positive definite matrices can be extended to this class of indefinite matrices. A parallel implementation of an interior point method is described in this paper. It uses object-oriented programming techniques and allows for exploiting different block-structures of matrices. Our implementation outperforms the industry-standard optimizer, shows very good parallel efficiency on massively parallel architecture and solves problems of unprecedented sizes reaching 10^9 variables.

1 Introduction

Since their discovery [1] interior point methods (IPMs) have enjoyed well-deserved interest and have been subject of intensive study which led to a development of complete theory [2] and a thorough understanding of their implementation [3]. Interior point methods for optimization have a number of advantages. Depending on the algorithm used they guarantee finding a solution of the problem in not more than $\mathcal{O}(\sqrt{n})$ or $\mathcal{O}(n)$ iterations where n is the problem dimension. In practice they display a faster convergence suggesting that they enjoy $\mathcal{O}(\log n)$ complexity. But most of all, IPMs are reliable and can be implemented to provide unprecedented efficiency when applied to solve very large scale problems. We illustrate these features in this paper.

The bulk of work in every iteration of an interior point method is the solution of an indefinite system of equations. This system is regularized [4] to guarantee that an (indefinite) triangular Cholesky-like decomposition of it can be found. There exists a vast body of literature about parallel Cholesky decomposition.

* Supported by the Engineering and Physical Sciences Research Council of UK, EPSRC grant GR/R99683/01.

Indeed, the method is often implemented to exploit block-operations and all independent operations are executed on different processors.

To increase the degree of parallelism in the implementation of Cholesky factorization one looks for such an ordering of a sparse matrix which concentrates nonzero entries in independent blocks and if possible limits the fill-in to these blocks. Very large scale optimization problems by their very nature display block-structure. It is a consequence of the way how these problems are modelled. Models of engineering problems commonly involve indexing variables over discretizations in several dimensions hence they replicate few generic blocks. Such blocks are usually loosely coupled, and the word "loosely" translates into a high degree of sparsity displayed by matrices involved. Examples of such models include features such as:

- dynamics: inter-temporal connections are spread over a long horizon,
- uncertainty: scenarios are induced by stochastic (event) tree, or
- spatial distribution: functions are discretized over their domains.

We have developed a structure-exploiting optimization code called OOPS (Object-Oriented Parallel Solver) [5–7]. OOPS is an implementation of the primal-dual interior point method which uses all recent algorithmic advances (see `http://maths.ed.ac.uk/~gondzio/parallel/solver.html`).
It allows *any* block-structure of the optimization problem to be exploited by the linear algebra operations of the interior point method. In this paper we illustrate the parallel efficiency of this software. We apply it to a class of min-variance portfolio optimization problems [6, 8]. These models are quadratic (or nonlinear when higher order moments are used to measure risk). Stochastic programming modelling techniques [9] are used and this leads to challenging optimization problems which defy standard software.

The paper is organised as follows. In Section 2 interior point methods for optimization are briefly explained. In Section 3 the linear algebra operations involved by IPMs are discussed and exploiting block structure of matrices in these operations is addressed. In Section 4 the object-oriented implementation of linear algebra operations is briefly discussed and in Section 5 the formulation of min-variance portfolio optimization problems is given. In Section 6 the computational results are reported and in Section 7 the conclusions are given.

2 Interior Point Methods for Optimization

Developed over the last two decades, interior point methods have gained a strong position in the area of optimization. They easily generalise from linear, through quadratic to nonlinear programming and for all these classes of problems provide efficient algorithms. In this section we discuss IPMs applied to convex nonlinear programs and briefly comment on the simplified linear and quadratic models. Next, we show how their implementation can take advantage of three particular block-structures: primal and dual block-angular and bordered block-diagonal.

The reader interested in the theory of IPMs is encouraged to consult [2]; aspects of their implementation for general problems are discussed in [3].

Consider the convex nonlinear optimization problem

$$\min \quad f(x) \qquad \text{s.t.} \qquad g(x) \leq 0,$$

where $x \in \mathcal{R}^n$, and $f : \mathcal{R}^n \mapsto \mathcal{R}$ and $g : \mathcal{R}^n \mapsto \mathcal{R}^m$ are convex, twice differentiable. Having introduced a nonnegative slack variable $z \in \mathcal{R}^m$ the inequality constraint can be rewritten as an equation $g(x) + z = 0$. The inequality $z \geq 0$ is "replaced" by the logarithmic barrier terms giving the following barrier problem

$$\min \quad f(x) - \mu \sum_{i=1}^{m} \ln z_i \qquad \text{s.t.} \qquad g(x) + z = 0$$

and the associated Lagrangian

$$L(x, y, z, \mu) = f(x) + y^T(g(x) + z) - \mu \sum_{i=1}^{m} \ln z_i.$$

The first order optimality conditions for the barrier problem (conditions for a saddle point of Lagrangian) have the following form:

$$\begin{array}{rl} \nabla_x L(x, y, z, \mu) = 0 \Rightarrow & \nabla f(x) + \nabla g(x)^T y = 0 \\ \nabla_y L(x, y, z, \mu) = 0 \Rightarrow & g(x) + z = 0 \\ \nabla_z L(x, y, z, \mu) = 0 \Rightarrow & YZe = \mu e \\ & (y, z) \geq 0, \end{array} \tag{1}$$

where $Y = \text{diag}\{y_1, y_2, \cdots, y_m\}$ and $Z = \text{diag}\{z_1, z_2, \cdots, z_m\}$. Interior point algorithm for nonlinear programming [2] applies Newton method to solve this system of equations and gradually reduces the barrier parameter μ to guarantee the convergence to the optimal solution of the original problem. The Newton direction is obtained by solving the system of linear equations:

$$\begin{bmatrix} Q(x,y) & A(x)^T & 0 \\ A(x) & 0 & I \\ 0 & Z & Y \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} -\nabla f(x) - A(x)^T y \\ -g(x) - z \\ \mu e - YZe, \end{bmatrix}, \tag{2}$$

where the matrices $Q(x, y) = \nabla^2 f(x) + \sum_{i=1}^{m} y_i \nabla^2 g_i(x) \in \mathcal{R}^{n \times n}$ and $A(x) = \nabla g(x) \in \mathcal{R}^{m \times n}$ are the Hessian of Lagrangian and the Jacobian of constraints, respectively. After substituting $\Delta z = \mu Y^{-1} e - Ze - ZY^{-1}\Delta y$ in the second equation we get

$$\begin{bmatrix} -Q(x,y) & A(x)^T \\ A(x) & \Theta_D \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta y \end{bmatrix} = \begin{bmatrix} \nabla f(x) + A(x)^T y \\ -g(x) - \mu Y^{-1} e \end{bmatrix}, \tag{3}$$

where $\Theta_D = ZY^{-1}$ is a diagonal scaling matrix. The matrix involved in this set of linear equations is symmetric and indefinite. For convex optimization problem (when f and g are convex), the matrix Q is positive semidefinite.

If the variables x have a sign restriction ($x \geq 0$) the above system contains an additional primal scaling matrix $\Theta_P = XS^{-1}$ and the indefinite matrix used in (3) takes the form

$$H = \begin{bmatrix} -(Q(x,y) + \Theta_P^{-1}) & A(x)^T \\ A(x) & \Theta_D \end{bmatrix}. \tag{4}$$

In case of linear programming matrix A is constant and $Q = 0$ and in case of quadratic programming matrices A and Q are constant. LP and QP problems are usually formulated with the equality constraints and nonnegative variables hence the corresponding reduced systems take forms

$$H_{LP} = \begin{bmatrix} -\Theta_P^{-1} & A^T \\ A & \end{bmatrix} \qquad \text{and} \qquad H_{QP} = \begin{bmatrix} -(Q + \Theta_P^{-1}) & A^T \\ A & \end{bmatrix},$$

respectively. To simplify notation in the rest of this paper we will drop arguments of $A(x)$ and $Q(x,y)$ in (4) and use A and Q instead.

Standard approach [10, 11] requires 2×2 pivots be used in symmetric decomposition of matrix H. In our implementation we follow [4] and regularize matrix H to transform it to a quasi-definite matrix [12]. We add primal and dual regularizations and obtain

$$H_R = \begin{bmatrix} -Q - \Theta_P^{-1} & A^T \\ A & \Theta_D^{-1} \end{bmatrix} + \begin{bmatrix} -R_p & 0 \\ 0 & R_d \end{bmatrix},$$

where diagonal positive definite matrices $R_p \in \mathcal{R}^{n \times n}$ and $R_d \in \mathcal{R}^{m \times m}$ can be interpreted as adding proximal terms to the primal and dual objective functions, respectively. After this modification H_R becomes quasi-definite hence for any symmetric row and column permutation a triangular decomposition $H_R = LDL^T$ exists with diagonal matrix D. This matrix has exactly n negative and m positive pivots. Since 2×2 pivots do not have to be used, the symbolic and numerical factorization phases can be split as in the positive definite case. This is an important feature in the implementation of interior point methods and it is essential for the implementation of structure exploiting linear algebra techniques.

3 Linear Algebra for Block-Structured Matrices

There are many structures that could be exploited by an interior point solver. We restrict our attention to those in which Hessian and Jacobian matrices are built of blocks. The well-known elementary structures observed in Jacobian matrices A are:

$$\begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_n \\ B_1 & B_2 & \cdots & B_n & B_0 \end{bmatrix}, \begin{bmatrix} A_1 & & & & C_1 \\ & A_2 & & & C_2 \\ & & \ddots & & \vdots \\ & & & A_n & C_n \end{bmatrix}, \begin{bmatrix} A_1 & & & & C_1 \\ & A_2 & & & C_2 \\ & & \ddots & & \vdots \\ & & & A_n & C_n \\ B_1 & B_2 & \cdots & B_n & B_0 \end{bmatrix}, \tag{5}$$

and represent a primal block-angular, dual block-angular and row and column bordered structure, respectively. However, many real-life problems have more complicated nested structures that embed those and other elementary blocks. We assume that Hessian matrix Q has a closely related structure induced by the column partitioning of A.

The corresponding matrix H can be reordered leading to structures which can be exploited by a parallel factorization. Suppose, for example, that the augmented system matrix has the symmetric bordered block-diagonal structure.

$$H = \begin{bmatrix} H_1 & & & G_1^T \\ & H_2 & & G_2^T \\ & & \ddots & \vdots \\ & & & H_n \; G_n^T \\ G_1 & G_2 & \cdots & G_n \; H_0 \end{bmatrix}, \tag{6}$$

where $H_i \in \mathcal{R}^{n_i \times n_i}, i = 0, ..., n$ and $G_i \in \mathcal{R}^{n_0 \times n_i}, i = 1, ..., n$. Under the condition that H is quasi-definite (and we assume that it has been regularized to satisfy this condition), we can obtain a Cholesky-like block decomposition $H = LDL^T$, where

$$L = \begin{bmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_n \\ L_{n,1} & L_{n,2} & \cdots & L_{n,n} \; L_0 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_n \\ & & & & D_0 \end{bmatrix}$$

and

$$H_i = L_i D_i L_i^T \tag{7a}$$

$$L_{n,i} = G_i L_i^{-T} D_i^{-1} \tag{7b}$$

$$S = H_0 - \sum_{i=1}^{n} G_i H_i^{-1} G_i^T \;=\; L_0 D_0 L_0^T. \tag{7c}$$

This decomposition can be used to compute the solution to the system $Hu = b$, where $u = (u_1, \ldots, u_n, u_0)^T$, $b = (b_1, \ldots, b_n, b_0)^T$ by the following sequence of operations:

$$z_i = L_i^{-1} b_i, \quad i = 1, \ldots, n \tag{8a}$$

$$z_0 = L_0^{-1}(b_0 - \sum_{i=1}^{n} L_{n,i} z_i) \tag{8b}$$

$$y_i = D_i^{-1} z_i, \quad i = 0, \ldots, n \tag{8c}$$

$$u_0 = L_0^{-T} y_0 \tag{8d}$$

$$u_i = L_i^{-T}(y_i - L_{n,i}^T u_0), \quad i = 1, \ldots, n \tag{8e}$$

Note that blocks $L_{n,i}$ do not have to be stored. This leads to obvious memory savings: blocks $L_{n,i}$ are computed because they contribute to the Schur complement matrix S in (7c) but they do not have to be stored any longer. This also

results in time savings, since the multiplications with blocks $L_{n,i}$ and $L_{n,i}^T$ in equations (8b) and (8e) are executed as sequences of less expensive operations using the following formulae:

$$\begin{aligned} L_{n,i}z_i &= G_i L_i^{-T} D_i^{-1} z_i = G_i(L_i^{-T} D_i^{-1} z_i), \\ L_{n,i}^T u_0 &= D_i^{-1} L_i^{-1} G_i^T u_0 = D_i^{-1} L_i^{-1}(G_i^T u_0). \end{aligned}$$

Consequently, these operations have complexity $\mathcal{O}(\mathrm{nz}(G_i) + \mathrm{nz}(L_i))$ rather than $\mathcal{O}(\mathrm{nz}(L_{n,i}))$ and it is usual to expect that $\mathrm{nz}(L_{n,i}) \gg \mathrm{nz}(G_i) + \mathrm{nz}(L_i)$.

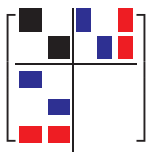
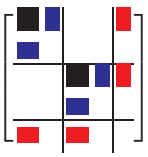
The block factorization $H = LDL^T$ together with computations (7) and (8) are therefore an *implicit* representation of the inverse of H.

Apart from the above mentioned efficiency gains the use of a block implicit inverse facilitates the parallelisation of the calculation. Indeed most of the two operations: computing the symmetric decomposition (7) and using it for solving system of equations (8) will parallelise trivially. The sums in (7c) and (8b) require parallel communications, while operations involving L_0 and D_0 (namely factorization of S and the L_0^{-1}, L_0^{-T} operations in (8b, 8d)) have to be performed on all processors.

We conclude this section by giving examples of reordered matrices H for the three common structures mentioned earlier. The reordering preserves symmetry, that is we apply the same permutation to block-rows and block-columns of H. To simplify the presentation we have made three assumptions: (i) Jacobian matrices A in (5) have only two diagonal blocks, (ii) Hessian matrices Q have block-diagonal structures induced by the block-column partitions in A, and (iii) the $(2,2)$ block in H is zero.

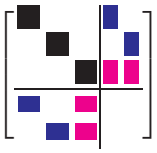
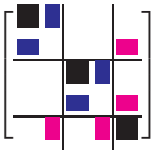
Primal Block-Angular Structure

Blocks of H have been permuted following the reordering $\{1,3;\ 2,4;\ 5\}$.

$H =$  , $PHP^T =$ 

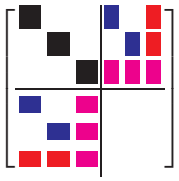
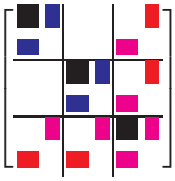
Dual Block-Angular Structure

Blocks of H have been permuted following the reordering $\{1,4;\ 2,5;\ 3\}$.

$H =$  , $PHP^T =$ 

Row and Column Bordered Block-Diagonal Structure

Blocks of H have been permuted following the reordering $\{1,4;\ 2,5;\ 3,6\}$.

$H =$  , $PHP^T =$ 

4 Object-Oriented Implementation

As shown in the previous section block-structured matrices can be reordered by permuting blocks to such forms which offer an advantage for parallel computations. Many real life problems have more complicated structures which include a nesting of elementary block-structures. The nested block-structure of a matrix can be thought of as a tree. Its root is 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 information about the type of structure this node represents. This tree determines the order and type of linear algebra operations needed by the interior point algorithm.

The design of OOPS follows object-oriented principles, treating the blocks (and sub-blocks) of matrices as objects [7, 5]. We use a `Matrix` interface that defines all linear algebra methods needed for an interior point algorithm such as `Factorize`, `solveL` or `solveLt`. The interface also provides all operations required for a given `Matrix` object to become a sub-matrix in the nested structured matrix. In such case `Matrix` object is accessed by the `Matrix` object corresponding to its ancestor in the tree defining the nested structure.

Several specialised classes provide concrete implementation of the `Matrix` interface, each exploiting a different possible structure such as for example *primal block-angular*, *dual block-angular*, *bordered block-diagonal*, *rank corrector* as well as a standard *dense matrix* or *sparse matrix*. The implementing classes can be

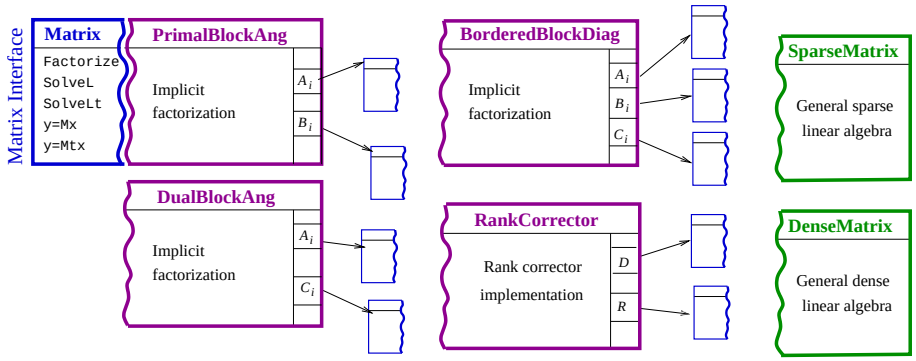


Fig. 1. The matrix interface and several implementations of it.

classified as either *leaf-node* classes such as `DenseMatrix` or `SparseMatrix` or the *complex* classes, such as `PrimalBlockAng`, `DualBlockAng`, `RankCorrector` or `BorderedBlockDiag` (see Figure 1). The design of OOPS library is based on the assumption that an efficient implementation of all methods for a *complex* class can be reduced to a sequence of methods performed on its constituents [7, 5]. The top-level class here does not need to know the exact type of its constituent

objects nor whether they themselves are of *leaf-node* or *complex* type, it merely needs to know that they support the methods of the interface and assumes that they do so in a way most efficient for their particular structure. Summing up, OOPS re-creates the structured matrix tree with a tree of `Matrix` objects.

5 Large-Scale Portfolio Optimization Problems

To demonstrate the efficiency of OOPS we have applied it to solve a class of portfolio optimization problems. We follow a description of these problems given in [9] and consider extensions [13] which allow for higher order moments to be incorporated into the model (for more details see [6, 8]).

We consider investment of an initial wealth b into assets $j = 1, \ldots, J$ with uncertain returns. We allow the portfolio be be rebalanced at discrete times $t = 1, \ldots, T$ and we want (i) to maximize the expected final wealth of the portfolio at time T, and (ii) to minimize the associated risk. The standard formulation of this problem leads to Markowitz portfolio optimization problem [14, 15].

The stochastic process is approximated by a discrete distribution and modelled as an event tree. With each node of the tree i we associate the time stage t it belongs to, the ancestor node $a(i)$, and the list of successor nodes (children) which belong to the next time stage $t+1$. The probability of reaching node i is denoted by p_i and it is equal to the product of probabilities associated with all arcs in the the event tree leading from the root node to node i.

At every node i and for each asset j we define three variables $x^h_{i,j}, x^b_{i,j}$ and $x^s_{i,j}$ which denote the amount of asset held, bought and sold, respectively. The *inventory constraint* for an asset i writes:

$$(1 + r_{i,j})x^h_{a(i),j} = x^h_{i,j} - x^b_{i,j} + x^s_{i,j}, \quad \forall i \neq 0, j, \tag{9}$$

where $r_{i,j}$ is the return associated with a branch (arc of event tree) connecting ancestor node $a(i)$ with i. The *budget constraint* imposes an equality of cash inflow from selling assets and cash outflow for buying new assets:

$$\begin{array}{l} \sum_j (1 + c_t) v_j x^b_{i,j} = \sum_j (1 - c_t) v_j x^s_{i,j} \quad \forall i \neq 0 \\ \sum_j (1 + c_t) v_j x^b_{0,j} = b, \end{array} \tag{10}$$

where v_j is a unit price of asset j and c_t is a transaction cost. This constraint takes a simplified form for the root node $i = 0$ where one can only purchase assets of total value equal to the initial budget b.

In the standard approach the wealth is measured by an expected value of the final portfolio converted into cash

$$y = I\!E((1 - c_t) \sum_{j=1}^{J} v_j x^h_{T,j}) = (1 - c_t) \sum_{i \in L_T} p_i \sum_{j=1}^{J} v_j x^h_{i,j}, \tag{11}$$

where L_T denotes the subset of nodes in the event tree which belong to the terminal stage. It is common to use the variance of return as a risk measure:

$$r \;=\; \mathrm{Var}((1-c_t)\sum_{j=1}^{J} v_j x^h_{T,j}) = \sum_{i\in L_T} p_i[(1-c_t)\sum_j v_j x^h_{i,j} - y]^2. \qquad (12)$$

The min-variance portfolio minimizes the aggregate objective of the form $y-\lambda r$ which combines two criteria into a single one and uses an arbitrary parameter λ to express the wealth-risk trade-off. The larger the λ the more attention is paid to risk hence more of safe assets are selected into the portfolio.

Summing up, the standard multi-stage Markowitz portfolio optimization problem consists in minimizing $y-\lambda r$ subject to constraints (9), (10), (11) and (12). Models require all decision variables $x^h_{t,j}, x^b_{t,j}, x^s_{t,j}$ be nonnegative and may impose additional constraints on portfolio selection. This standard model leads to a quadratic programming problem. It is a challenging problem because the event tree corresponding to a multi-stage problem is usually very large (it grows exponentially with the number of stages). Various extensions to the standard model which do not change the underlying structure were given in [6, 8].

6 Numerical Results

In this section we present computational results of our approach.

We have compared the performance of OOPS with that of the commercial code CPLEX 9.1. Since we do not possess a parallel CPLEX license these results are from runs on a serial 3GHz Linux PC with 2GB of memory. We summarize our findings in Table 1. As can bee seen OOPS needs consistently less memory than CPLEX. CPLEX actually fails to solve problem C70 due to running out of memory (OoM). In this case we give an estimate solution time based on the number of flops reported from its symbolic Cholesky factorization. The smallest problem C33 is solved slightly faster by CPLEX, while for larger problems OOPS becomes much more efficient than CPLEX. We demonstrate the parallel efficiency of our code on a massively parallel environment. All computations were performed on the BlueGene (BlueSky) service at Edinburgh Parallel Computing Centre (EPCC). This machine has 1024 nodes each of which comprises 2 IBM-PowerPC-440 processors running at 700Mhz and 512MB of RAM. In our experiments we run the machine in co-processor mode, that is the two processors

prob	vars	cons	f.s.d.	CPLEX 9.1		OOPS	
				time	mem	time	mem
C33	168.451	57.274	33	292	497MB	344	156MB
C50	382.801	130.153	50	1361	1.3GB	828	345MB
C70	745.651	253.522	70	(5254)	OoM	1627	664MB

Table 1. Comparison of OOPS with CPLEX 9.1.

on each node are split into a computing and a communicating unit, with all the memory available to the computing unit.

On the BlueGene service the memory is local to each node. A problem that just solves on the available memory on n processors is therefore likely to run out of memory on $n/2$ processors. To circumvent this difficulty we present our results in two series of problems. The first comprises the biggest problem we have been able to solve on BlueGene: an ALM problem with 6 stages arranged in an event tree of dimension 128x24x16x10x5x4 resulting in a total of 12.831.873 scenarios considered. This problem has just over 500 million variables. When solved on 1024 nodes, each node works with a decision tree of dimension 3x16x10x5x4 or 12532 scenarios, leading to a sparse system matrix of size 175.448×488.748. We have solved variations of this problem A16 through A1024 on $16 - 1024$ nodes where each node works with the same sub-tree as for the big problem, but the total problem has fewer first stage decisions (f.s.d.).

Prob	f.s.d.	constraints	variables	nz(A)	nz(L)	peak Mem
A16	2	2.806.987	7.819.462	15.638.922	118.774.704	260MB
A32	4	5.613.959	15.638.884	31.277.766	237.549.408	260MB
A64	8	11.227.903	31.277.728	62.555.454	475.098.816	264MB
A128	16	22.455.791	62.555.416	125.110.830	950.197.632	264MB
A256	32	44.911.567	125.110.792	250.221.582	1.900.395.264	268MB
A512	64	89.823.133	250.221.582	500.443.086	3.800.790.528	276MB
A1024	128	179.646.223	500.443.048	1.000.886.094	7.601.581.056	292MB
B1280	128	352.875.799	1.010.507.968	2.021.015.944	18.869.419.008	661MB

Table 2. Problem Dimensions.

For completeness we have also included the details of our largest problem in the series solved at all (B1280). This example has a slightly larger event tree and almost twice as many variables per scenario as A1024. Due to larger memory requirements it could not be solved on BlueGene, but was solved on the 1600 1.7GHz-processor HPCx service instead. Due to limited allocation of computing resources on this system we are unable to provide further details for this problem. Problem sizes for this series are summarized in Table 2. Columns

Prob	nodes	time(20iters)	peak mem/node	generation	communication	rest
A16	16	1815	260MB	6	26	1783
A32	32	1845	260MB	12	51	1782
A64	64	1911	264MB	23	102	1786
A128	128	2050	264MB	45	206	1799
A256	256	2289	268MB	89	416	1784
A512	512	2797	276MB	178	825	1794
A1024	1024	3818	294MB	361	1666	1791
B1280	1280	1139 (HPCx)	661MB	-	-	-

Table 3. Solution Statistics and breakdown by parts of algorithm.

$\mathrm{nz}(A), \mathrm{nz}(L)$ are the numbers of nonzeros in the system matrix and the implicit inverse of the augmented system, respectively. Column *peak Mem* is the peak

Memory used per node by our implementation. As can be seen the number of nonzeros in the implicit inverse grows linearly in the problem size and hence the memory per node stays roughly the same. The memory increase for higher number of processors is due to the local $\mathcal{O}(nodes^2)$ memory requirements of communication routing tables.

Table 3 gives run times for the first 20 iterations of each problem. Due to the changing topology of the scenario tree between problems our IPM takes different numbers of iterations to reach optimality. We have therefore truncated our benchmark runs after 20 iterations. We have also ensured that the same numbers of centrality correctors were used in each runs, so that results are comparable. The largest problems A1024 and B1280 are solved to optimality in 45 and 53 iterations respectively. To interpret the results note that in the setup of this series the calculations done on each processor for (7, 8) are the same for all the problems. Computation time should therefore not increase with problem size. Indeed this is demonstrated by the results where the only time increase is due to (not entirely parallelisable) problem generation and parallel communications.

Our second series of experiments comprises a 64x24x16x10 decision tree problem, resulting in a total of 271.936 scenarios and a system matrix of size $3.807.119 \times 10.605.544$. This problem is solvable within the available memory on 16 nodes, while the tree architecture should enable efficient parallelisation to up to 512 nodes. As can be seen in Table 4 the algorithm parallelises well,

nodes	peak Mem	time	Comm	Cholesky	Solves	MatVectProd
16	426MB	2587 (1.00)	24	1484 (1.00)	956 (1.00)	28.8 (1.00)
32	232MB	1303 (0.99)	13	743 (1.00)	485 (0.98)	18.0 (0.80)
64	132MB	688 (0.94)	6	377 (0.98)	270 (0.88)	13.0 (0.55)
128	84MB	348 (0.93)	3	187 (0.99)	139 (0.86)	9.0 (0.40)
256	56MB	179 (0.90)	3	93 (0.99)	73 (0.82)	5.8 (0.31)
512	46MB	94 (0.86)	2	47 (0.98)	39 (0.76)	3.9 (0.23)

Table 4. Second series of results.

reaching a parallel efficiency of 0.86 on 512 compared with 16 processors. If the time spent by the algorithm is broken down, we see that communications and problem generation (not reported because it was below 1s) are less of an issue than for the first series due to smaller problem size. The factorization parallelises virtually perfectly (the only non-parallel bit, the factorization of S being negligible). The backsolves (8) parallelise fairly well, while the worst efficiency is obtained from matrix vector products (mainly needed to obtain primal-dual residuals), but these do not contribute much to the overall performance.

7 Conclusions

We have demonstrated in this paper that block-structure of matrices can be exploited by an interior point algorithm. IPMs work with indefinite systems which can be transformed to quasi-definite ones by adding regularization terms. After this transformation an arbitrary symmetric reordering of the indefinite

matrix can be used and a symmetric decomposition can be computed which does not need 2×2 pivots be used. Hence full advantage of the block-structure in the matrix can be taken and by exploitation of block-operations a high degree of parallelism can be achieved. Indeed, we have demonstrated that a modern implementation of interior point method run on massively parallel computer displays good parallel efficiency. Eventually, this allowed us to solve optimization problems of dimensions reaching one billion variables.

Acknowledgements
We are grateful to the EPCC for allowing us to use the BlueGene service and to Dr Joachim Hein in particular for his help in running OOPS on this machine.

References

1. Karmarkar, N.K.: A new polynomial–time algorithm for linear programming. Combinatorica **4** (1984) 373–395
2. Wright, S.J.: Primal-Dual Interior-Point Methods. SIAM, Philadelphia (1997)
3. Andersen, E.D., Gondzio, J., Mészáros, C., Xu, X.: Implementation of interior point methods for large scale linear programming. In Terlaky, T., ed.: Interior Point Methods in Mathematical Programming. Kluwer Acad Pub (1996) 189–252
4. Altman, A., Gondzio, J.: Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization. Optimization Methods and Software **11-12** (1999) 275–302
5. Gondzio, J., Sarkissian, R.: Parallel interior point solver for structured linear programs. Mathematical Programming **96**(3) (2003) 561–584
6. Gondzio, J., Grothey, A.: Parallel interior point solver for structured quadratic programs: Application to financial planning problems. Technical Report MS-03-001, School of Mathematics, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK (2003) Accepted for publication in *Annals of Operations Research*.
7. Gondzio, J., Grothey, A.: Exploiting structure in parallel implementation of interior point methods for optimization. Technical Report MS-04-004, School of Mathematics, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK (2004)
8. Gondzio, J., Grothey, A.: Solving nonlinear portfolio optimization problems with the primal-dual interior point method. Technical Report MS-04-001, School of Mathematics, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK (2004) Accepted for publication in *European Journal of Operational Research*.
9. Ziemba, W.T., Mulvey, J.M.: Worldwide Asset and Liability Modeling. Publications of the Newton Institute. Cambridge University Press, Cambridge (1998)
10. Arioli, M., Duff, I.S., de Rijk, P.P.M.: On the augmented system approach to sparse least-squares problems. Numerische Mathematik **55** (1989) 667–684
11. Duff, I.S., Erisman, A.M., Reid, J.K.: Direct methods for sparse matrices. Oxford University Press, New York (1987)
12. Vanderbei, R.J.: Symmetric quasidefinite matrices. SIAM Journal on Optimization **5** (1995) 100–113
13. Konno, H., Shirakawa, H., Yamazaki, H.: A mean-absolute deviation-skewness portfolio optimization model. Annals of Operational Research **45** (1993) 205–220
14. Markowitz, H.M.: Portfolio selection. Journal of Finance (1952) 77–91
15. Steinbach, M.: Markowitz revisited: Mean variance models in financial portfolio analysis. SIAM Review **43**(1) (2001) 31–85