School of Mathematics

An Object-Oriented Parallel Interior Point Solver for Structured Nonlinear Programming Problems

Andreas Grothey, Jacek Gondzio

Design Philosophy

Truely large-scale optimization problems are usually

- sparse
- block-structured
- often have nested structure

(due to e.g. dynamics, uncertainty, spatial distribution etc.)

We assume that structure is known! $\Rightarrow$ no automatic detection.

Exploiting structure and inherent parallelism is key to efficient implementation of IPMs:

- Faster linear algebra
- Reduced memory use
- Parallel machines often cheaper to install than comparable serial machine: e.g. cluster of PCs running Linux.

OOPS: Object-Oriented Parallel interior point Solver

- Use Interior Point Methods, because:
  - they are predictable (number of iterations $\mathcal{O}(\log n)$)
  - they can take advantage of the problem structure
  - their linear algebra operations are parallelisable
- Exploiting structure & parallelism:
  - need linear algebra dedicated to problem structure
  - parallel implementation needs to be taylored to specific problem.

    $\Rightarrow$ Need implementation for every different type of problem ?

$\Rightarrow$ Use Object-Oriented linear algebra implementation

Solving QP by Interior Point Method

$$\min c^\top x + \tfrac{1}{2} x^\top Q x \quad \text{s.t.} \quad \begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned} \tag{QP}$$

Optimality conditions:

$$\begin{aligned} c + Qx - A^\top y - z &= 0 \\ Ax &= b \\ XZe &= 0 \quad (\mu e) \\ x, z &\geq 0 \end{aligned}$$

$\Rightarrow$ Newton Step:

$$\begin{bmatrix} -Q - \Theta & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} Qx + c - A^\top y - \mu X^{-1} e \\ b - Ax \end{bmatrix} \tag{NS-QP}$$

where

$$\Theta = X^{-1} Z, \quad X = \mathrm{diag}(x), \quad Z = \mathrm{diag}(z)$$

Solving NLP by Interior Point Method

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

Add slacks to nonlinear inequalities:

$$\begin{aligned} \min f(x) \;\text{ s.t. }\; g(x) + z &= 0 \\ z &\geq 0 \end{aligned} \qquad \text{(NLP)}$$

Optimality conditions:

$$\begin{aligned} \nabla f(x) + \nabla g(x)^\top y &= 0 \\ g(x) + z &= 0 \\ YZe &= 0 \quad (\mu e) \\ x, z &\geq 0 \end{aligned}$$

$\Rightarrow$ Newton Step:

$$\begin{bmatrix} Q(x,y) & A(x)^\top \\ A(x) & -\Theta \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -\nabla f(x) - A(x)^\top y \\ -g(x) - \mu Y^{-1} e \end{bmatrix} \qquad \text{(NS-NLP)}$$

where

$$Q(x,y) = \nabla^2_{xx}(f(x) + y^\top g(x)), \quad A(x) = \nabla g(x)$$
$$\Theta = ZY^{-1}, \quad Y = \operatorname{diag}(y), \quad Z = \operatorname{diag}(z)$$

Linear Algebra of IPMs

Main work: solve

$$\underbrace{\begin{bmatrix} -Q-\Theta & A^\top \\ A & 0 \end{bmatrix}}_{\Phi} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix} \qquad \text{or} \qquad \underbrace{\begin{bmatrix} -Q & A^\top \\ A & \Theta \end{bmatrix}}_{\Phi} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix}$$

for several right-hand-sides at each iteration

$\Rightarrow$ Two stage solution procedure

- factorize $\Phi = LDL^\top$
- backsolve(s) to compute direction $(\Delta x, \Delta y)$ + corrections

$\Rightarrow$ Φ changes numerically but not structurally at each iteration

Key to efficient implementation is exploiting structure of Φ in these two steps

Structured Matrices

- Bordered Block-Diagonal Structure:

$$\Phi = \begin{pmatrix} A_1 & & & B_1^\top \\ & \ddots & & \vdots \\ & & A_n & B_n^\top \\ B_1 & \cdots & B_n & B_{n+1} \end{pmatrix}$$

- Block Tri-Diagonal Structure:

$$\Phi = \begin{pmatrix} A_1 & B_1^\top & & \\ B_1 & \ddots & \ddots & \\ & \ddots & \ddots & B_{n-1}^\top \\ & & B_{n-1} & A_n \end{pmatrix}$$

- Rank-corrector:
  $\Phi = D + RR^\top$, D easily invertible, R small number of columns
- Special Matrices (Network, Projection etc.)
- General Sparse Matrix

Structures may also be nested

Example: Bordered Block-Diagonal Structure (Schur complement)

$$\underbrace{\begin{pmatrix} \Phi_1 & & & B_1^\top \\ & \ddots & & \vdots \\ & & \Phi_n & B_n^\top \\ B_1 & \cdots & B_n & \Phi_0 \end{pmatrix}}_{\Phi} = \underbrace{\begin{pmatrix} L_1 & & & \\ & \ddots & & \\ & & L_n & \\ L_{1,0} & \cdots & L_{n,0} & L_0 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} D_1 & & & \\ & \ddots & & \\ & & D_n & \\ & & & D_0 \end{pmatrix}}_{D} \underbrace{\begin{pmatrix} L_1^\top & & & L_{1,0}^\top \\ & \ddots & & \vdots \\ & & L_n^\top & L_{n,0}^\top \\ & & & L_0^\top \end{pmatrix}}_{L^\top}$$

- Cholesky-like factors can be obtained by Schur-complement:

$$\begin{aligned} \Phi_i &= L_i D_i L_i^\top & L_{i,0} &= B_i L_i^{-\top} D_i^{-1},\ i = 1, \ldots n \\ C &= \Phi_0 - \textstyle\sum_{i=1}^n L_{i,0} D_i L_{i,0}^\top & C &= L_0 D_0 L_0^\top \end{aligned}$$

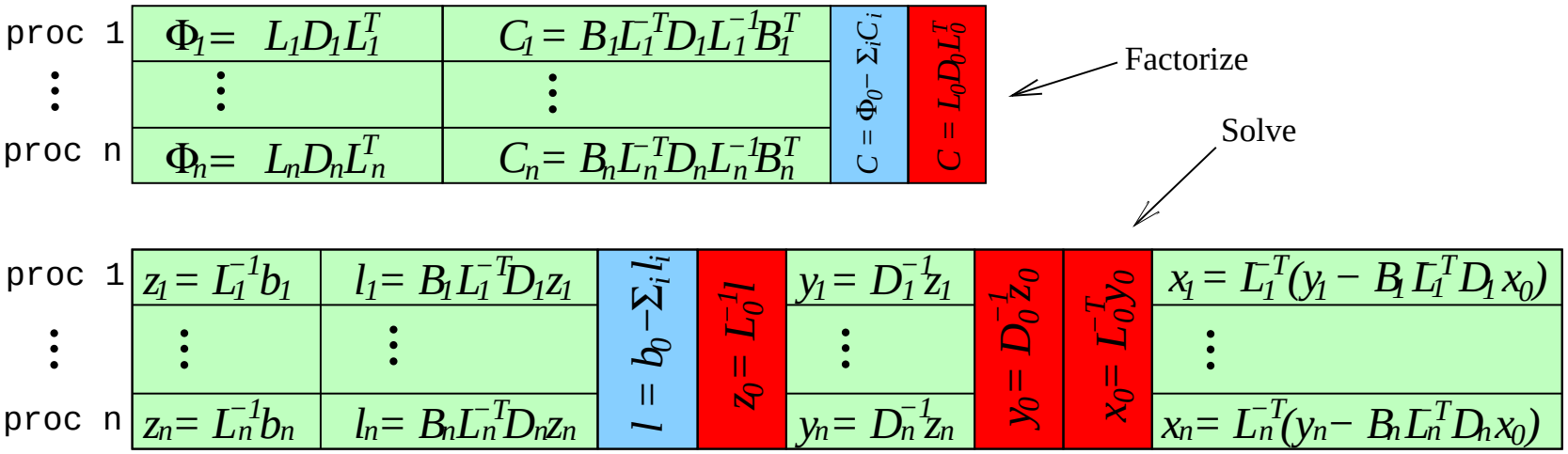
- And the system $\Phi x = b$ can be solved by

$$\begin{aligned} z_i &= L_i^{-1} b_i \\ z_0 &= L_0^{-1}(b_0 - \textstyle\sum L_{i,0} z_i) \\ y_i &= D_i^{-1} z_i \end{aligned} \qquad \begin{aligned} x_0 &= L_0^{-\top} y_0 \\ x_i &= L_i^{-\top}(y_i - L_{i,0}^\top x_0) \end{aligned}$$

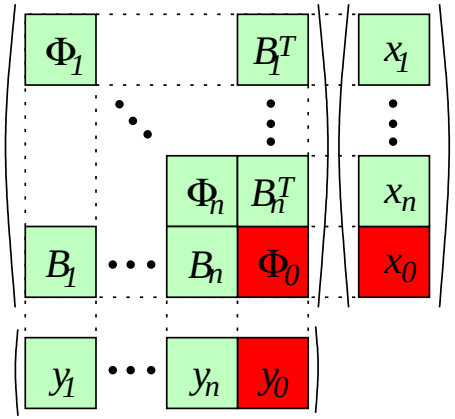
- Operations (Cholesky, Solve, Product) are only performed on sub-blocks
  $\Rightarrow$ Can also exploit structure in sub-blocks

Exploiting Parallelism: Bordered Block-Diagonal Structure:

- Distribution of computations:



- Storage:



processor 1

processor n

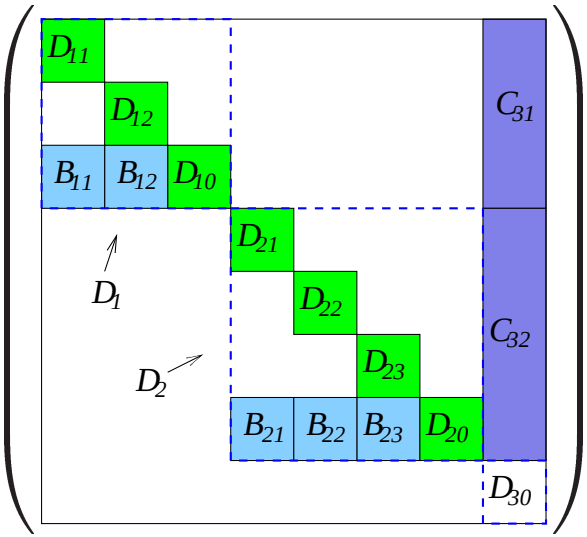
all processors

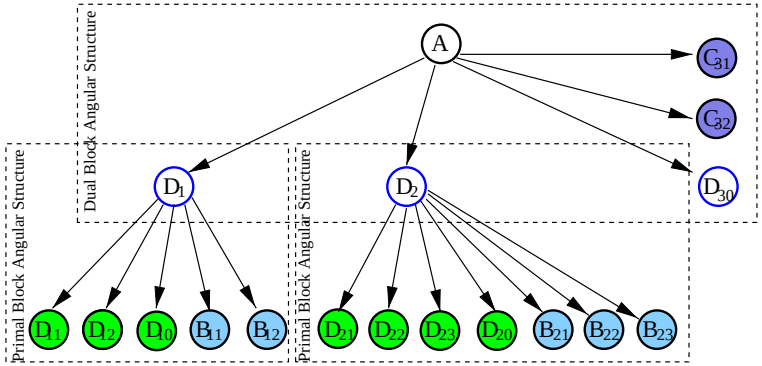
On separate processors

On all processors

Communications

Tree representation of matrices:





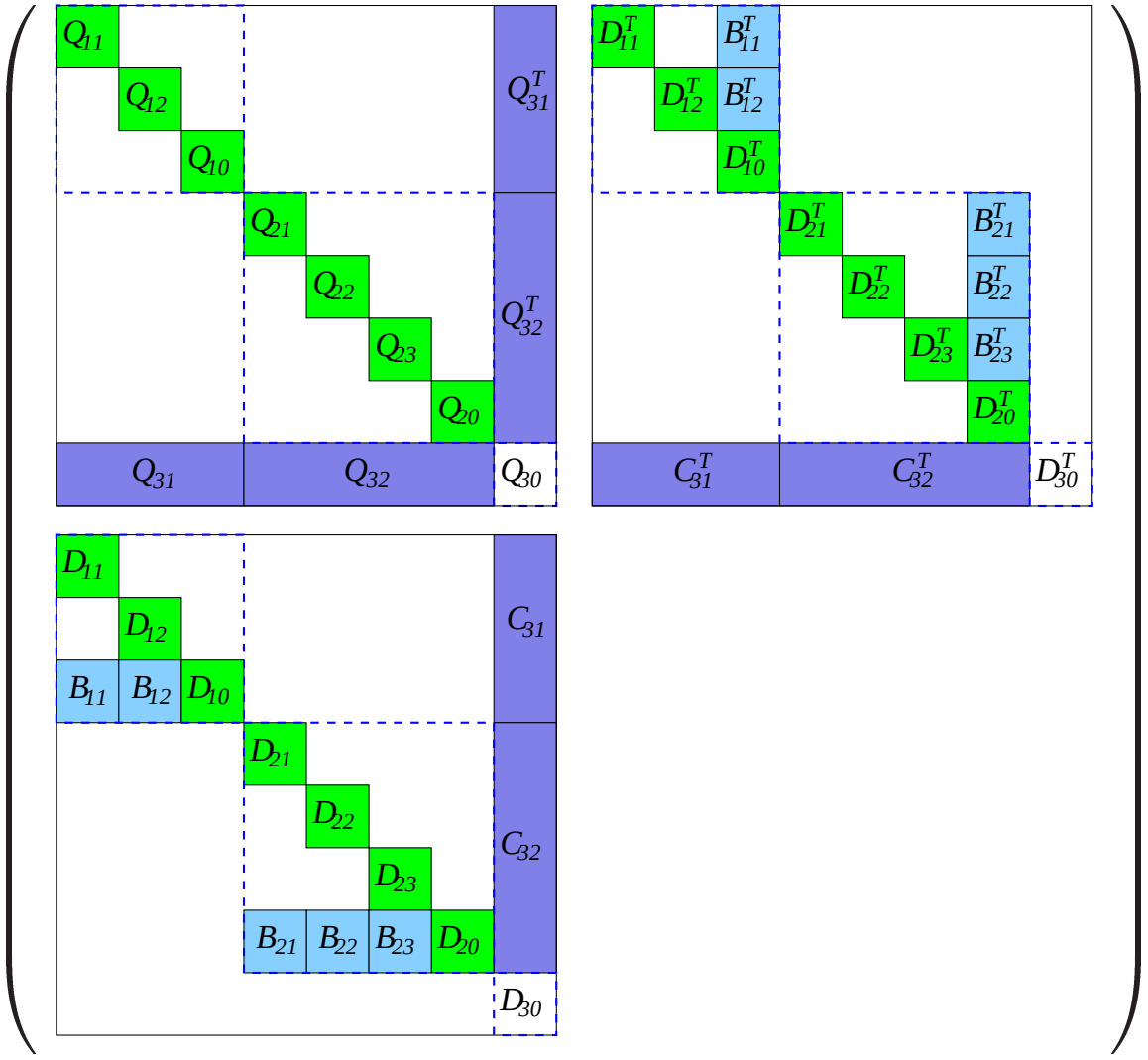
Structure Exploiting Linear Algebra

Every block of A, Q, Φ should have special structure exploiting linear algebra

- Blocks may be nested
- All blocks may be of different structure

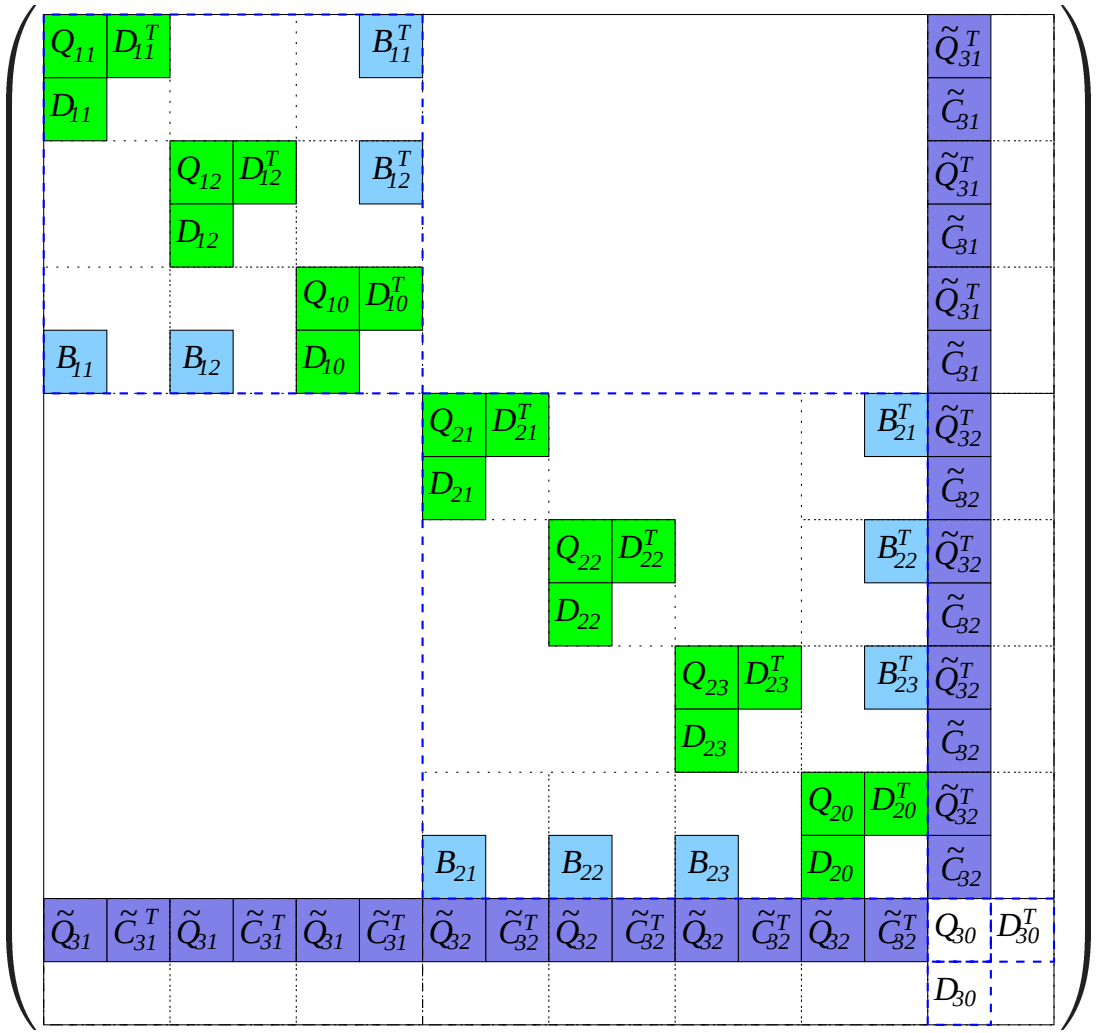
Structures of A and Q imply structure of Φ:

$$\begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} =$$

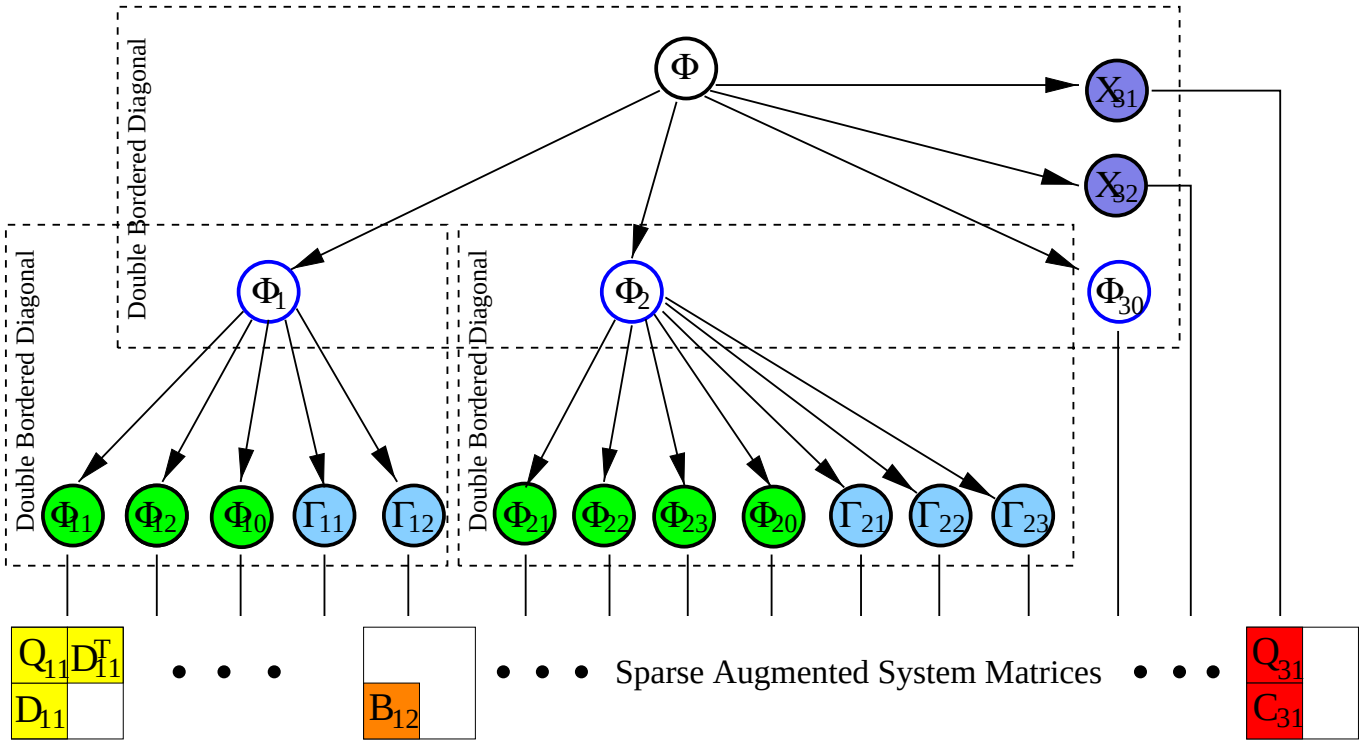


Structures of A and Q imply structure of Φ:

$$\Phi = P \begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} P^{-1} =$$



Tree Representation of Augmented Matrix



- Same tree as for A, but nodes are augmented system matrices
- Blocks can be reordered automatically by re-assigning pointers

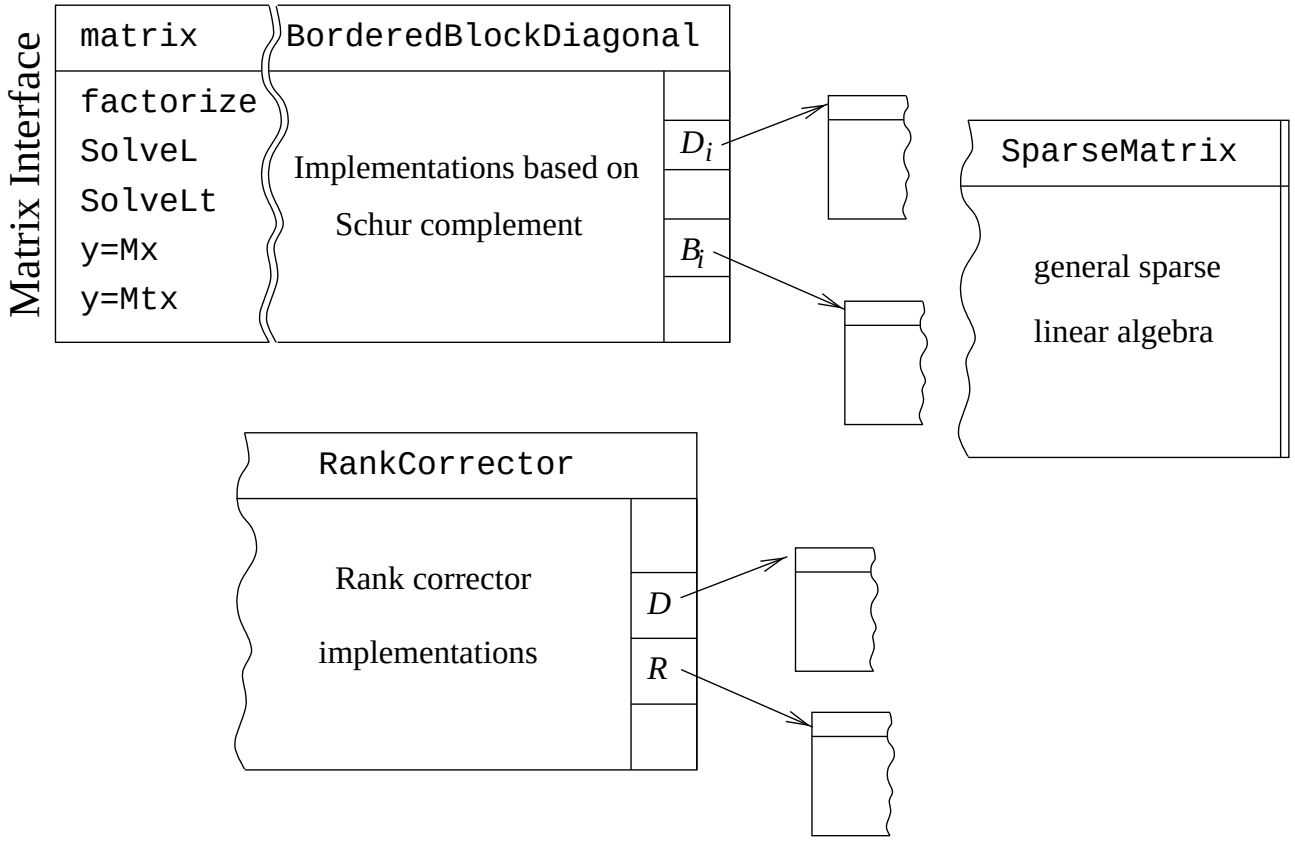
Object-oriented approach

Every structure needs its own special Linear Algebra implementation
$\Rightarrow$ prohibitive coding cost

Better: Object Oriented approach

- Every block of a matrix is implementations of abstract `matrix` interface
- `matrix` objects M can be accessed from outside by their `methods`, i.e.
  - Factorize M
  - Solve L, Solve D, Solve $L^\top$
  - Mx, $M^\top x$
- Each `matrix` object has its own implementation for these methods (most efficient for its structure, exploiting parallelism)
- allow self-referencing

Object-oriented approach:



Rebuild tree with matrix interface structures

`matrix` interface

Set of supported methods ($\mathcal{M}$):

- Factorize $M = LDL^\top$
- Solve L, Solve D, Solve $L^\top$
- Matrix-Vector products: Mx, $M^\top x$
- retrieve column/row from M
- set up $\Phi = P \begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} P^{-1}$

closure condition:

Any method in $\mathcal{M}$ can be implemented for any supported structure by only using methods in $\mathcal{M}$ on its sub-blocks

Advantages of Object-Oriented approach:

- One implementation for each type of block: not for each block
- Separates logic of IPM from linear algebra
  $\Rightarrow$ Always efficient linear algebra, independent of algorithm
- Automatic coarse grain parallelism
- Linear Algebra module can be used outside IPM

Extension to NLP

- Linear Algebra can be used in Interior Point NLP solver.
- Can wrap QP solver in SQP framework
  $\Rightarrow$ Issue of efficient warm-start of QP arises. current work
- or use native Interior Point NLP solver. current work

Example: Asset and Liability Management Problem - ALM

- A set of assets $\mathcal{J} = \{1, ..., J\}$ is given (e.g. bonds, stock, real estate).
- At every stage $t = 0, ..., T-1$ we can buy or sell different assets.
- The return of asset j at stage t is *uncertain* (but distribution is known).

We have to make investment decisions:
what to buy or sell, at which time stage

Objectives:

- maximize the final wealth
- minimize the associated risk

Asset and Liability Management Problem - II

Parameters:

v_j	value of j-the asset	$r_{i,j}$	(stochastic) return of asset j at node i
p_i	probability of node i	c_t	transaction cost
b	initial cash	L_t	final stage nodes

Decision variables:

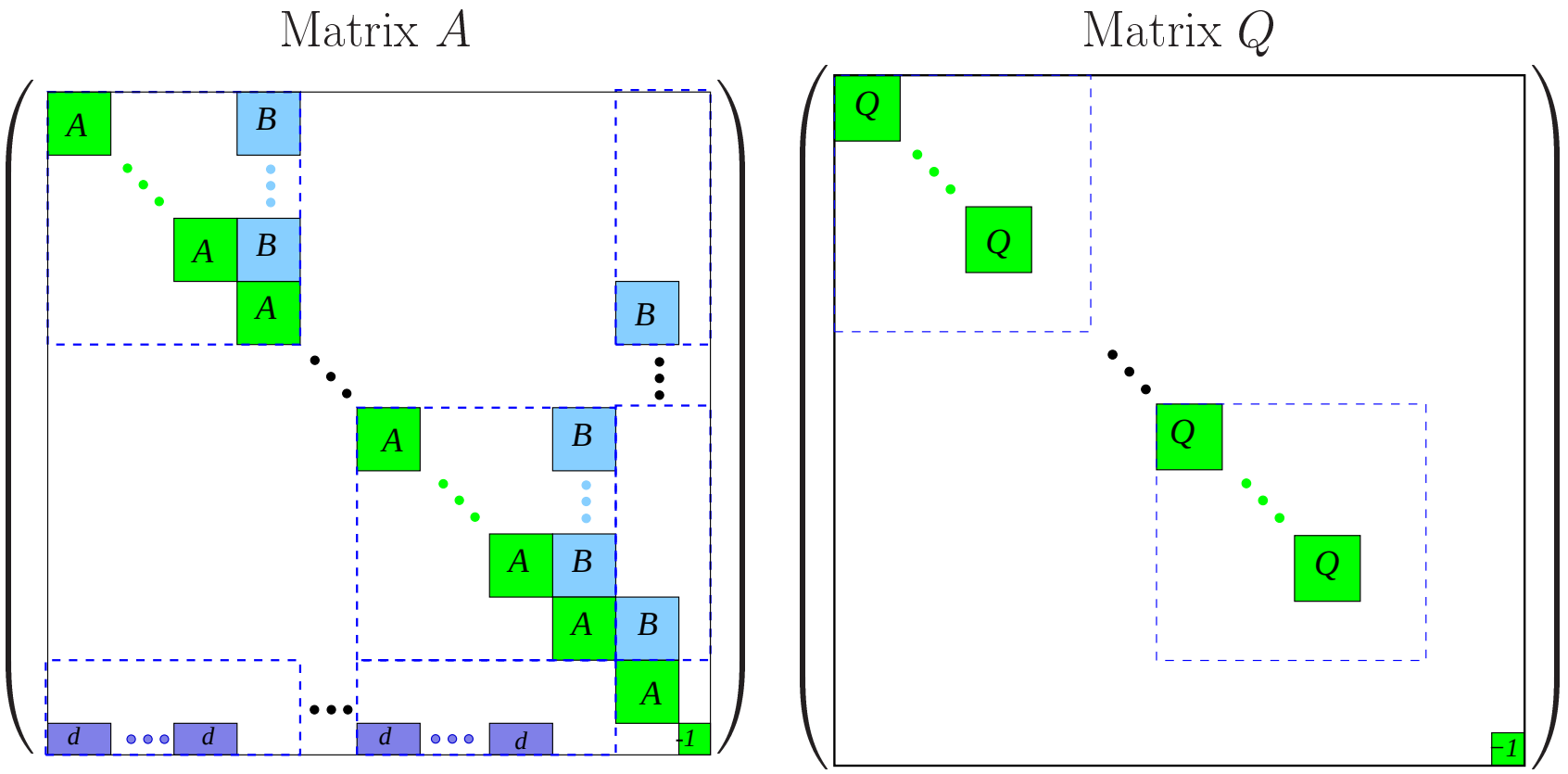
$x^h_{i,j}, x^s_{i,j}, x^b_{i,j}$	units of asset j held, sold, bought in node i
y	expected return

The ALM problem can then be expressed as

$$\max_{x,y\geq 0} \quad y - \rho[\sum_{i\in L_T} p_i[(1-c_t)\sum_j v_j x^h_{j,i}]^2 - y^2] \qquad (\mathbb{E}(X) - \rho[\mathbb{E}(X^2) - \mathbb{E}(X)^2])$$

$$\begin{aligned}
\text{s.t.} \quad (1+r_{j,i})x^h_{j,a(i)} &= x^h_{j,i} - x^b_{j,i} + x^s_{j,i}, \quad \forall i \neq 0, j && \text{Inventory}\\
\textstyle\sum_j (1+c_t)v_j x^b_{j,i} &= \textstyle\sum_j (1-c_t)v_j x^s_{j,i}, \quad \forall i \neq 0 \\
\textstyle\sum_j (1+c_t)v_j x^b_{j,0} &= b && \left.\begin{array}{l}\\ \\ \end{array}\right\}\text{Cash Balance}\\
(1-c_t)\textstyle\sum_{i\in L_T} p_i \sum_j v_j x^h_{j,i} &= y. && \text{Expectation}
\end{aligned}$$

ALM: Structure of matrices A and Q:



Results (ALM: nonconvex QP formulation):

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
ALM6	4	120	5	1742521	10.455.127	26.137.816
ALM7	4	120	10	1742521	19.167.732	52.275.631

Problem	1 proc		2 procs		4 procs		5 procs		6 procs		8 procs	
	t(s)	it	t(s)	su	t(s)	su	t(s)	su	t(s)	su	t(s)	su
ALM1	86	14	45	1.91			18	4.78				
ALM2	1933	22	968	1.99			387	4.99				
ALM3	8172	29	4102	1.99			1638	4.99				
ALM4	6435	33	3221	2.00	1642	3.92			1092	5.89	856	7.52
ALM5	8958	18	4566	1.96	2286	3.92					1195	7.50
UNS1	6119	27	3260	1.88	1633	3.76	1312	4.66	1092	5.60	872	7.02
ALM6	1470	18	on 16 processors									
ALM7	8465	19	on 16 processors									

24 750MHz UltraSparc-III processors, 48GB of shared memory

Results (ALM QP: comparison with CPLEX): *QPs have been convexified*

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
ALM8a	4	10	50	1111	56.662	166.651
ALM8b	3	33	50	1123	57.274	168.451
ALM8c	3	50	50	2552	130.153	382.801
ALM8d	3	70	50	4971	253.522	745.651

Problem	CPLEX 7.0			OOPS		
	time (s)	iter	mem (Mb)	time (s)	iter	mem (Mb)
ALM1	215	36	202	231	14	91
ALM2	3107	51	1859	4570	26	922
ALM8a	1317	29	452	1196	14	258
ALM8b	2838	31	637	368	16	142
ALM8c	10910	29	1590	860	16	319
ALM8d	~51000	??	Out of Mem	1723	17	678

400MHz UltraSparc-II processor, 2GB of memory

Results (ALM QP: comparison with CPLEX): *QPs have been convexified*

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
ALM8a	4	10	50	1111	56.662	166.651
ALM8b	3	33	50	1123	57.274	168.451
ALM8c	3	50	50	2552	130.153	382.801
ALM8d	3	70	50	4971	253.522	745.651

Problem	CPLEX 7.0			OOPS		
	time (s)	iter	mem (Mb)	time (s)	iter	mem (Mb)
ALM1	215	36	202	231	14	91
ALM2	3107	51	1859	4570	26	922
ALM8a	1317	29	452	1196	14	258
ALM8b	2838	31	637	368	16	142
ALM8c	10910	29	1590	860	16	319
ALM8d	~51000	??	Out of Mem	1723	17	678

400MHz UltraSparc-II processor, 2GB of memory

Extensions to NLP: (s_i^+, s_i^- measure derivation from $\mathbb{E}(X)$ for each scenario)

Standard Markowitz formulation:

$$\max\ y - \sigma \textstyle\sum_{i\in L_t} p_i (v^\top x_i - y)^2 \quad \text{s.t.} \quad \begin{array}{rrcll} (C1) & \sum_{i\in L_t} p_i v^\top x_i - y & = & 0 & \\ (C2) & B x_{a(i)} - A x_i & = & 0, & i \neq 0 \\ (C3) & A x_0 & = & b & \end{array} \quad \text{(QP)}$$

Downside risk constrained:

$$\max\ y \quad \text{s.t.} \quad \begin{array}{rcll} \sum_{i\in L_t} p_i (s_i^+)^2 & \leq & \rho & \\ v^\top x_i + s_i^+ - s_i^- - y & = & 0, & i \in L_T \end{array} \quad \text{(NLP)}$$

$(C1) - (C3)$

Nonlinear utility function:

$$\max\ \log(1+y) \quad \text{s.t.} \quad \begin{array}{rcll} \sum_{i\in L_t} p_i (s_i^+)^2 & \leq & \rho & \\ v^\top x_i + s_i^+ - s_i^- - y & = & 0, & i \in L_T \end{array} \quad \text{(NLP)}$$

$(C1) - (C3)$

Skewness formulation:

$$\max\ y + \gamma \sum_{i\in L_t} p_i (s_i^+ - s_i^-)^3 \quad \text{s.t.} \quad \begin{array}{rcll} \sum_{i\in L_t} p_i ((s_i^+)^2 + (s_i^-)^2) & \leq & \rho & \\ v^\top x_i + s_i^+ - s_i^- - y & = & 0, & i \in L_T \end{array} \quad \text{(NLP)}$$

$(C1) - (C3)$

$\Rightarrow$ Solver allows flexibility in modelling

Results: NLP formulation: SQP type approach + naive warmstart

Problem	Stages	Blocks	Assets	Total Nodes	constraints	variables
NLP4	3	70	40	4971	208,713	606,322
NLP5	4	24	25	14425	388,876	1,109,525
NLP6	4	40	50	65641	3,411,693	9,974,152
NLP7	4	55	20	169456	3,724,953	10,500,112

Semi-variance constrained (one quadratic constraint):

Problem	1 proc		2 procs		4 procs		8 procs	
	iter	time(s)	time(s)	su	time(s)	su	time(s)	su
NLP4	35	568	258	2.20	141	4.02	92	6.11
NLP5	30	1073	516	2.08	254	4.21	148	7.27
NLP6	41	17764	9028	1.97	4545	3.91	2390	7.43
NLP7	43	18799	9391	2.00	4778	3.93	2459	7.64

24 750MHz UltraSparc-III processors, 48GB of shared memory

Logarithmic utility function (nonlinear objective + quadratic constraint)

Problem	1 proc		2 procs		4 procs		8 procs	
	iter	time(s)	time(s)	su	time(s)	su	time(s)	su
NLP4	25	448	214	2.09	110	4.07	72	6.22
NLP5	31	1287	618	2.08	306	4.20	179	7.19
NLP6	56	25578	13044	1.96	6650	3.85	3566	7.17
NLP7	60	24414	12480	1.96	6275	3.89	3338	7.31

Skewness formulation (nonlinear objective + quadratic constraint)

Problem	1 proc		2 procs		4 procs		8 procs	
	iter	time(s)	time(s)	su	time(s)	su	time(s)	su
NLP4	50	820	390	2.10	208	3.94	130	6.31
NLP5	43	1466	715	2.05	396	3.70	207	7.08
NLP6	65	28678	14393	1.99	7144	4.01	3845	7.46
NLP7	62	23664	11963	1.98	6131	3.86	3097	7.64

24 750MHz UltraSparc-III processors, 48GB of shared memory

Statistics of quadratic programs used for comparison with CPLEX.

Problem	Stages	Blocks	Assets	Total Nodes	constraints	variables
QP-NLP1	4	24	12	14425	201.351	546.950
QP-NLP2	3	60	20	3661	80.483	226.862
QP-NLP3	3	80	20	6481	142.503	401.662
QP-NLP4	3	70	40	4971	208.713	606.322

Comparison with CPLEX 7.0:

Problem	CPLEX 7.0			OOPS		
	time (s)	iter	mem (Mb)	time (s)	iter	mem (Mb)
QP-NLP1	615	28	369	406	14	325
QP-NLP2	955	16	340	159	15	136
QP-NLP3	2616	16	736	283	13	238
QP-NLP4	12236	18	1731	531	17	365

400MHz UltraSparc-II processor, 2GB of memory

Conclusions:

- Implementation of general purpose structure exploiting linear algebra module
- Used inside IPM to solve QPs and NLPs
- Efficient implementation: QPs/NLPs with up to 50/10 million variables solved
- Compares favourably with CPLEX 7.0 (on selected problems)
- Parallelizes well: near perfect speed-up on 2-8 processors

Future Work:

- More efficient NLP treatment: warmstart, more robust, etc.
- Implement other structures/other strategies (Iterative solver etc)
- New applications for LA module: Quasi-Newton methods ?
- Link to modelling language (SMPS is supported, anything else?)

Object-Oriented Parallel Solver (OOPS):

`http://www.maths.ed.ac.uk/~gondzio/parallel/solver.html`

References:

- J. Gondzio and A. Grothey, *Reoptimization with the primal-dual interior point method*, **SIAM Journal on Optimization** 13 (2003) pp 842–864.
- J. Gondzio and A. Grothey, *Parallel interior point solver for structured quadratic programs: application to financial planning problems*, Tech. Rep. MS-03-001, School of Maths, University of Edinburgh, April 2003.
- J. Gondzio and A. Grothey, *Solving nonlinear portfolio optimization problems with the primal-dual interior point method*, Tech. Rep. MS-04-001, School of Maths, University of Edinburgh, May 2004.

Papers available from:
`http://www.maths.ed.ac.uk/ERGO/group/`