Parallélisation des algorithmes de points intérieurs pour la programmation linéaire

Journées en l'honneur de Pierre Huard

Nelson Maculan[1] Luiz Carlos da Costa Jr[1]
Fernanda Sousa Thomé[1] Geraldo Veiga[2]

[1]COPPE – Université Fédérale de Rio de Janeiro, Brésil

[2]$\mathcal{R}^n$ Ciência e Tecnologia

24 et 25 novembre 2008

Outline

Brief History of Computational LP - Pioneers

- Pas loin d'ici: Fourier (1826) studies the properties of system of linear inequalities
- De la Vallée-Poussin (1911) develops an iterative procedure for linear minimax estimation which can be adjusted to solve linear optimization problems (Farebrother, 2006)
- Kantorovich (1939) proposes rudimentary algorithm for linear programming applied to production planning
- These contributions only come to attention after independent development of linear programming theory and the Simplex Method

Brief History of Computational LP - Simplex Method I

- Early works by Leontief, von Neumann and Koopsman directly influenced the theoretical development of linear programming (Dantzig, 2002)
- Exponential behavior in theory - Almost linear in practice
- Designed "to be computable", developed side-by-side with digital computers (Dantzig, 2002)
- Orchard-Hays (1954) produces first successful LP software
- Sparse matrix representation and product-form of the inverse
- Largest problem solved: 26 x 71 solved in 8 hours (Bixby, 2002)
- Large scale methods: Dantzig-Wolfe and Benders decomposition

Brief History of Computational LP - Simplex Method II

- Sparse LU representation of the basis with Bartel-Golub/Forrest-Tomlin/Fletcher-Matthews updates.
- More recent linear algebra improvements such as Markowitz threshold and sparse partial pivoting (Bixby, 2002)
- Modern implementations: CPLEX, Xpress and open source Glpk
- Parallel implementations of the simplex method usually exploits special structures
- A general approach hindered by the changing sparse pattern of the basic matrix

- The Ellipsoid Method (Khachiyan, 1979)
- Revolutionary for complexity theory without computational impact

Brief History of Computational LP - Interior Point I

- Karmarkar's algorithm (Karmarkar, 1984)
  - Projective algorithm with a potential function sets a lower complexity for linear programming: $\mathcal{O}(n^{3.5}L)$
  - Claims of great performance gains for a dual-affine scaling variant (Adler et al., 1989a)
  - Similar algorithm had gone unnoticed by LP researchers (Dikin, 1967)
- Primal-Dual/Path Following methods
  - New wave of interest in linear programming reintroduces path-following methods developed in the nonlinear context: Logarithm Barrier Function (Fiacco and McCormick, 1968) and Method of Centers (Huard, 1967)
  - Central trajectory methods with lower complexity $\mathcal{O}(n^3L)$
  - Primal/Dual infeasible methods become standard for implementation, included in leading LP software.

Affine-Dual Algorithm

- c, x n-vectors; A $m \times n$ matrix; b, y m-vectors

$$\max \{b^\top x \mid A^\top y \leq c\}$$

- Add slack variables

$$\max \{b^\top x \mid A^\top y + v = c,\ v \geq 0\}$$

- Scaling transformation

$$\hat{v} = D_v^{-1} v \quad \text{where} \quad D_v = \text{diag}(v_1^k, \ldots, v_m^k)$$

- Projected gradient as search direction

$$h_y = (A D_v^{-2} A^\top)^{-1} b \quad \text{and} \quad h_v = -A^\top h_y$$

Affine-Dual Algorithm II

1. **procedure** dualAffine (A, b, c, y^0, *stopping criterion*, γ)
2. $k := 0$;
3. **do** *stopping criterion* not satisfied $\rightarrow$
4. $\quad v^k := c - A^\top y^k$;
5. $\quad D_v := \mathrm{diag}(v_1^k, \ldots, v_m^k)$;
6. $\quad h_y := (AD_v^{-2}A^\top)^{-1}b$;
7. $\quad h_v := -A^\top h_y$;
8. $\quad$ **if** $h_v \geq 0 \rightarrow$ **return fi**;
9. $\quad \alpha := \gamma \times \min\{-v_i^k/(h_v)_i \mid (h_v)_i < 0, i = 1, \ldots, m\}$;
10. $\quad y^{k+1} := y^k + \alpha h_y$;
11. $\quad k := k + 1$;
12. **od**
13. **end** dualAffine

Primal-Dual Algorithm with infeasibilities I

- Formulation:
  - Upper bounds for a subset of variables
  - c, x, s, z are n-vectors
  - u_b, x_b, s_b, w_b n_b-vectors – x_n, s_n n_n-vectors
  - A $m \times n$ matrix – b, y m-vectors
- Add slack variables

$$\min \{c^\top x \mid Ax = b, x_b + s_b = u_b,\ x \geq 0, s_b \geq 0\}$$

$$\max \{b^\top y - u_b^\top w_b \mid A_b^\top y - w_b + z_b = c_b,$$
$$A_n^\top y + z_n = c_n,\ w_b \geq 0,\ z \geq 0\}$$

Primal-Dual Algorithm with infeasibilities II

- $X = \text{diag}(x)$, $S = \text{diag}(s)$, $W = \text{diag}(w)$, $Z = \text{diag}(z)$
- μ Central trajectory parameter
- Karush-Kuhn-Tucker conditions:

$$
\begin{aligned}
Ax &= b \\
x_b + s_b &= u_b \\
A_b^\top y - w_b + z_b &= c_b \\
A_n^\top y + z_n &= c_n \\
XZe &= \mu e \\
S_b W_b e &= \mu e \\
x, s_b, w_b, z &> 0
\end{aligned}
$$

Primal-Dual Algorithm with infeasibilities III

- System of equations with primal and dual infeasibilities

$$
\begin{aligned}
A\Delta x^k &= -(Ax^k - b) = r_p^k \\
\Delta x_b^k + \Delta s_b^k &= -(x_b^k + s_b^k - u_b) = r_u^k \\
A_b^\top \Delta y^k - \Delta w_b^k + \Delta z_b^k &= -(A_b^\top y^k + z_b^k - w_b^k - c_b) = (r_d^k)_b = 0 \\
A_n^\top \Delta y^k + \Delta z_n^k &= -(A_n^\top y^k + z_n^k - c_n) = (r_d^k)_n \\
Z^k \Delta x^k + X^k \Delta z^k &= -(X^k Z^k e - \mu_k e) = r_{xz}^k \\
W_b^k \Delta s_b^k + S_b^k \Delta w_b^k &= -(W_b^k S_b^k e - \mu_k e) = (r_{sw}^k)_b
\end{aligned}
$$

Primal-Dual Algorithm with infeasibilities IV

- Normal Equations

$$A\Theta^k A^\top \Delta y^k = \bar{b}$$

where

$$\Theta^k = \begin{bmatrix} (Z_b^k (X_b^k)^{-1} + W_b^k (S_b^k)^{-1})^{-1} & 0 \\ 0 & (Z_n^k)^{-1} X_n^k \end{bmatrix}$$

$$\begin{aligned} \bar{b} = r_p^k &+ A_b \Theta_b^k ((r_d^k)_b + (S_b^k)^{-1}(r_{sw}^k - W_b r_u^k) - (X_b^k)^{-1} r_{xz}^k) \\ &+ A_n \Theta_n^k ((r_d^k)_n - (X_n^k)^{-1} r_{xz}^k) \end{aligned}$$

Primal-Dual Algorithm with infeasibilities V

- Other search direction computed without substantial computational effort

$$
\begin{aligned}
\Delta x_b^k =& \Theta_b^k A_b^\top \Delta y^k - \Theta_b^k((r_d^k)_b + \\
& (S_b^k)^{-1}(r_{sw}^k - W_b r_u^k) - (X_b^k)^{-1}(r_{xz}^k)_b) \\
\Delta x_n^k =& \Theta_n^k A_n^\top \Delta y^k - \Theta_n^k((r_d^k)_n - (X_n^k)^{-1}(r_{xz}^k)_n) \\
\Delta s_b^k =& r_u^k - \Delta x_b^k \\
\Delta z^k =& (X^k)^{-1}(r_{xz} - Z^k \Delta x^k) \\
\Delta w_b^k =& A_b^\top \Delta y^k + \Delta z_b^k
\end{aligned}
$$

Parallelization opportunities in Interior Point Direct Factorization I

- Main computational step common to all variants is the solutions of a system of normal equations

$$A\Theta^k A^\top \Delta y^k = \bar{b}$$

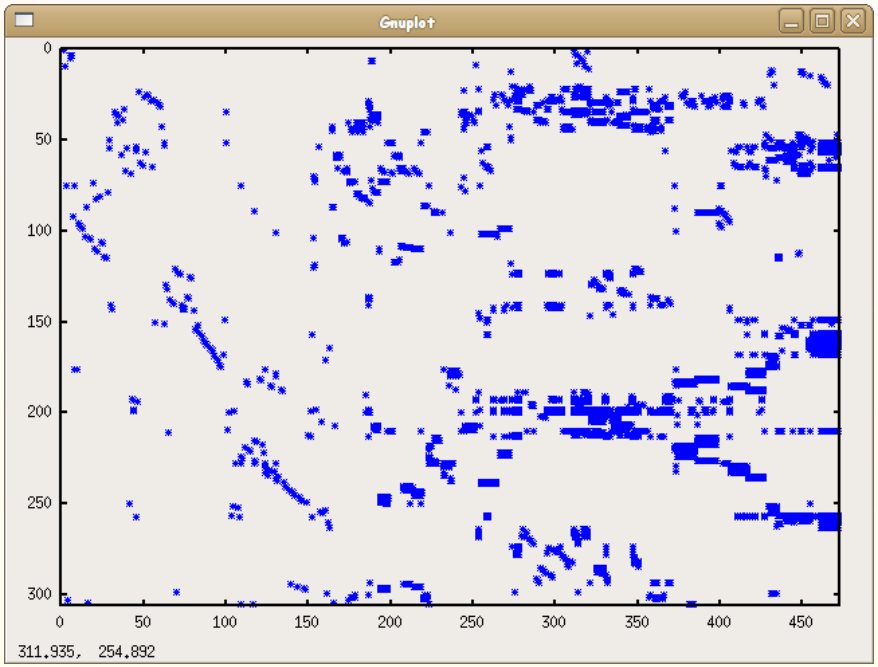
- Examining an implementation in Matlab/Octave, potentially computationally expensive steps:
- Computing system matrix

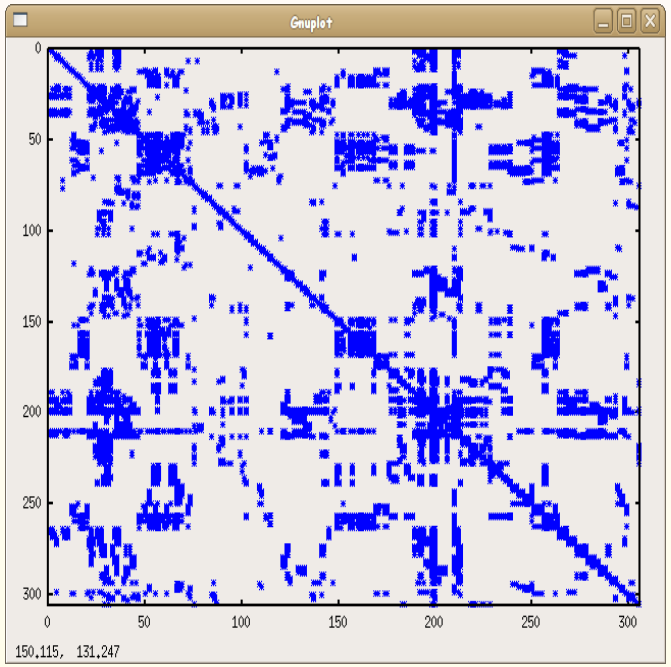
```
B = A*sparse(diag(d))*A';
```

- Custom parallel sparse linear algebra

Parallelization opportunities in Interior Point Direct Factorization II

- Example: BandM from the Netlib collection



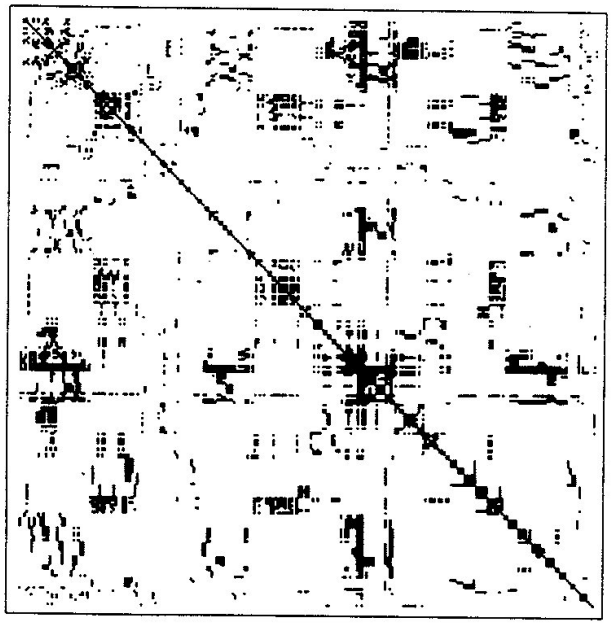


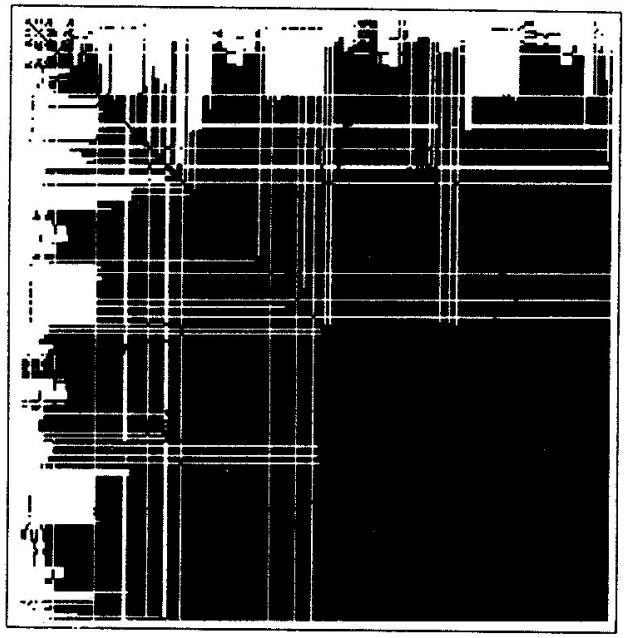
Parallelization opportunities in Interior Point Direct Factorization III

- Order for sparsity

```
ordering = symamd(B);
```

- Reordering for sparsity: Matrix $AA^\top$ and Cholesky factors without ordering (Adler et al., 1989b)





Parallelization opportunities in Interior Point Direct Factorization IV

- Matrix $AA^\top$ and Cholesky factors after minimum *degree ordering*

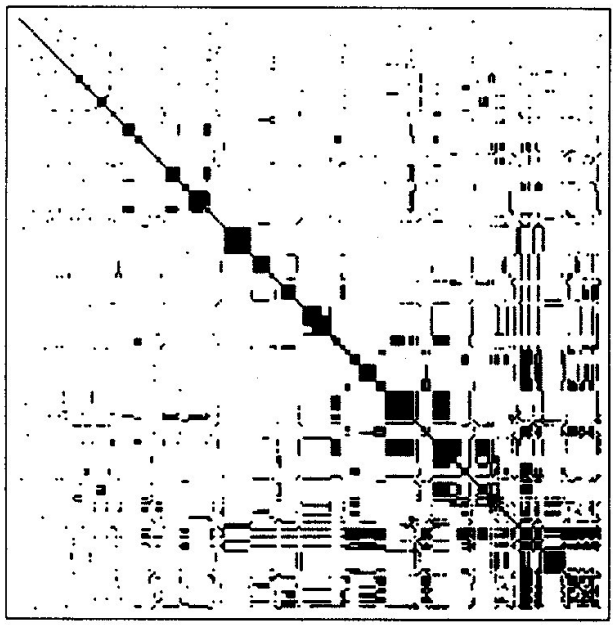


Figure 6. Nonzero pattern of AA^T after ordering (*minimum degree* ordering heuristic).

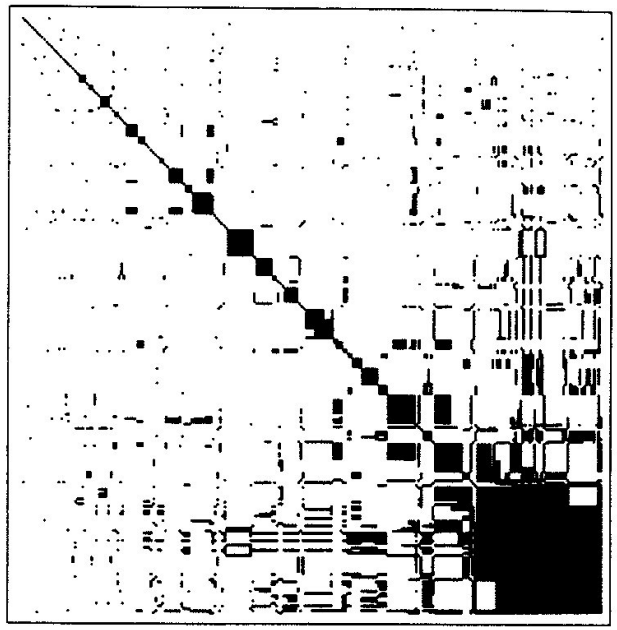


Figure 7. Nonzero pattern of LU factors after ordering (*minimum degree* ordering heuristic).

Parallelization opportunities in Interior Point Direct Factorization V

- Reordering rows of A to avoid *fill-in*
- Optimal ordering is *NP-Complete* (Yannakakis, 1981)
- Linear solvers compute the ordering during the *Analyse* step, based solely on the matrix sparsity pattern
- Performed only once in interior point algorithms, sparsity pattern are identical for all iterations
- Parallel/Distributed MPI based implementations available: ParMETIS

Parallelization opportunities in Interior Point Direct Factorization VI

- Direct Cholesky factorization

```
R = chol(B(ordering,ordering));
```

- Repeated at every iteration, consumes most of the computational effort
- For larger problems: Main parallelization target
- Chart displaying the portion of the algorithm running time for Netlib problems, suggesting a increase with size
- Available Parallel/Distributed implementations: MUMPS (Amestoy et al., 2000) for distributed memory architectures and PARDISO for shared memory

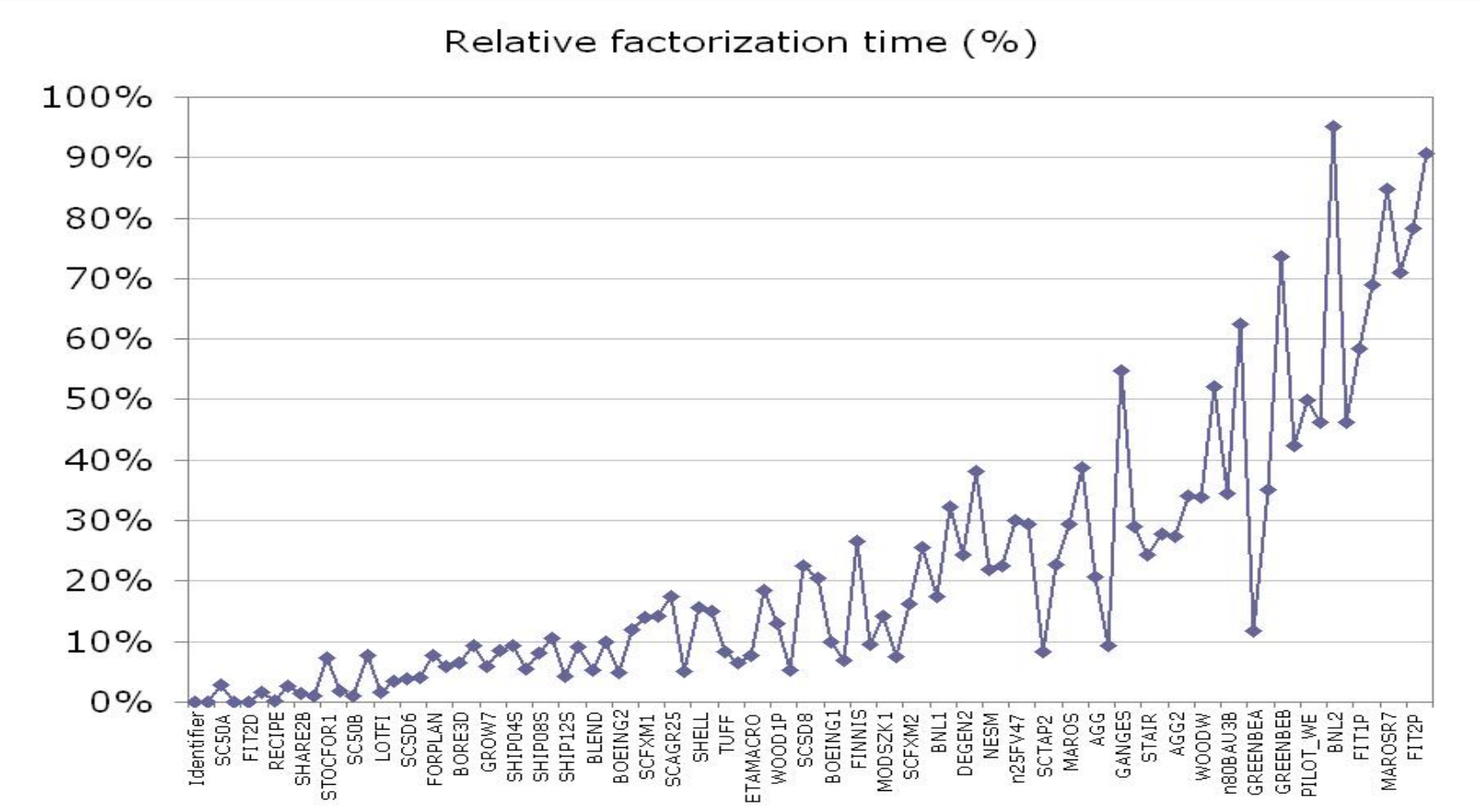
Parallelization opportunities in Interior Point Direct Factorization VII

- Triangular Solution for rhs

```
dy(ordering) = R\(R'\bbar(ordering));
```

- General sparse linear algebra parallelization
- In distributed implementations, parallelization implied by *Factorization* step

Parallelization opportunities in Interior Point Direct Factorization VIII



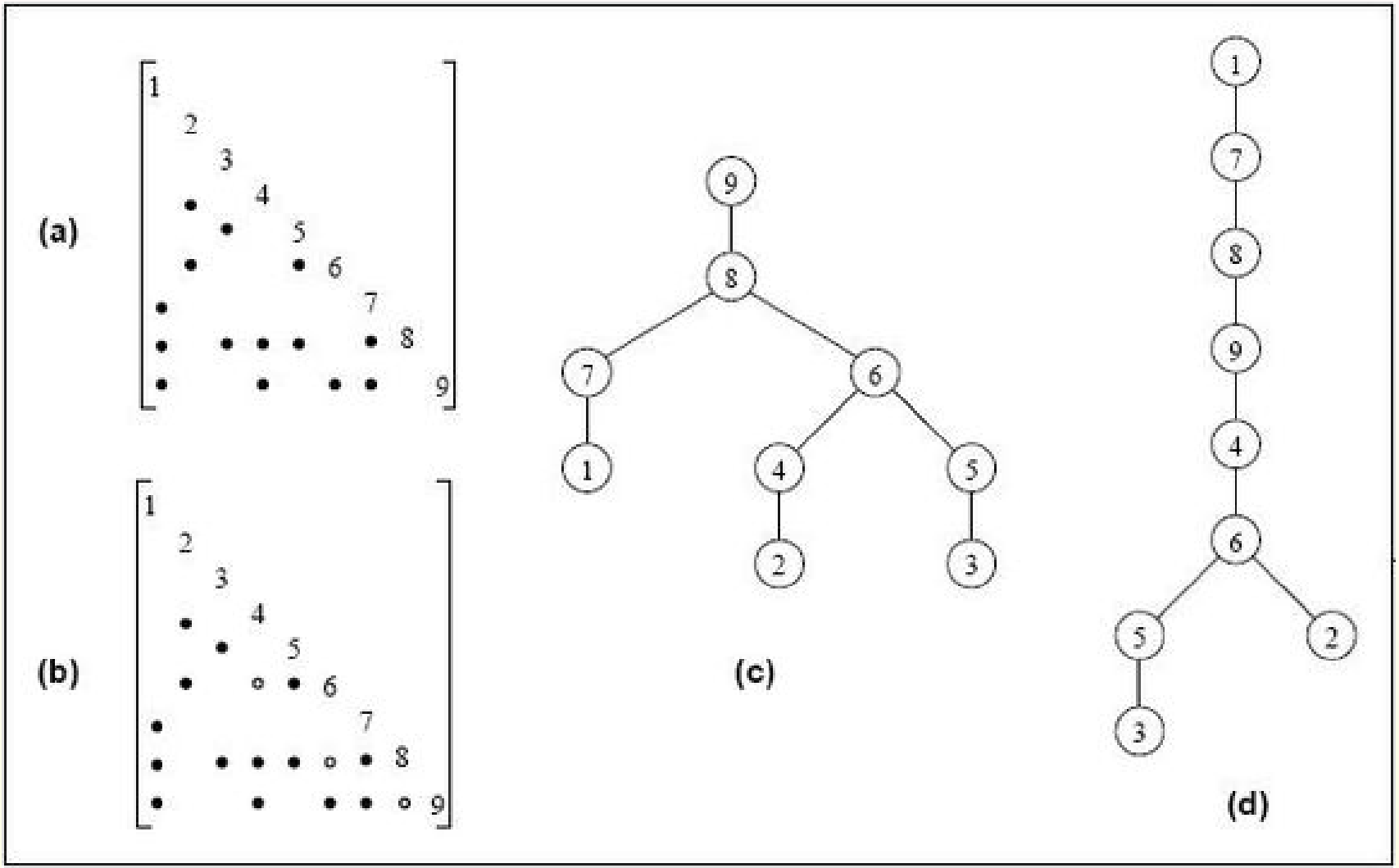
Parallelization strategies

- MPI: Message Passing/Distributed Memory
  - Standard for high-performance computing
  - Processors operate with private memory spaces, sharing results of only through point-to-point or collective communication
  - Goals are high performance, scalability and portability
  - Bindings for Fortran and C/C++
  - Target architectures are both high performance computer clusters tightly linked with fast switched interconnects and grids of loosely-coupled systems
- Shared memory multiprocessing
  - Multiple computing threads operate in shared memory space
  - Programming standards: OpenMP and Pthreads (Posix threads)
  - Suited for multi-core processor architectures
- Hybrid model of parallel programming use multi-core MPI nodes executing shared memory threads

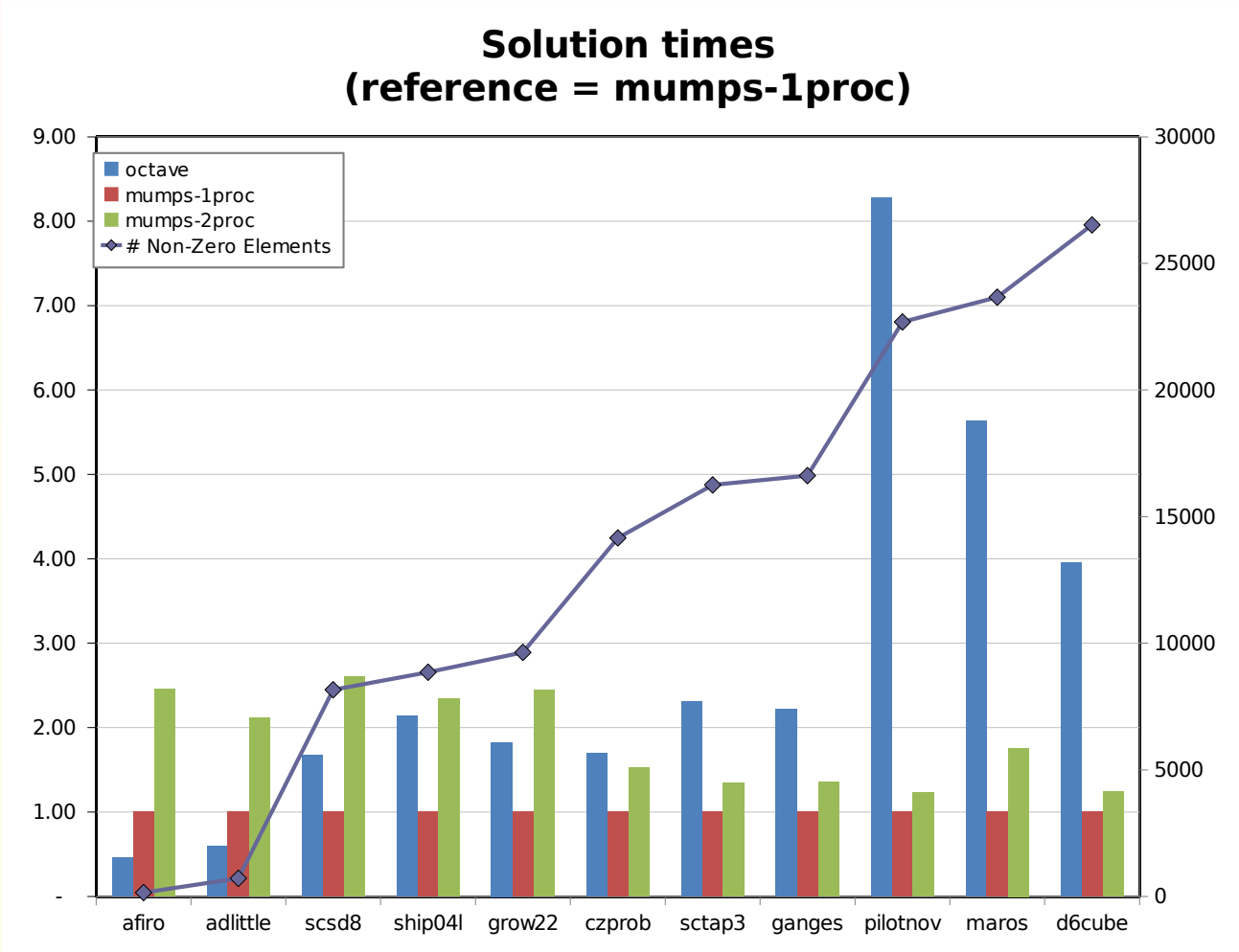
Experimenting with MUMPS

- Multifrontal Massively Parallel Solver MUMPS (Amestoy et al., 2000) for distributed memory architectures
- Multifrontal methods first build an assembly tree
- At each node, a dense submatrix (frontal matrix) is assembled using data from the original matrix and from the children of the node
- Main source of parallelism consists in simultaneously assigning same level frontal matrices onto separate processors
- MUMPS uses standard linear algebra libraries BLAS, BLACS, ScaLAPACK
- BLAS functions can use shared memory parallelism, depending on implementation
- Experiments with Netlib collection unsuccessful due to small size, but suggest better performance as problems grow

Multifrontal assembly trees for two orderings



Experiments with Netlib problems



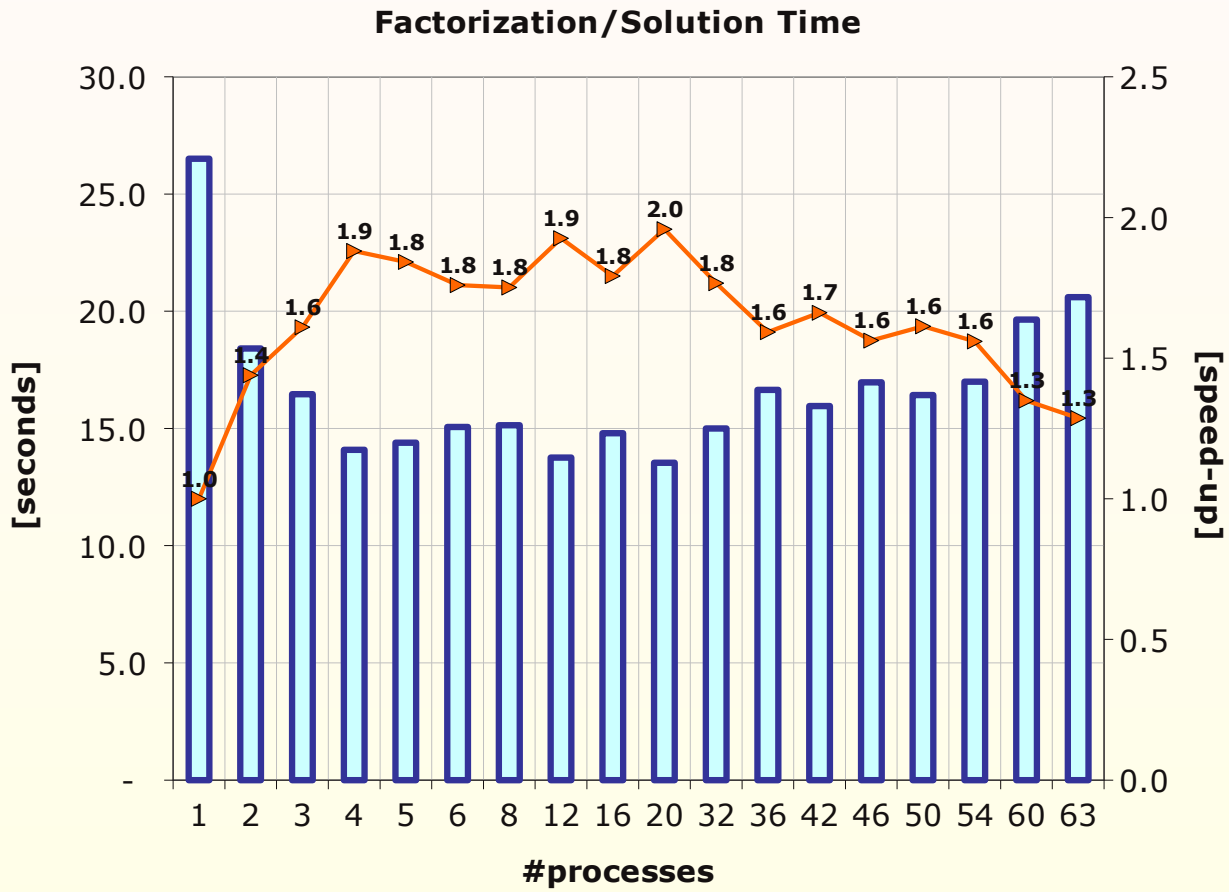
Power system expansion planning model

- Linear relaxation of mixed integer planning model for the expansion of a combined hydro and thermal power system
- Formulated with Optgen© modeling tool, developed by PSR
- Problem instance generated with Brazilian system of 280 hydro and 120 thermal plants
- LP size: 840285 columns, 598066 rows and 2462627 nonzeros entries
- Interior Point linear system: 360455 rows and 27390204 nonzeros

Case Study Experiment

- Experiment solves one typical system from an interior point iteration
- SGI Altix ICE 8200 with 64 quad-Core Intel Xeon CPU and 512 Gbytes of distributed RAM, using a Infiniband interconnect
- Software infrastructure: MUMPS 4.8.3 with BLAS, BLACS, ScaLAPACK provided by Intel MKL 10.1
- MUMPS is successful in low-scale parallelization
- Times for the Analyze stage comparable
- Total computation is dominated by matrix-matrix multiplication
- Shared memory parallelism using OpenMP in the BLAS and Lapack routines has little effect in this architecture

MUMPS Speedup



Conclusion and future work

- Large-scale problems using implementations with direct factorization can profit from parallelization, but less than expected
- Parallelization still an art form: No assurance of performance, too dependent on the infrastructure and algorithms
- MUMPS and other MPI-based tools are designed for high performance clusters
- Multi-core workstations are a better suited for shared memory parallelization
- Other sources of parallelism must be addressed
- Experiments with iterative methods for solving interior point linear systems
- Full implementation of the primal-dual algorithm for HPC environment

References I

I. Adler, N. Karmarkar, M. Resende, and G. Veiga. An implementation of Karmarkar's algorithm for linear programming. *Mathematical Programming*, 44:297–335, 1989a. doi: 10.1007/BF01587095. URL `http://citeseer.ist.psu.edu/173633.html`.

I. Adler, N. Karmarkar, M. G. C. Resende, and G. Veiga. Data structures and programming techniques for the implementation of Karmarkar's algorithm for linear programming. *ORSA J. Computing*, 1(2):84–106, 1989b. doi: 10.1007/BF01587095. URL `http://citeseer.ist.psu.edu/adler89data.html`.

P. R. Amestoy, I. S. Duff, and J. Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184:501–520, 2000. doi: 10.1016/S0045-7825(99)00242-X.

Robert E. Bixby. Solving real-world linear programs: A decade and more of progress. *Oper. Res.*, 50(1):3–15, 2002. ISSN 0030-364X. doi: http://dx.doi.org/10.1287/opre.50.1.3.17780.

George B. Dantzig. Linear programming. *Oper. Res.*, 50(1):42–47, 2002. ISSN 0030-364X. doi: 10.1287/opre.50.1.42.17798.

C.J. de la Vallée-Poussin. Sur la méthode de l'approximation minimum. *Ann. Soc. Sci. Bruxelles*, 35:1–16, 1911.

I. I. Dikin. Iterative solution of problems of linear and quadratic programming. *Soviet Mathematics Doklady*, 8: 674–675, 1967.

Richard William Farebrother. A linear programming procedure based on de la vallée poussin's minimax estimation procedure. *Computational Statistics & Data Analysis*, 51(2):453–456, 2006. doi: 10.1016/j.csda.2005.10.005.

A.V. Fiacco and G.P. McCormick. *Nonlinear programming: Sequential unconstrained minimization technique*. John Wiley and Sons, New York, 1968.

References II

J. B. J. Fourier. Solution d'une question particulière du calcul des inégalités. *Nouveau Bulletin des Sciences par la Société Philomatique de Paris*, pages 99–100, 1826. reprinted: G. Olms, Hildesheim, 1970, pp. 317-319.

P. Huard. *Resolution of mathematical programming with nonlinear constraints by the method of centers*, pages 209–219. North-Holland, Amsterdam, 1967.

L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Sciences*, 6: 366–422, 1939. English translation of the Russian original published in 1939 by the Publication House of the Leningrad State University.

N. Karmarkar. New polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. doi: 10.1007/BF02579150.

L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:1979, 1979.

M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J Alg Disc Meth*, 2:77–79, 1981.

```
1  function [x,y,s,w,z,fp,fd] = pdBounds(A,b,c,u,ub)
2  % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
3  % Version: 2008-07-15
4  %
5  % Primal-dual interior-point method for problems with equality constraints,
6  % bounded and unbounded variables
7  %
8  % Primal: min {c'x | Ax = b, x(ub) +s(ub) = u(ub), x,s(ub)>=0}
9  % Dual: max{b'y - u'(ub)w(ub) | A'y - w(ub) + z = c, w >= 0, z >= 0}
10 %
11 % input: A - m x n sparse constraint matrix
12 %        b - m x 1 resource vector
13 %        c - n x 1 cost vector
14 %        u - n x 1 bounds vector
15 %        ub - n x 1 vector (maximum dimension) with indices of bounded variables
16 %
17 % output: x - n x 1 vector of primal solutions
18 %         y - m x 1 vector of dual solutions for equality constraints
19 %         s - n x 1 vector of primal slacks (defined for indices in ub only)
20 %         w - n x 1 vector of dual solutions for upper-bound constraints
21 %         z - n x 1 vector of dual slacks
22 %         fp - primal objective value
23 %         fd - dual objective value
24 %
25 % Internal parameters:
26 %         itmax - Maximum number of iterations in algorithm
27 %         tol - Convergence tolerance
28 %         tolzro - Tolerance to zero
```

```
29  %          x0fac - Factor for computing minimum entry in x0
30  %          max_diag - Maximum diagonal element allowed
31  %          min_eta - Minimum safety factor in step
32  %-------------------------------------------------------------------------
33
34  % Check input arguments
35  if nargin ~= 5
36    error('pdBounds: Missing arguments - 5 required');
37  end
38
39  if (~issparse(A))   error('pdBounds: Matrix A is not sparse');   end
40  [m,n] = size(A);
41  if (m <= 0 || n <= 0)
42    error('pdBounds: Matrix A must be nontrivial');
43  end
44  if (n ~= length(c))
45    error('pdBounds: size of vector c must match number of columns in A');
46  end
47  if (m ~= length(b))
48    error('pdBounds: size of vector b must match number of rows in A');
49  end
50  if (n ~= length(u))
51    error('pdBounds: size of vector u must match number of columns in A');
52  end
53
54  % Build Indices for bounded and unbounded variables
55  nub = setdiff([1:n], ub);
56  n_ub = size(ub,2); n_nub = n - n_ub;
57
```

```
58  % Precompute norms
59  nrm_b = norm(b,2); nrm_c = norm(c,2);   nrm_cb = norm(c(ub),2);
60  if(n_ub > 0)
61    nrm_u = norm(u(ub),2);
62  else
63    nrm_u = 0;
64  end
65
66  % Set internal parameters
67  itmax = 100;                    % Maximum number of iterations in algorithm
68  tol = 1.e-8;                    % Convergence tolerance
69  tolzro = 1.e-20;                % Tolerance to zero
70  x0fac = 10;                     % Factor computing minimum entry in x0
71  max_diag = 1.e+20;              % Maximum diagonal element allowed
72  min_eta = .9995;                % Minimum safety factor in step
73
74  % Start CPU clock
75  ts=cputime;
76
77  % Initialize arrays and variables
78  x = zeros(n,1); s = zeros(n,1); w = zeros(n,1); z = zeros(n,1);
79  rd = zeros(n,1); ru = zeros(n,1);rxz = zeros(n,1); rsw = zeros(n,1); dx = zeros(n,1);
80  ds = zeros(n,1); dw = zeros(n,1); dz = zeros(n,1); tn = zeros(n,1);
81  y = zeros(m,1); dy = zeros(m,1); rp = zeros(m,1); tm = zeros(m,1); mu = 0;
82
83  % Find minimum degree ordering for a sparse Cholesky factorization of ADA'
84  B = A*A'; ordering = symamd(B);
85
86  % Compute initial primal solution
```

```
87   R = chol(B(ordering,ordering));
88   if(n_ub > 0)
89     bbar = 2*b - A(:,ub)*u(ub);
90     tm(ordering) = R\(R'\bbar(ordering));
91     x(ub) = .5*(A(:,ub)'*tm + u(ub));
92     minx0 = max(norm(x(ub),inf)/x0fac,nrm_u/x0fac);
93     x(ub) = max(minx0,x(ub));
94     s(ub) = u(ub) - x(ub);
95     s(ub) = max(minx0,s(ub));
96     if(n_ub > 0)
97       x(nub) = .5*(A(:,nub)'*tm);
98       minx0 = max(norm(x(nub),inf)/x0fac,nrm_b/x0fac);
99       x(nub) = max(minx0,x(nub));
100    end
101  else
102    tm(ordering) = R\(R'\b(ordering));
103    x = A'*tm;
104    minx0 = max(norm(x,inf)/x0fac,nrm_b/x0fac);
105    x = max(minx0,x);
106  end
107
108  % Set initial dual interior solution
109  z = ones(n,1)./x; z = (max(1,nrm_c)/norm(z,2))*z;
110  if(n_ub > 0)
111    w(ub) = ones(n_ub,1)./s(ub); w(ub) = (max(1,nrm_cb)/norm(w(ub),2))*w(ub);
112  end
113  y = zeros(m,1);
114
115  % Main algorithm loop
```

```
116
117   for iter = 0:itmax
118
119     % Adjust dual slacks for bounded variables to force zero dual residual
120     if(n_ub > 0)
121       rd(ub) = c(ub) - A(:,ub)'*y - z(ub) + w(ub);
122     end
123     for i = ub
124       if(rd(i) > 0)
125         z(i) = z(i) + rd(i);
126       elseif(rd(i) < 0)
127         w(i) = w(i) - rd(i);
128       end
129       rd(i)=0;
130     end
131
132     % Compute current primal and dual objective values
133     fp = c'*x; fd = b'*y;
134     if(n_ub > 0)
135       fd = fd - u(ub)'*w(ub);
136     end
137
138     % Compute current primal, dual and complementarity residuals
139     rp = b - A*x; nrm_rp = norm(rp,2)/max(1,nrm_b);
140     rd = c - A'*y - z;
141     if(n_ub > 0)
142       rd(ub) = rd(ub) + w(ub);
143     end
144     nrm_rd = norm(rd,2)/max(1,nrm_c);
```

```
145
146     if(n_ub > 0)
147       ru(ub) = u(ub) - x(ub) - s(ub); nrm_ru = norm(ru(ub),2)/max(1,nrm_u);
148       rsw(ub) = s(ub).*w(ub);  nrm_rsw = norm(rsw(ub),2);
149     else
150       nrm_ru = 0; nrm_rsw = 0;
151     end
152     rxz = x.*z; nrm_rxz = norm(rxz,2);
153
154     % Compute trajectory parameter
155     if(n_ub > 0)
156       mu = full((sum(rxz) + sum(rsw))/(n + n_ub));
157     else
158       mu = full(sum(rxz)/n);
159     end
160
161     % Print iteration report
162     fprintf(1,'Iteration %3i: mu= %15.8e, fp= %13.8e, fd= %13.8e\n', iter, full(mu), fp, f
163     fprintf(1,'Residuals: rp= %15.8e, rd= %13.8e, ru= %13.8e\n', nrm_rp, nrm_rd, nrm_ru);
164     fprintf(1,'           rxz= %13.8e, rsw= %13.8e\n', nrm_rxz, nrm_rsw);
165
166     % Test for convergence
167     if(abs(fp-fd)/max(1,abs(fd)) < tol & nrm_rp < tol & nrm_rd < tol & nrm_ru < tol)
168       break;
169     end
170
171     % Choose target trajectory parameter and adjust residuals
172     sigma = min(0.1,100*mu);
173     rxz = sigma*mu*ones(n,1) - rxz;
```

```
174     if(n_ub > 0)
175       rsw(ub) = sigma*mu*ones(n_ub,1) - rsw(ub);
176     end
177
178     % Compute scaling matrix and coefficient matrix for normal equations
179     if(n_ub > 0)
180       d(ub) = min(max_diag, ones(n_ub,1)./(z(ub)./x(ub) + w(ub)./s(ub)));
181     end
182     if(n_nub > 0)
183       d(nub) = min(max_diag, x(nub)./z(nub));
184     end
185     B = A*sparse(diag(d))*A';
186
187     % Cholesky factorization of normal equations
188     R = chol(B(ordering,ordering));
189
190     % Compute rhs for normal equations
191     bbar = rp;
192     if(n_ub > 0)
193       tn(ub) = diag(d(ub))*(rd(ub) + (rsw(ub)-w(ub).*ru(ub))./s(ub) - rxz(ub)./x(ub));
194     end
195     if(n_nub > 0)
196       tn(nub) = diag(d(nub))*(rd(nub) - rxz(nub)./x(nub));
197     end
198     bbar = bbar + A*tn;
199
200     % Solve the normal equations system for dy and recover dx, ds, dw, dz
201     dy(ordering) = R\(R'\bbar(ordering));
202     dx = diag(d)*(A'*dy) - tn;
```

```
203     dz = (rxz - z.*dx)./x;
204     if(n_ub > 0)
205       ds(ub) = ru(ub) - dx(ub);
206       dw(ub) =  A(:,ub)'*dy + dz(ub);
207     end
208
209     % Compute maximum feasible step
210     alphax = ratioTest(x,dx,tolzro);
211     if(n_ub > 0)
212       alphas = ratioTest(s(ub),ds(ub),tolzro);
213       alphap = min(alphax,alphas);
214     else
215       alphap = alphax;
216     end
217     if (isinf(alphap))
218       error('Extreme ray found in primal problem');
219     end
220
221     alphaz = ratioTest(z,dz,tolzro);
222     if(n_ub > 0)
223       alphaw = ratioTest(w(ub),dw(ub),tolzro);
224       alphad = min(alphaz,alphaw);
225     else
226       alphad = alphaz;
227     end
228     if (isinf(alphad))
229       % Check for dual unbounded solution
230       error('Extreme ray found in dual problem');
231     end
```

```
232
233     % Update solution
234     eta = max(min_eta, 1 - mu);
235     x = x + eta * alphap * dx;
236     y = y + eta * alphad * dy;
237     z = z + eta * alphad * dz;
238     if(n_ub > 0)
239       s(ub) = s(ub) + eta * alphap * ds(ub);
240       w(ub) = w(ub) + eta * alphad * dw(ub);
241     end
242   end
243
244   fprintf('Total CPU time = %g\n', cputime-ts);
245   fprintf(1,'Problem size: %i X %i, %i bounded, %i nonzeros, rank %i\n', size(A), n_ub, nn
246   fprintf(1,'               %i nonzero elements in AAT\n', nnz(A*A'));
247   fprintf(1,'               %i nonzero elements in LU factors\n', (2*nnz(R)-m));
248   return;
249   end
250
251   function alpha = ratioTest(x,dx,tolzro)
252   % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
253   % Returns the maximum step for x and direction dx
254   %-------------------------------------------------------------------------------
255     alpha = inf('double');
256     for i = 1 : length(x)
257       if (dx(i) < -tolzro)
258         alpha = min(alpha,-x(i)/dx(i));
259       end
260     end
```

```
261 end
```