Cybermedia Center Seminar: 18 Oct.2017

Introduction of direct solvers for large linear system with hands-on tutorial

Atsushi Suzuki[1]

[1] Cybermedia Center, Osaka University
atsushi.suzuki@cas.cmc.osaka-u.ac.jp

sparse direct solver

A : sparse matrix, nonsingular,
nonzero entries are in symmetric position : structural symmetric

$A = \Pi_S^T \Pi_N^T W L D U W \Pi_N \Pi_S$,

Π_S, Π_N : permutation, W : scaling matrix(e.g. diagonal)

Three phases of direct solver:

- symbolic factorization to define Π_S to minimize fill-in : ordering
- numeric factorization with pivoting Π_N to reduce numerical round-off error
- forward-backward substitution

remarks

- unsymmetric nonzero pattern will be included in symmetric nonzero pattern by adding zero value at the symmetric position.
- sparse matrix is obtained from discretization of partial differential equation by FDM, FVM, or FEM.
- direct solver can factorize matrix with 1M DOFs by a multi-core processor.
- alternative solver is iterative method: Krylov subspace method (CG, GMRES), multigrid method, but direct solver is most robust

State of the art : software for sparse direct solver

Software	parallel env.	elimination strategy	data manag.	pivoting	kernel detection
UMFPACK	—-	multi-frontal	static	yes	no
SuperLU_MT	shared	super-nodal	dynamic	yes	no
Pardiso	shared/dist.	super-nodal	dynamic	yes + $\sqrt{\varepsilon}$-p.	no
SuperLU_DIST	distributed	super-nodal	static	no, $\sqrt{\varepsilon}$-p.	no
MUMPS	dist./shared	multi-frontal	dynamic	yes	yes
Dissection	shared	multi-frontal	static	yes	yes

T. A. Davis, I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices,
ACM Trans. Math. Software, 25 (1999), 1–20.

J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, J. W. H. Liu.
A supernodal approach to sparse partial pivoting,
SIAM J. Matrix Anal. Appl., 20 (1999), 720–755.

O. Schenk, K. Gärtner. Solving unsymmetric sparse systems of liner equations with PARDISO,
Future Generation of Computer Systems, 20 (2004), 475–487.

X. S. Li, J. W. Demmel. SuperLU_DIST : A scalable distributed-memory sparse direct solver for unsymmetric linear systems,
ACM Trans. Math. Software, 29 (2003), 110–140.

P. R. Amestoy, I. S. Duff, J.-Y. L'Execellent. Mutlifrontal parallel distributed symmetric and unsymmetric solvers,
Comput. Methods Appl. Mech. and Engrg, 184 (2000) 501–520.

A. Suzuki, F.-X. Roux, A dissection solver with kernel detection for symmetric finite element matrices on shared memory computers,
Int. J. Numer. Meth. in Engng, 100 (2014) 136–164.

Hands-on tutorial to deal with direct solver packages

- `Pardiso` in `Intel MKL`, ver.2017.0.2 commercial
- `MUMPS` ver.5.1.1 CeCILL-C
- `Dissection` ver.1.0.3 GPLv3/CeCILL-C

Superscalar/Vector multicore parallel computers in CMC.

- `VCC` : Intel Xeon E5-2670V2@2.5GHz: 10 cores$\times 2$, 64GB mem. 100GFlop/s
- `SX-ACE` : NEC SX-ACE@1.0GHz: 4 cores$\times 1$, 64GB mem. 256GFlop/s

sparse matrces on structural mechanics/fluid dynamics from `https://sparse.tamu.edu/` and original one

matrix	n	nnz	characteristic
`poisson3Db`	85,623	237,4949	unsymmetric
`CoupCons3D`	416,800	22,322,336	unsymmetirc
`F1`	343,791	13,590,452	symmetric
`stokes.skewsym`	565,003	16,715,009	unsymmetirc, singular

development of multi-core CPU

- clock speed of CPU never becomes faster in recent ten years
  stagnating at around 3GHz
  e.g., Intel CPUs

clock	#cores	name	energy	year
3.6 GHz	2	Pentium D 960	130W	2006
3.2 GHz	4	Xeon E7-8893V4	140W	2016
2.1 GHz	18	Xeon E5-2695V4	120W	2016
1.5 GHz	72	Xeon Phi 7290F	260W	2016

  one core in a note pc $\simeq$ one core in super computers

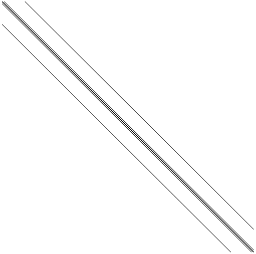
- multi-core processor with more than 64 cores
  – LDU-factorization with multi-frontal ordering
  $\Rightarrow$ no change of discretization of PDE up to 1M DOF
- advanced code for floating point operations
  – to utilize cache memory $\Leftarrow$ arithmetic intensive routine
  BLAS level 3, DGEMM
  – to minimize idle time of cores $\Leftarrow$ asynchronous exectuion,
  out-of-oder task operation

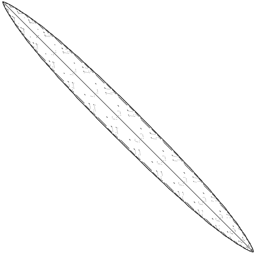
ordering of sparse matrix

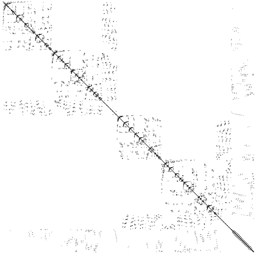
sparse matrix needs to be re-ordered

- to reduce fill-in
- to increase parallelization of factorization → multi-front
- to increase size of block structure → supernode

example:
7 stencil of Poisson equation, 11^3 nodes.





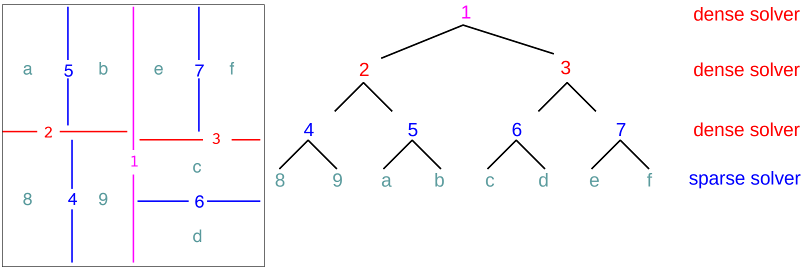


original matrix
band matrix

reverse Cuthill-McKee
sequential computation

nested-dissection
parallel computation

nested dissection by graph decomposition



A. George. Numerical experiments using dissection methods to solve n by n grid problems. SIAM J. Num. Anal. 14 (1977),161–179.

software package:

METIS : V. Kumar, G. Karypis, A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. 20 (1998) 359–392.

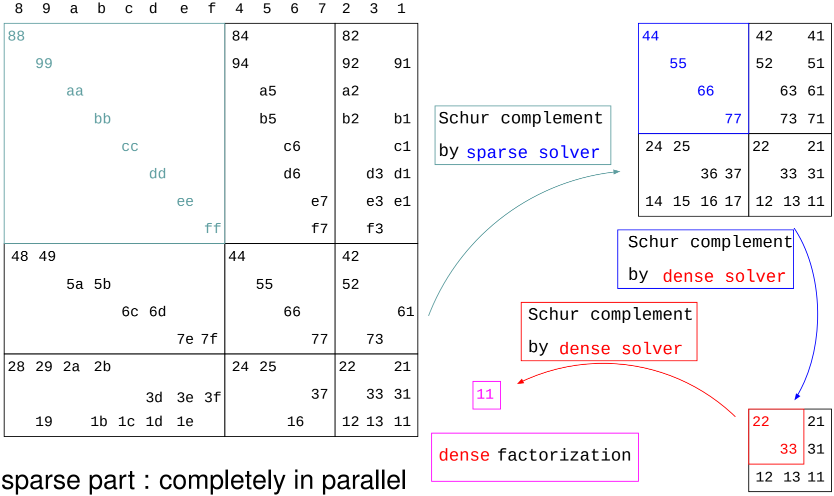
SCOTCH : F. Pellegrini J. Roman J, P. Amestoy, Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. Concurrency: Pract. Exper. 12 (2000) 69–84.

- each leaf can be computed in parallel $\Leftarrow$ multi-front
- load unbalance? because of different size of subdomains
- parallel computation of higher levels?
  # cores > # subdomains

recursive generation of Schur complement

$$\begin{bmatrix} A_{11} & A_{21} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & S_{22} \end{bmatrix} \begin{bmatrix} I_1 & A_{11}^{-1} A_{12} \\ 0 & I_2 \end{bmatrix}$$

$S_{22} = A_{22} - A_{21} A_{11}^{-1} A_{12} = A_{22} - (A_{21} U_{11}^{-1}) D_{11}^{-1} L_{11}^{-1} A_{12}$: recursively computed

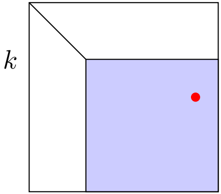


sparse part : completely in parallel

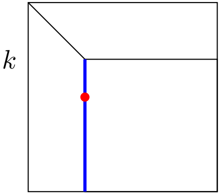
dense part : better use of BLAS 3; dgemm, dtrsm

pivoting strategy

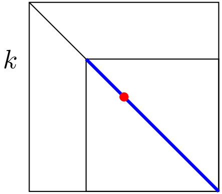
full pivoting : $A = \Pi_L^T L U \Pi_R$
find $\max_{k<i,j\le n} |A(i,j)|$



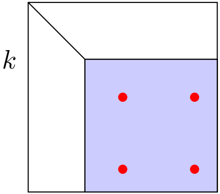
partial pivoting : $A = \Pi L U$
find $\max_{k<i\le n} |A(i,k)|$



symmetric pivoting : $A = \Pi^T L D U \Pi$
find $\max_{k<i\le n} |A(k,k)|$



2×2 pivoting : $A = \Pi^T \, L \, \tilde{D} \, U \Pi$
find $\max_{k<i,j\le n} \det \begin{vmatrix} A(i,i) & A(i,j) \\ A(j,i) & A(j,j) \end{vmatrix}$



sym. pivoting is mathematically not always possible

LDU factorization with rank-1 update

$$\begin{bmatrix} a_{11} & \beta_1^T \\ \alpha_1 & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \alpha_1 a_{11}^{-1} & S_{22} \end{bmatrix} \begin{bmatrix} a_{11} & \beta_1^T \\ 0 & I_2 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 \\ \alpha_1 a_{11}^{-1} & S_{22} \end{bmatrix} \begin{bmatrix} a_{11} & 0 \\ 0 & I_2 \end{bmatrix} \begin{bmatrix} 1 & a_{11}^{-1}\beta_1^T \\ 0 & I_2 \end{bmatrix}$$

Schur complement $S_{22} = A_{22} - \alpha_1 a_{11}^{-1} \beta_1^T$ rank-1 update by `dger`

LDU-factorization algorithm with symmetric pivoting

do $k = 1, \cdots, N$

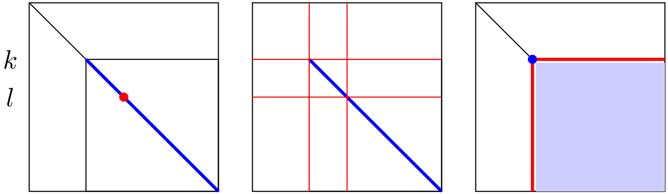
find max$|A(l,l)|$ with $k < l \leq n$,

exchange rows and columns : $A(k,*) \leftrightarrow A(l,*)$, $A(*,k) \leftrightarrow A(*,l)$.

`dscal` $A(k,j)$/=$A(k,k)$ $\quad k < j \leq N$,

`dscal` $A(i,k)$/=$A(k,k)$ $\quad k < i \leq N$,

`dger` $A(i,j)$-=$A(i,k)A(k,k)^{-1}A(k,j)$ $\quad k < i,j \leq N$.



LDU factorization of positive matrix : 1/2

$$A \text{ : coercive} \Leftrightarrow \quad (A\,x, x) > 0 \;\forall x \neq 0$$

$$\Leftrightarrow \quad \frac{A + A^T}{2} \text{ : sym. positive definite } \Rightarrow \exists A^{-1}$$

$$x^T A x = (Ax, x) = \frac{1}{2}\left\{(A\,x, x) + (A\,x, x)\right\} = \frac{1}{2}\left\{(A\,x, x) + (x, A^T\,x)\right\}$$

$$= \left(\frac{A + A^T}{2} x, x\right)$$

$$A = \begin{bmatrix} A_{11} & \beta_1 \\ \alpha_1^T & \alpha_{22} \end{bmatrix} \text{ : coercive} \Rightarrow \exists\; A_{11}^{-1},\; s_{22} \neq 0$$

$$\begin{bmatrix} A_{11} & \beta_1 \\ \alpha_1^T & \alpha_{22} \end{bmatrix} = \begin{bmatrix} I_1 & 0 \\ \alpha_1^T A_{11}^{-1} & 1 \end{bmatrix} \begin{bmatrix} A_{11} & \beta_1 \\ 0 & s_{22} \end{bmatrix}$$

$$s_{22} = \alpha_{22} - \alpha_1^T A_{11}^{-1}\;\beta_1 \in \mathbb{R}$$

proof

$$(A_{11}\,x_1, x_1) = \left(A \begin{bmatrix} x_1 \\ 0 \end{bmatrix}, \begin{bmatrix} x_1 \\ 0 \end{bmatrix}\right) > 0 \Rightarrow \exists A_{11}^{-1}$$

$$\begin{bmatrix} A_{11} & \beta_1 \\ \alpha_1^T & \alpha_{22} \end{bmatrix} \begin{bmatrix} A_{11}^{-1}\beta_1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ s_{22} \end{bmatrix}, \quad s_{22} = 0 \;\Rightarrow \text{dimker} A \geq 1 \;\Rightarrow\!\Leftarrow$$

LDU factorization of positive matrix : 2/2

$$\begin{aligned} A \text{ : coercive} \Leftrightarrow &\quad (A\,x, x) > 0 \;\forall x \neq 0 \\ \Leftrightarrow &\quad \frac{A + A^T}{2} \text{ : sym. positive definite} \end{aligned}$$

By recursively applying previous argument,

$A = \begin{bmatrix} A_{11} & \beta_1 \\ \alpha_1^T & \alpha_{22} \end{bmatrix}$: coercive $\Rightarrow$

$$\begin{aligned} \begin{bmatrix} A_{11} & \beta_1 \\ \alpha_1^T & \alpha_{22} \end{bmatrix} &= \begin{bmatrix} I_1 & 0 \\ \alpha_1^T A_{11}^{-1} & 1 \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & s_{22} \end{bmatrix} \begin{bmatrix} I_{11} & A_{11}^{-1}\beta_1 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} L_{11} & 0 \\ \alpha_1^T U_{11}^{-1} D_{11}^{-1} & 1 \end{bmatrix} \begin{bmatrix} D_{11} & 0 \\ 0 & s_{22} \end{bmatrix} \begin{bmatrix} U_{11} & D_{11}^{-1} L_{11}^{-1} \beta_1 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

- By mathematically exact computation, coercive matrix A has LDU-factorization with any permutation.
- By floating point computation, numerical pivoting is necessary to get stable factorization with good accuracy.

2×2 pivot for indefinite matrix

symmetric matrix

$$\begin{bmatrix} \frac{1}{4} & \frac{5}{4} & \frac{1}{2} \\ \frac{5}{4} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ 5 & 1 & \\ 2 & \frac{1}{3} & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} & & \\ & -6 & \\ & & \frac{2}{3} \end{bmatrix} \begin{bmatrix} 1 & 5 & 2 \\ & 1 & \frac{1}{3} \\ & & 1 \end{bmatrix}$$

by symmetric permutation

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{4} & \frac{5}{4} \\ \frac{1}{2} & \frac{5}{4} & \frac{1}{4} \end{bmatrix} = \begin{bmatrix} 1 & & \\ \frac{1}{2} & 1 & \\ \frac{1}{2} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 0 & 1 \\ & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ & 1 & 0 \\ & & 1 \end{bmatrix}$$

pivot strategy to take the largest entry may fail for indefinite matrix.
a remedy is to use 2×2 pivot.
an algorithm to mix 1×1 and 2×2 pivots for sym. indefinite matrix:

J. R. Bunch, L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems, *Math. Comput*, 31 (1977) 163–179.

$\sqrt{\varepsilon}$-perturbation

a regularization technique

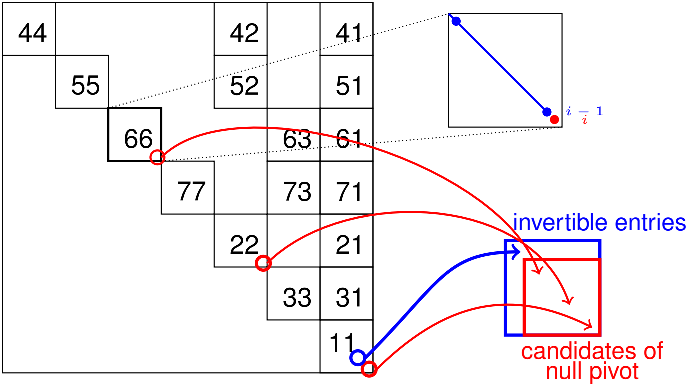
$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & \begin{matrix} 0 & \alpha \\ \beta & 0 \end{matrix} \end{bmatrix} \rightarrow \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & \begin{matrix} \sqrt{\varepsilon} & \alpha \\ \beta & 0 \end{matrix} \end{bmatrix}$$

- iterative refinement to improve accuracy of a solution
- user can/have to specify perturbation parameter for unsymmetric matrix (default = 10^{-13} for `Pardiso`)

symmetric pivoting with postponing for block strategy

- nested-dissection decomposition may produce singular sub-matrix for indefinite matrix

τ : given threshold for postponing, 10^{-2} for MUMPS, Dissection
$|A(i,i)|/|A(i-1,i-1)| < \tau \;\Rightarrow \{A(k,j)\}_{i\leq k,j}$ are postponed



Schur complement matrix from postponed pivots is computed
kernel detection algorithm : QR-factorization by MUMPS

kernel detection (rank deficient problem)

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & S_{22} \end{bmatrix} \begin{bmatrix} I_1 & A_{11}^{-1}A_{12} \\ 0 & I_2 \end{bmatrix} \quad S_{22} = 0 \Rightarrow \mathrm{Ker} A = \begin{bmatrix} A_{11}^{-1}A_{12} \\ -I_2 \end{bmatrix}$$

symmetric semi-positive definite, $m + k = 4 + 6 = 10$

by Householder-QR factorization:

4.60e-02	-1.20e-02	2.91e-03	1.16e-02	2.24e-02	-9.33e-05	-3.60e-02	8.22e-03	-7.77e-03	-2.90e-02
0.0	3.84e-02	4.84e-03	-2.21e-02	1.87e-02	1.30e-03	-9.14e-03	-2.74e-02	1.48e-02	-1.91e-02
0.0	0.0	2.96e-02	1.68e-03	-2.55e-02	1.11e-04	1.28e-02	-1.04e-04	-1.12e-03	-1.20e-02
0.0	0.0	0.0	1.28e-02	-1.66e-03	8.48e-04	-4.29e-05	-7.90e-04	-8.56e-03	2.51e-03
0.0	0.0	0.0	0.0	1.23e-11	-5.49e-13	-8.30e-12	1.67e-13	2.10e-14	-7.08e-12
0.0	0.0	0.0	0.0	0.0	6.70e-13	-1.02e-13	-5.33e-13	-1.62e-13	1.18e-13
0.0	0.0	0.0	0.0	0.0	0.0	3.33e-13	-2.48e-14	-6.61e-14	-3.18e-13
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.22e-13	4.46e-15	-4.34e-14
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.05e-14	8.16e-15
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.09e-14

how to set threshold to distinguish between
non-zero (1.28e-02) and zero (1.23e-11) values ?

Pardiso	no capability of kernel detection.
MUMPS	user has to choose this value.
Dissection	an algorithm by measuring dimension of residual of matrix with a projection onto the image space.

kernel detection algorithm based on LDU (Dissection)

$A : N \times N$ unsymmetric, dim Ker$A = k \geq 1$, dim Im$A \geq m$.
two parameters: l, n, which define size of factorization,

$$\begin{array}{c} N-n \updownarrow \\ n \updownarrow \end{array} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & S_{22} \end{bmatrix} \begin{bmatrix} I_1 & A_{11}^{-1}A_{12} \\ 0 & I_2 \end{bmatrix} \quad \widetilde{\mathrm{Im}}_n = \mathrm{span}\begin{bmatrix} \widetilde{A_{11}^{-1}}A_{12} \\ -I_2 \end{bmatrix}^{\perp}.$$

- projection : $P_n^{\perp} : \mathbb{R}^N \to \widetilde{\mathrm{Im}}_n$
- solution in subspace, $\widetilde{A}_{N-l}^{\dagger} b = \begin{bmatrix} \widetilde{A_{11}^{-1}} b_1 \\ 0 \end{bmatrix}$, $b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \begin{array}{c} \updownarrow N-l \\ \updownarrow l \end{array}$

$\widetilde{A_{11}^{-1}}$: computed in quadruple-precision with perturbation to simulate double-precision round-off error.
n : candidate of dimension of the kernel
compute for $l = n-1, n, n+1$

$$\mathrm{err}_l^{(n)} := \max\left\{ \max_{x=[0\ x_l]\neq 0} \frac{||P_n^{\perp}(\widetilde{A}_{N-l}^{\dagger} A\,x - x)||}{||x||}, \max_{x=[x_{N-l}\ 0]\neq 0} \frac{||\widetilde{A}_{N-l}^{\dagger} A\,x - x||}{||x||} \right\}$$

$$\begin{array}{ccccccc} n = k+1 & \Leftrightarrow & \mathrm{err}_k^{(k+1)} \approx 0 & \wedge & \mathrm{err}_{k+1}^{(k+1)} \approx 0 & \wedge & \mathrm{err}_{k+2}^{(k+1)} \sim 1 \\ n = k & \Leftrightarrow & \mathrm{err}_{k-1}^{(k)} \gg 0 & \wedge & \mathrm{err}_k^{(k)} \approx 0 & \wedge & \mathrm{err}_{k+1}^{(k)} \sim 1 \\ n = k-1 & \Leftrightarrow & \mathrm{err}_{k-2}^{(k-1)} \gg 0 & \wedge & \mathrm{err}_{k-1}^{(k-1)} \gg 0 & \wedge & \mathrm{err}_k^{(k-1)} \sim 1 \end{array}$$

Theoretically $\neg A_{N-k+1}^{-1}$, but $\mathrm{err}_{k-1}^{(k)}$ is computable; $\sim ||\widetilde{A_N^{-1}} A_N - I_N||$.

LDU factorization with rank-b update

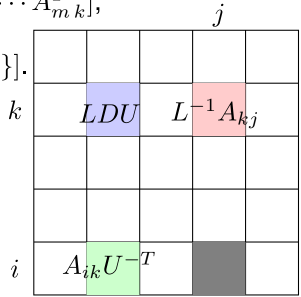
$$\begin{bmatrix} A_{11} & A_{21} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21}A_{11}^{-1} & S_{22} \end{bmatrix} \begin{bmatrix} I_1 & A_{11}^{-1}A_{12} \\ 0 & I_2 \end{bmatrix}$$
$$= \begin{bmatrix} L_{11} & 0 \\ A_{21}U_{11}^{-1}D_{11}^{-1} & I_2 \end{bmatrix} \begin{bmatrix} D_{11} & 0 \\ 0 & S_{22} \end{bmatrix} \begin{bmatrix} U_{11} & D_{11}^{-1}L_{11}A_{12}^{-1} \\ 0 & I_2 \end{bmatrix}$$

Schur complement $S_{22} = A_{22} - A_{21}A_{11}^{-1}A_{12}$ rank-b update by `dgemm`

LDU-factorization algorithm by block strategy

`do` $k = 1, \cdots, m = N/b$

LDU-factorization $A_{kk} = L_{kk}D_{kk}U_{kk}$ with symmetric pivoting.
solve multiple-RHS $L_{kk}[\{X_j\}] = [A_{k\,k+1}, \cdots A_{k\,m}]$,
solve multiple-RHS $U_{kk}^T[\{Y_j^T\}] = [A_{k+1\,k}^T, \cdots A_{m\,k}^T]$,
scaling $[\{X_j\}]$=$D_{kk}[\{X_j\}]$,
rank-b update $[\{S_{i\,j}\}] = [\{A_{i\,j}\}] - [\{Y_j^T X_i\}]$.



- for modern CPU (multi-arithmetic units)
- parallelization

? limited search of pivots

task dependency analysis : example by dense matrix (Dissection)

LDL^T factorization

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ & A_{22} & A_{23} & A_{24} \\ & & A_{33} & A_{34} \\ & & & A_{44} \end{bmatrix}$$

dtrsm + dcaling

dgemm

$$\begin{bmatrix} A_{22} & A_{23} & A_{24} \\ & A_{33} & A_{34} \\ & & A_{44} \end{bmatrix} -= \begin{bmatrix} A_{21} \\ A_{31} \\ A_{41} \end{bmatrix} A_{11}^{-1} \begin{bmatrix} A_{12} & A_{13} & A_{14} \end{bmatrix}$$

$\alpha^{(1)}$:	factorization	$L_{11} D_{11} L_{11}^T = A_{11}^{(1)}$
$\beta_k^{(1)}$:	DTRSM + scaling	$D_{11}^{-1}(L_{11}^{-1} A_{1\,k}^{(1)})$
$\gamma_{k,\,l}^{(1)}$:	DGEMM	$A_{k,\,l} -= (L_{11}^{-1} A_{1\,k})^T D_{11}^{-1} (L_{11}^{-1} A_{1\,l}^{(1)})$
$\alpha^{(2)}$:	factorization	$L_{22} D_{22} L_{22}^T = A_{22}^{(2)}$

$\alpha^{(1)} \leftarrow \{\beta_2^{(1)}, \beta_3^{(1)}, \beta_4^{(1)}\} \leftarrow \{\gamma_{2,2}^{(1)}, \gamma_{2,3}^{(1)}, \gamma_{3,3}^{(1)}, \cdots ,\gamma_{4,4}^{(1)}\} \leftarrow \alpha^{(2)} \leftarrow \{\beta_3^{(2)}, \beta_4^{(2)}\} \leftarrow$
$\{\gamma_{3,3}^{(2)}, \gamma_{3,4}^{(2)}, \gamma_{4,4}^{(2)}\} \leftarrow \alpha^{(3)} \cdots$

factorization is critical task, others depending on.

$\Downarrow$

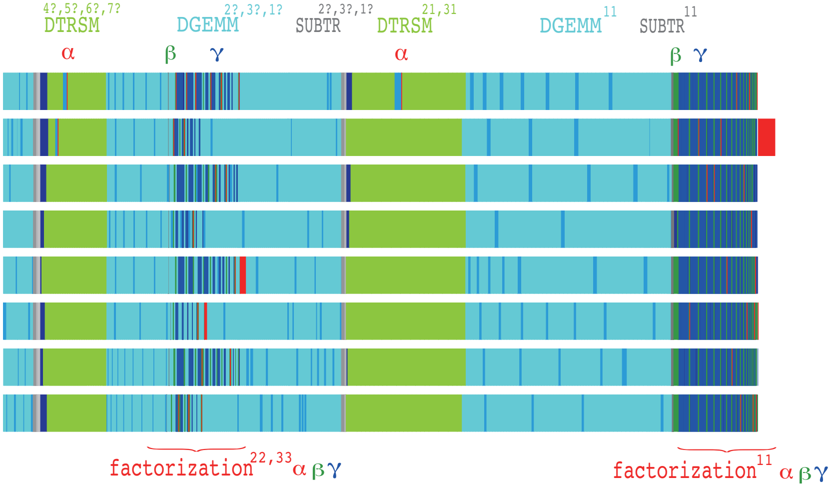
cf. DAG + runtime

$\alpha^{(1)} \leftarrow \{\beta_2^{(1)}\text{-}\gamma_{2,2}^{(1)}\text{-}\alpha^{(2)}, \beta_3^{(1)}, \beta_4^{(1)}\} \leftarrow \{\gamma_{2,3}^{(1)}, \gamma_{3,3}^{(1)}, \cdots ,\gamma_{4,4}^{(1)}\} \leftarrow$
$\{\beta_3^{(2)}\text{-}\gamma_{3,3}^{(2)}\text{-}\alpha^{(3)}, \beta_4^{(2)}\} \leftarrow \{\gamma_{3,4}^{(2)}, \gamma_{4,4}^{(2)}\} \leftarrow$

$\beta_2^{(1)}\text{-}\gamma_{2,2}^{(1)}\text{-}\alpha^{(2)}$: atomic

critical path is analyzed $\Rightarrow$ asynchronous task execution

asynchronous task execution on multi-core (Dissection)



tasks of Schur complement in nested-dissection level 3, factorization in level 2 are scheduled together.

tasks on critical path are scheduled statically. others : 20% dynamic.

sparse matrix format : 1/3

n : # of rows

nnz : # of nonzeros

$[A]_{i\,j}$: nonzero entries at $(i,\ j)$

- COO (Coordinate) format `MUMPS`

```
int irow[nnz];
int jcol[nnz];
double coef[nnz];
```

- CSR (Compressed Sparse Row) /
  CRS (Compressed Row Storage) format `Pradiso`, `Dissection`

```
int ptrow[n+1];
int indcol[nnz];
double coef[nnz];
```

$[A]_{i\,j}$=`coef`[k]

j=`indcol`[k], `ptrow`[i] $\leq k <$ `ptrow`[$i+1$]

sparse matrix format, zero-based index : 2/3

an example, 5×5 unsymmetric matrix, $n = 5$, $nnz = 15$.

$$
\boxed{\begin{array}{ccccc}
1.1 & 1.2 & & 1.4 & \\
2.1 & 2.2 & 2.3 & & 2.5 \\
 & 3.2 & 3.3 & & \\
4.1 & & & 0.0 & 4.5 \\
 & 5.2 & & 5.4 & 5.5
\end{array}}
\qquad
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & & 2 & \\
1 & 3 & 4 & 5 & & 6 \\
2 & & 7 & 8 & & \\
3 & 9 & & & 10 & 11 \\
4 & & 12 & & 13 & 14
\end{array}
$$

i	0			1				2		3			4			5
`ptrow[`i`]`	0			3				7		9			12			15
`indcol[`k`]`	0	1	3	0	1	2	4	1	2	0	3	4	1	3	4	
`coef[`k`]`	1.1	1.2	1.4	2.1	2.2	2.3	2.5	3.2	3.3	4.1	0.0	4.5	5.2	5.4	5.5	

- diagonal entry should exist even if the value is 0

  `Pardiso`, `Dissection`
- `indcol[]` should be in ascending order in each row

sparse matrix format, zero-based index : 3/3

5×5 symmetric matrix, upper triangular, $n = 5$, $nnz = 10$.

$$
\boxed{\begin{matrix}
1.1 & 1.2 & & 1.4 & \\
 & 2.2 & 2.3 & & 2.5 \\
 & & 3.3 & & \\
 & & & 0.0 & 4.5 \\
 & & & & 5.5
\end{matrix}}
\qquad
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & & 2 & \\
1 & & 3 & 4 & & 5 \\
2 & & & 6 & & \\
3 & & & & 7 & 8 \\
4 & & & & & 9
\end{array}
$$

i	0			1			2	3		4	5
`ptrow[`i`]`	0			3			6	7		9	10
`indcol[`k`]`	0	1	3	1	2	4	2	3	4	4	
`coef[`k`]`	1.1	1.2	1.4	2.2	2.3	2.5	3.3	0.0	4.5	5.5	

- upper triangular matrix is accepted by `Pardiso`
- `MUMPS` accepts COO in upper or lower triangular sparse matrix with 1-based index in any order in the array of triplet

usage of Paridso from C/C++

```
MKL_INT *ptrow = new MKL_INT[n + 1]; // CSR data
MKL_INT *indcol = new MKL_INT[nnz];
double *coef = new double[nnz];
double *x = new double[n]; // solution
double *y = new double[n]; // RHS
void *pt[64];          // to keep internal pointers
MKL_INT *iparm = new MKL_INT[64]; // parameters!
MKL_INT mtype = 11;  // structurally symmetric
MKL_INT nrhs = 1;
MKL_INT phase;
MKL_INT maxfct = 1, mnum = 1, msglvl = 1, error;
MKL_INT idum;          // dummy pointer instaed of user
                       // providing permutation
phase = 11;  // symbolic factorization
pardiso(pt, &maxfct, &mnum, &mtype, &phase, &n,
        (void *)coef, ptrow, indcol, &idum, &nrhs,
        iparm, &msglvl, (void *)y, (void *)x,
        &error);
phase = 22;  // numeric factorization
phase = 33;  // Fw/Bw substitution
phase = -1;  // free working data
```

usage of MUMPS from C/C++

```
MUMPS_INT *irow = new MUMPS_INT[nnz];
MUMPS_INT *jcol  = new MUMPS_INT[nnz];
double *coef = new double[nnz];
double *x = new double[n]; // solution&RHS
DMUMPS_STRUC_C id;
id.job = (-1); // job init
id.par = 1;
id.sym = isSym ? 2 : 0;
id.comm_fortran = USE_COMM_WORLD; // dummy MPI communicator
dmumps_c(&id);
id.job = 1; // symbolic facotrization
id.n = n; id.nz = nnz; id.irn = irow; id.jcn = jcol;
// id.icntl[]  set parameters
dmumps_c(&id);
id.job = 2; // numeric factorization
id.a = coef;
dmumps_c(&id);
id.job = 3;  // Fw/Bw substitution
id.nrhs = 1;
id.rhs = x;
dmumps_c(&id);
id.job = -2;  // free working data
```

usage of Dissection (fortran-interface) from C/C++

GPLv3 + link exception

```
int *ptrow = new int[n + 1]; // CSR data
int *indcol = new int[nnz];
double *coef = new double[nnz];
double *x = new double[n]; // solution & RHS
uint64_t *dslv = new uint64_t; // Dissection solver object
int real_or_complex = 1;       // 1 : real, 2 : complex
int verbose=1;               // write messages
int call = 0;
diss_init(*dslv, call, real_or_complex, num_threads, verbose);
int sym = 0;                 // unsymmetric, 1: symmetric upper
int ordering = 0;            // 0 : SCOTCH, 1 : METIS
diss_s_fact(*dslv, n, ptrow, indcol, sym, ordering);
int scaling = 1;             // diagonal scaling
int indefinte_flag = 1;      // 0 for positive semi-definite
double eps_pivot = 1.0e-2; // threshold for postponing
diss_n_fact(*dslv, coef, scaling, eps_pivot, indefinite_lfag);
int projection = 1;          // find solution in the image space
int transpose = 0;           // 1 for A^T x = b
diss_solv_1(*dslv, x, projectoin, transpose);
diss_free(*dslv);            // clean the Dissection solver obj.
```

usage of Dissection (C++-interface)

GPLv3/CeCILL-C

```
typedef dd_real quadruple;    // long double for SX
int *ptrow = new int[n + 1];  // CSR data
int *indcol = new int[nnz];
quadruple *coef = new quadruple[nnz];
quadruple *x = new quadruple[n]; // solution & RHS
FILE *fp;                        // file to keep messages
DissectionSolver<quadruple> *dslv =
  new DissectionSolver<quadruple>(num_threads, true, 0, fp);
bool isSym = false;
bool upper_flag = false; // true for isSym = true
bool isWhole = false;    // true for isSym = true
dslv->SymbolicFact(n, (int *)ptrow, (int *)indcol,
                   isSym, upper_flag, isWhole, ordering);
int scaling = 1;         // diagonal scaling
double eps_pivot = 1.0e-2;
bool kernel_detection_all = false;
dslv->NumericFact(0, (quadruple *)coef, scaling,
                  eps_pivot, kernel_detection_all);
bool projection = true;  // solution in the image space
bool isTrans = false;    // true for A^T x = b
bool isScaling = true;   // with internal scaling
dslv->SolveSingle(x, projection, isTrans, isScaling);
delete dslv;
```

Hands-on tutorial : 1/7

in the frontend of CMC, `login.hpc.cmc.osaka-u.ac.jp`

`/sc/cmc/apl/DirectSolverSeminar/` contains materials

```
-- MM-matrix/ : matrix data from https://sparse.tamu.edu
   VCC/ : sources, library, compiled objects ... for Intel Xeon
   -- include/ header files from
                    MUMPS, Dissection, SCOTCH, METIS, QD, zlib
      lib/ library files of those
      patch/ patch files to the original distribution for compilation
      MUMPS/ MUMPS 5.1.1
      dissection/ Dissection 1.0.3
      metis-5.1.0/ METIS 5.1.0
      qd-2.3.17/ QD 2.3.17 pacthed for intel compiler
      scotch_6.0.4/ SCOTCH 6.0.4
      zlib-1.2.11/ Zlib 1.2.11 (used for SCOTCH)
      tutorial/
        -- MM-pardiso.cpp, Makefile.pardiso.C++
           MM-MUMPS.cpp, MM-Dissection.cpp, ..., job.pbs
           Dissection/ : quadruple precision solver with C++
SX/ : sources, library, compiled objects ... for NEC SX-ACE
   -- ...
      tutorial/
```

Hands-on tutorial : 2/7

Makefile.Pardiso.C++

```
CXX = icpc
CCFLAGS = -O3
LD = $(CXX)

LIB_DIR_MKL = /opt/intel/compilers_and_libraries_2017/linux/mkl/lib/intel64
INC_DIR_MKL = /opt/intel/compilers_and_libraries_2017/linux/mkl/include/

all: MM-Pardiso

.cpp.o:
	$(CXX) $(CCFLAGS) -I$(INC_DIR_MKL) -I. -c $< -o $@

MM-Pardiso: MM-Pardiso.o elapsed_tme.o
	$(LD) -o MM-Pardiso -g -O3 MM-Pardiso.o elapsed_time.o \
	-Xlinker -rpath=$(LIB_DIR_MKL) -L$(LIB_DIR_MKL) \
	-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 \
	-lintlc -lsvml -lm

clean:
	rm -fr *~ *.o *.so core *.d MM-Pardiso
```

```
% mkdir VCC
% rsync -avu /sc/cmc/apl/DirectSolverSeminar/VCC/tutorial ./VCC/
% rsync -avu /sc/cmc/apl/DirectSolverSeminar/MM-matrix ./
% cd VCC/tutorial
% make -f Makefile.Pardiso.C++ all
% ldd MM-Pardiso
% qsub job.pbs
```

Hands-on tutorial : 3/7

```
% ldd MM-MUMPS
linux-vdso.so.1 =>  (0x00007fff8ff16000)
libdmumps.so => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libdmumps.so
libmumps_common.so => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libmumps_common.so
libpord.so => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libpord.so
libmpiseq.so => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libmpiseq.so
libesmumps.so => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libesmumps.so
libscotch.so => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libscotch.so
libscotcherr.so => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libscotcherr.so
libscotcherrexit.so => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libscotcherrexit.so
libmetis.so => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libmetis.so
libmkl_intel_lp64.so => $INTEL_MKL_LIB/libmkl_intel_lp64.so
libmkl_intel_thread.so => $INTEL_MKL_LIB/libmkl_intel_thread.so
libmkl_core.so => $INTEL_MKL_LIB/libmkl_core.so
libiomp5.so => $INTEL_COMPILER_LIB/libiomp5.so
libm.so.6 => /lib64/libm.so.6
librt.so.1 => /lib64/librt.so.1
libstdc++.so.6 => /usr/lib64/libstdc++.so.6
libgcc_s.so.1 => /lib64/libgcc_s.so.1
libc.so.6 => /lib64/libc.so.6
libdl.so.2 => /lib64/libdl.so.2
libifport.so.5 => $INTEL_COMPILER_LIB/libifport.so.5
libifcoremt.so.5 => $INTEL_COMPILER_LIB/libifcoremt.so.5
libimf.so => $INTEL_COMPILER_LIB/libimf.so
libsvml.so => $INTEL_COMPILER_LIB/libsvml.so
libintlc.so.5 => $INTEL_COMPILER_LIB/libintlc.so.5
libpthread.so.0 => /lib64/libpthread.so.0
libirng.so => $INTEL_COMPILER_LIB/libirng.so
libz.so.1 => /sc/cmc/apl/DirectSolverSeminar/VCC/lib/libz.so.1
/lib64/ld-linux-x86-64.so.2

INTEL_COMPILER_LIB
=/opt/intel/compilers_and_libraries_2017.2.174/linux/compiler/lib/intel64
INTEL_MKL_LIB=
/opt/intel/compilers_and_libraries_2017.2.174/linux/mkl/lib/intel64
```

Hands-on tutorial : 4/7

parameters for each solver

- Pardiso

```
./MM-Pardiso [matrix-market file] [num_threads] [perturbation]
      iparam[1] = 2; // METIS
      iparam[9] = perturbation; // 13 ↔ 10^-13
```

  parallelization is controlled by setting internal value of MKL

```
 mkl_set_num_threads(num_threads);
```

- MUMPS

```
./MM-MUMPS [matrix-market file] [ordering] [num_threads]
           [postponing_threshold] [null_ pivot]
      icntl[6] = ordering; // 3 : SOCTCH, 5 : METIS
      cntl[0] = postponing_threshold; // 1.0e-2
      cntl[2] = (-1.0)*null_pivot; // 1.0e-8
```

  parallelization is controlled by setting internal value of OpenMP

```
 omp_set_dynamic(0);
 omp_set_num_threads(num_threads);
```

- Dissection

```
./MM-Dissection [matrix-market file] [ordering] [num_threads]
                [postponig_threshold]
      ordering = 0; // SCOTCH
      postponing_threashold = 1.0e-2;
```

  parallelization is controlled by initialization call

Hands-on tutorial : 6/7

results on `CoupCons3D` with 4 cores

Pardiso

error	$3.2520097 \times 10^{-14}$	
residual	$9.8931920 \times 10^{-17}$	
	CPU time (sec.)	elapsed time (sec.)
symbolic	5.6900	4.2584
numeric	87.030	21.821
fw/bw	1.7100	0.4275

MUMPS

error	$3.1367255 \times 10^{-14}$	
residual	$2.6733498 \times 10^{-16}$	
	CPU time (sec.)	elapsed time (sec.)
symbolic	3.1900	3.1770
numeric	81.090	21.472
fw/bw	1.9400	0.4818

Dissection

error	$7.6341570 \times 10^{-14}$	
residual	$7.1339225 \times 10^{-16}$	
	CPU time (sec.)	elapsed time (sec.)
symbolic	14.040	10.936
numeric fact	86.210	22.909
fw/bw	1.1100	0.3155

Hands-on tutorial : 7/7

results on `stokes.skewsym` with 4 cores : singular, dimension 2 kernel

Pardiso

error	1.6661762×10^{-2}	
residual	$1.4771827 \times 10^{-16}$	
	CPU time (sec.)	elapsed time (sec.)
symbolic	6.3400	5.3841
numeric	13.070	3.2679
fw/bw	1.0400	0.2612

MUMPS : kernel dimension detected as 2

error	$1.0595600 \times 10^{-10}$	
residual	$1.8276515 \times 10^{-14}$	
	CPU time (sec.)	elapsed time (sec.)
symbolic	4.9000	4.8968
numeric	23.120	6.2821
fw/bw	1.6600	0.4157

Dissection : kernel dimension detected as 2

error	$7.9142093 \times 10^{-13}$	
residual	$6.2848715 \times 10^{-16}$	
	CPU time (sec.)	elapsed time (sec.)
symbolic	12.200	9.6785
numeric fact	18.980	5.4162
fw/bw	0.6500	0.1840

Web resource

- Pardiso
  `https://software.intel.com/en-us/mkl-developer-reference-c-intel-mkl-pardiso-parallel-direct-sparse-solver-interface`

  `https://software.intel.com/en-us/mkl-developer-reference-fortran-intel-mkl-pardiso-parallel-direct-sparse-solver-interface`
- MUMPS
  `http://mumps.enseeiht.fr`
- Dissection
  `http://www.freefem.org/ff++/ff++/download/dissection`