additive Schwarz preconditioner for Krylov subspace method

Atsushi Suzuki[1]

[1]R-CCS, Large-scale Parallel Numerical Computing Technology Research Team
atsushi.suzuki.aj@a.riken.jp

preconditioner for Krylov subspace method

preconditioner $Q \simeq A^{-1}$, operation of Q to residual vector $\vec{r}$ is efficient : economical, i.e, light-weight or arithmetic intensive, i.e. heavy
right preconditioned system

$$AQ(Q^{-1}\vec{x}) = \vec{b} \Leftrightarrow AQ\vec{x}' = \vec{b} \text{ initial residual } \vec{r}_0 = \vec{b} - A\vec{x}_0$$

peroconditioned GMRES

$$\begin{aligned} &\text{find } \vec{x}'_m \in Q^{-1}x_0 + K_m(AQ, \vec{r}_0) \\ &\qquad ||\vec{b} - AQ\vec{x}'_m|| \leq ||\vec{b} - AQ\vec{y}|| \quad \forall \vec{y} \in Q^{-1}x_0 + K_m(AQ, \vec{r}_0) \\ \Leftrightarrow\ &\text{find } \vec{x}_m \in x_0 + K_m(QA, Q\vec{r}_0) \\ &\qquad ||\vec{b} - A\vec{x}_m|| \leq ||\vec{b} - A\vec{y}|| \quad \forall \vec{y} \in x_0 + K_m(QA, Q\vec{r}_0) \end{aligned}$$

- diagonal preconditioner $[Q]_{ii} = |[A]_{ii}|^{-1}$
- SOR preconditioner $A = L + D + U$ with parameter $0 < \omega < 2$, $Q = (\frac{D}{\omega} + U)^{-1} \frac{2-\omega}{\omega} D (\frac{D}{\omega} + L)^{-1}$
- incomplete factorization, ILU preconditioner
- multigrid preconditioner, HYPRE, gamg in PETSc based on algebraic multigrid
- additive Schwarz preconditioner based on local direct solver in overlapping subdomain, GenEO two-level method in HPDDM

incomplete LU factorization as preconditioner

nonzero pattern $NZ(A) := \{(i,j)\,;\, a_{ij} \neq 0\}$

Algorithm (ILU(0))

do i=2,$\cdots, N$
  do k=$1 \cdots, i-1\,;\ (i,k) \in NZ(A)$
    $a_{ik} = a_{ik}/a_{kk}$
    do j=$k+1, \cdots, N\,;\ (i,j) \in NZ(A)$
      $a_{ij} = a_{ij} - a_{ik}/a_{kj}$

subroutine in intel Math Kernel Library (MKL)

```
int nrow, ierr;
double *coefs, *ilu;   // non-zero values
int *ia, *ja;          // CSR non-zero indexes
int ipar[128];         // ipar[30]=1 to conitnue for 0 diagonal
double dpar[128];      // dpar[30]=1.0e-16, dpar[31]=1.0e-10

dcsrilu0(&nrow, coefs, ia, ja, ilu, ipar, dpar, &ierr);
```

Schwarz methods as preconditioner

overlapping decomposition of the matrix with index
$\Lambda = \bigcup_{p=1}^{P} \Lambda_p,\ \Lambda_p \cap \Lambda_q \neq \emptyset$

- R_p: restriction from the total DOF to sub-matrix : $\Lambda \to \Lambda_p$
- D_p : discrete representation of partition of the unity

$$\sum_{p=1}^{P} R_p^T D_p R_p = I_N,$$

$$[D_p]_{kk} = \begin{cases} 1 & k \in \Lambda_p,\ k \notin \Lambda_q, \forall q \neq p, \\ 1/\#\{p; k \in \Lambda_p\} & \text{otherwise} \end{cases}$$

ASM preconditioner

$$M_{\text{ASM}}^{-1} = \sum_{p=1}^{P} R_p^T (R_p A R_p^T)^{-1} R_p$$

ASM does not converge as fixed point iteration, but M_{ASM}^{-1} is symmetric and works well as a preconditioner for CG method.

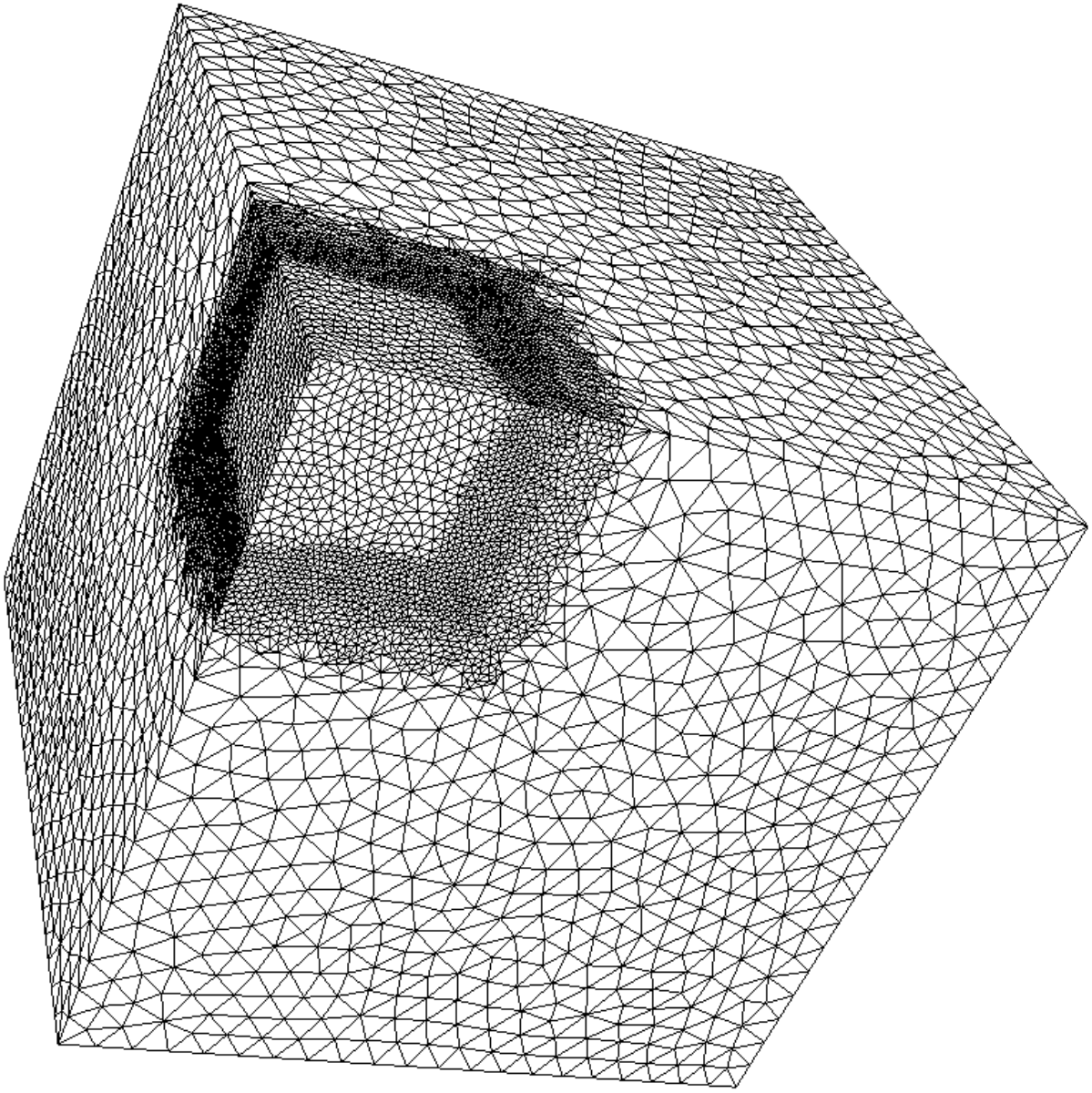
RAS preconditioner

$$M_{\text{RAS}}^{-1} = \sum_{p=1}^{P} R_p^T D_p (R_p A R_p^T)^{-1} R_p$$

RAS does converge but M_{RAS}^{-1} is not symmetric and then works as a preconditioner for GMRES method.

Numerical example : 1/3

unstructured mesh generated by `tetgen` and `FreeFEM`
P2 finite element, $N = 677,163$, $nnz = 28,853,844$



Numerical example : 2/3

stopping criteria : relative residual $\leq 10^{-12}$
Navier equations : elasticity problem discretized by P2 finite element

$N = 750,141$, $nnz = 31,214,610$: structured mesh generated from 32^3 cubes

		elapsed time (sec.)			
overlap	# iteration	total	LU-factorization	iteration	error
1	59	81.397	44.749	36.647	4.41361e-12
2	41	87.871	55.339	32.532	1.70397e-12
3	33	106.13	72.975	33.158	1.45462e-12
ILU(0)	186	58.347	16.146	42.201	4.66513e-11

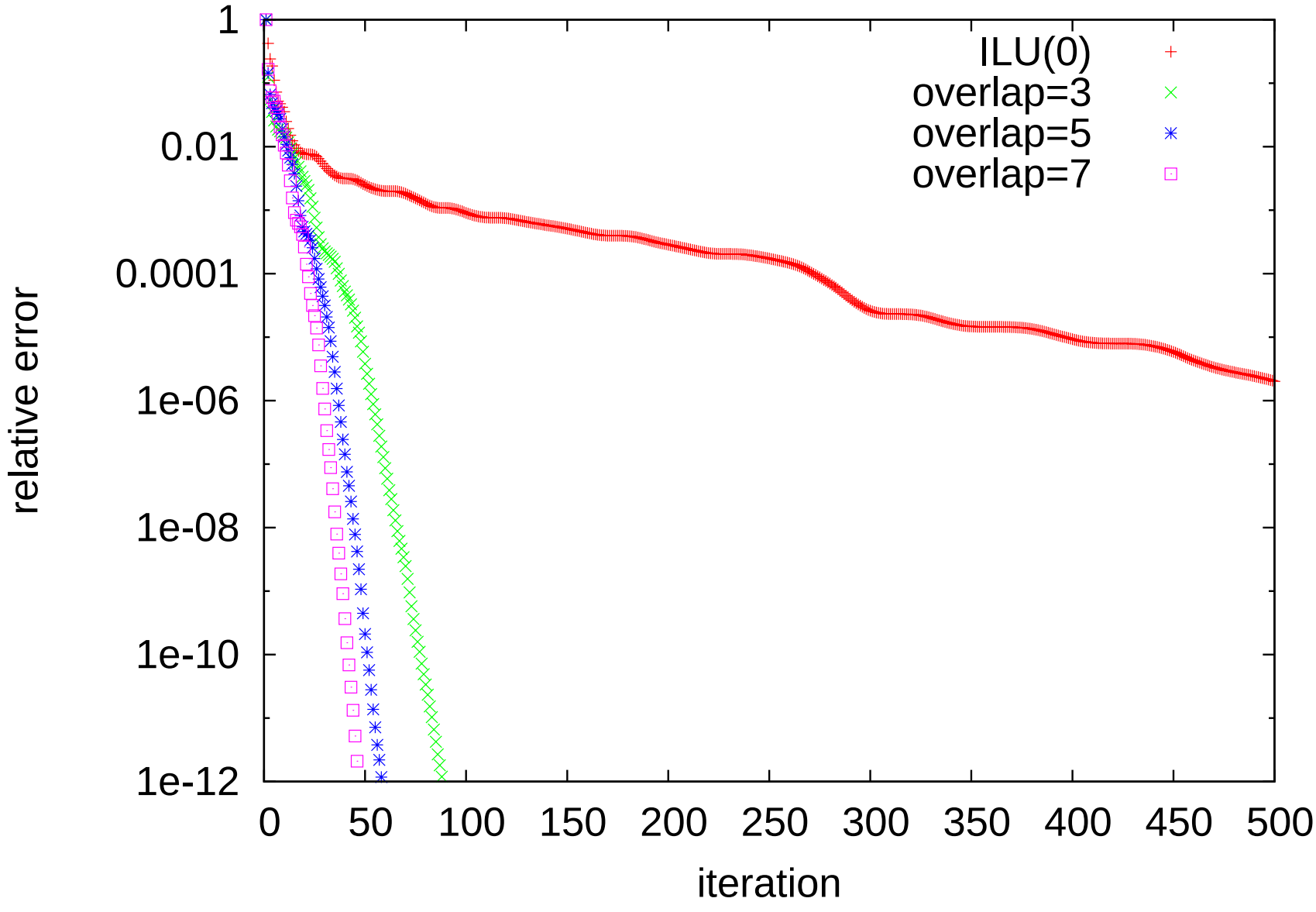
$N = 677,163$, $nnz = 28,853,844$: unstructured mesh (with mesh refinement)

		elapsed time (sec.)			
overlap	# iteration	total	LU-factorization	iteration	error
1	88	77.978	29.413	48.565	4.23959e-12
2	58	87.975	45.055	42.920	4.62264e-12
3	46	111.68	69.063	47.112	8.12039e-12
ILU(0)	500	209.91	30.791	179.12	1.12751e-2

ILU(0) preconditioner is not strong enough for unstructured mesh problem

Numerical example : 3/3

convergence history



short introduction of direct solver

coefficient matrix A will be factorized as $\Pi_L^T L\, D\, U \Pi_R$

- Π_L, Π_R : left/right permutation to realize full pivot
- symmetric pivot $\Pi_L = \Pi_R = \Pi$ is enough by adding ε perturbation and/or 2×2 pivoting

permutation Π consists of

- reordering of index to reduction of fill-in entries form symbolic information (non-zero pattern of sparse matrix)
- dynamic procedure to select the maximum entry in diagonals and exchange row and column

three phases of direct solver

- symbolic factorization
  Π_S to reduce fill-in and to introduce parallel factorization by multi-frontal method
- numeric factorization
  with Π_N dynamic pivoting
- forward/backward substitution

$$\Pi^T L\, D\, U \Pi \vec{x} = \vec{b}$$

$$\vec{z} = L^{-1} \Pi \vec{b} \qquad \text{forward substitution } L\vec{z} = \Pi\vec{b}$$

$$\vec{y} = D^{-1} \vec{z}$$

$$\vec{x} = \Pi^T U^{-1} \vec{y} \qquad \text{backward substitution } U\Pi\vec{x} = \vec{y}$$

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.

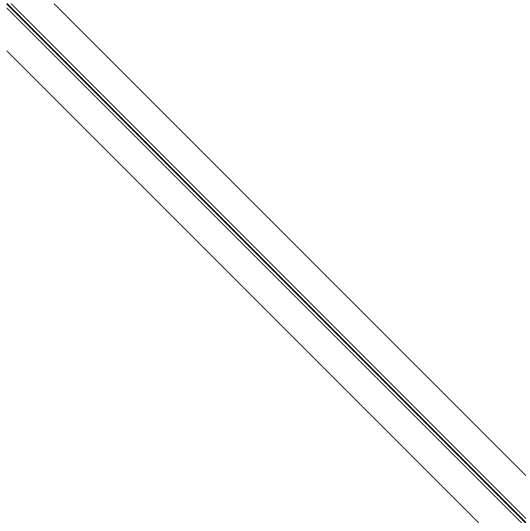
A. Suzuki, F.-X. Roux, A dissection solver with kernel detection for symmetric finite element matrices on shared memory computers,
Int. J. Numer. Meth. in Engng, 100 (2014) 136–164.

ordering of sparse matrix

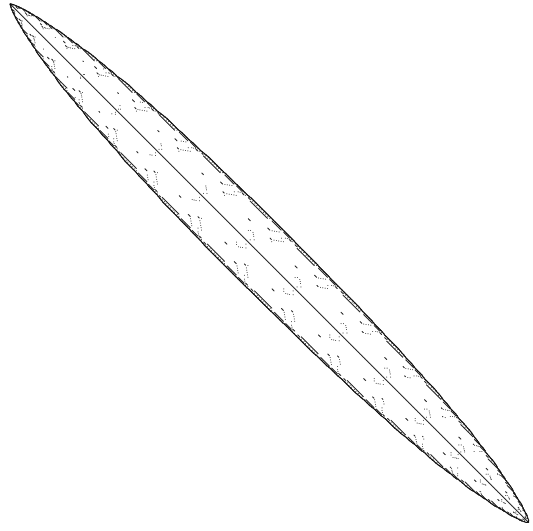
sparse matrix needs to be re-ordered

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

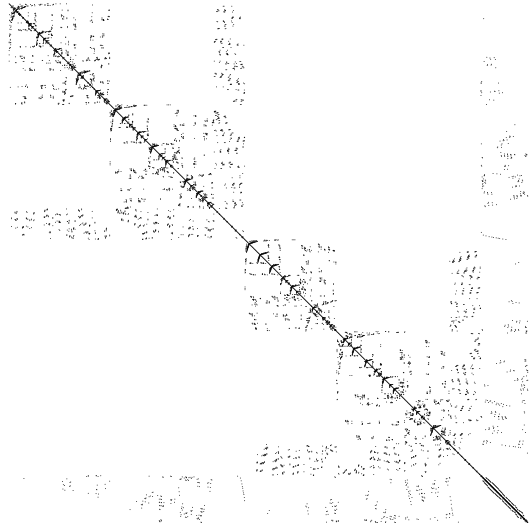
example:
7 stencil of Poisson equation in 3D, 11^3 nodes.



original matrix
band matrix

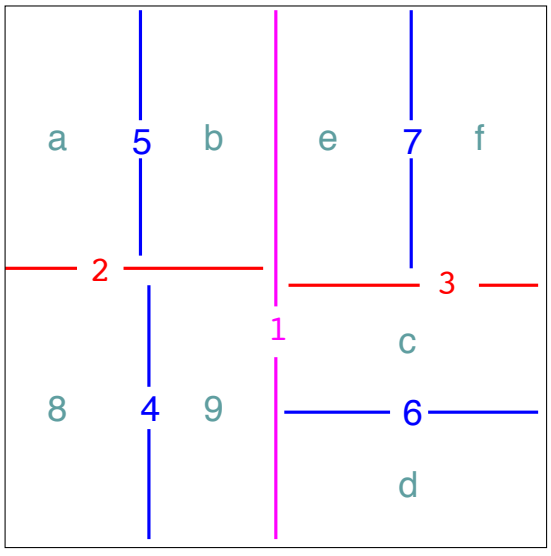


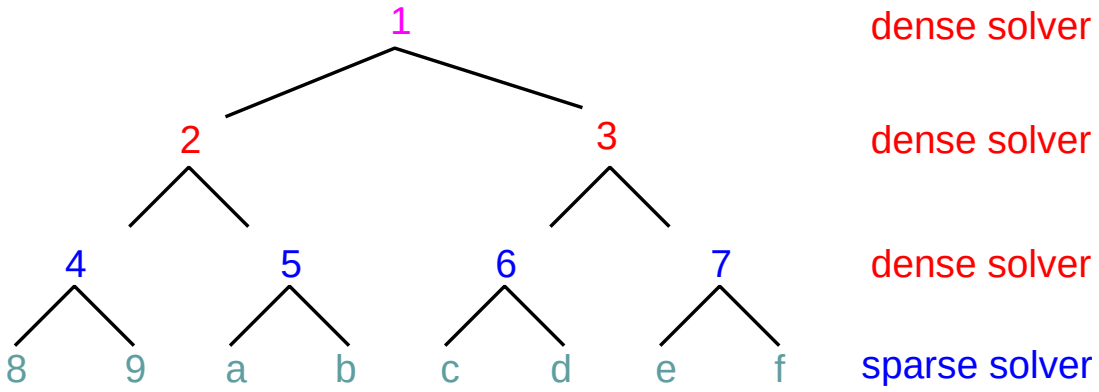
reverse Cuthill-McKee
sequential computation



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:

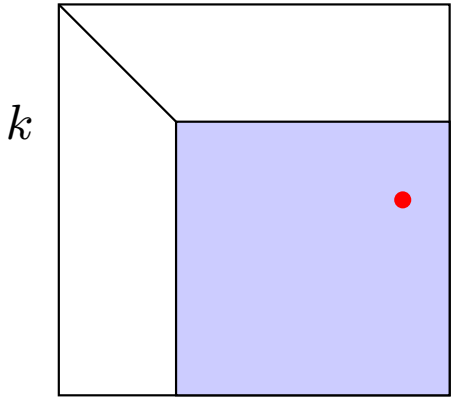
METIS : V. Kumar, G. Karypis, A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. 20 (1998) 359–392.

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.

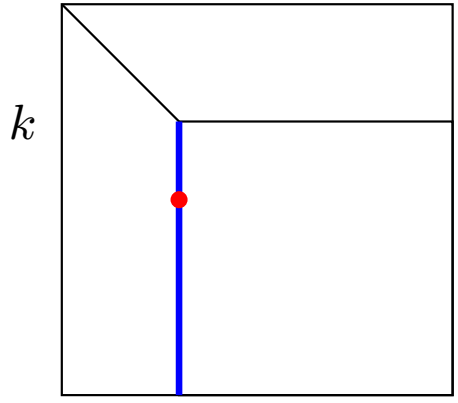
- each leaf can be computed in parallel $\Leftarrow$ multi-front

pivoting strategy

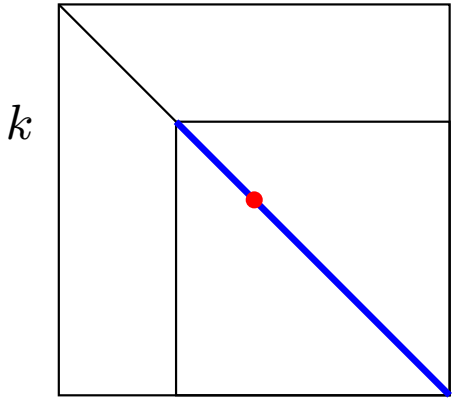
full pivoting : $A = \Pi_L^T L U \Pi_R$
find max$_{k<i,j\leq n}|A(i,j)|$



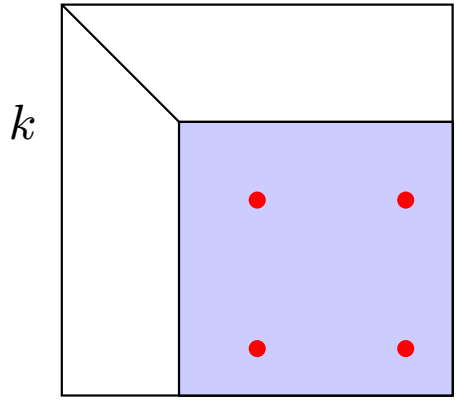
partial pivoting : $A = \Pi L U$
find max$_{k<i\leq n}|A(i,k)|$



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



2×2 pivoting : $A = \Pi^T L \tilde{D} U \Pi$
find max$_{k<i,j\leq n}$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 $k = 1, \cdots, N$

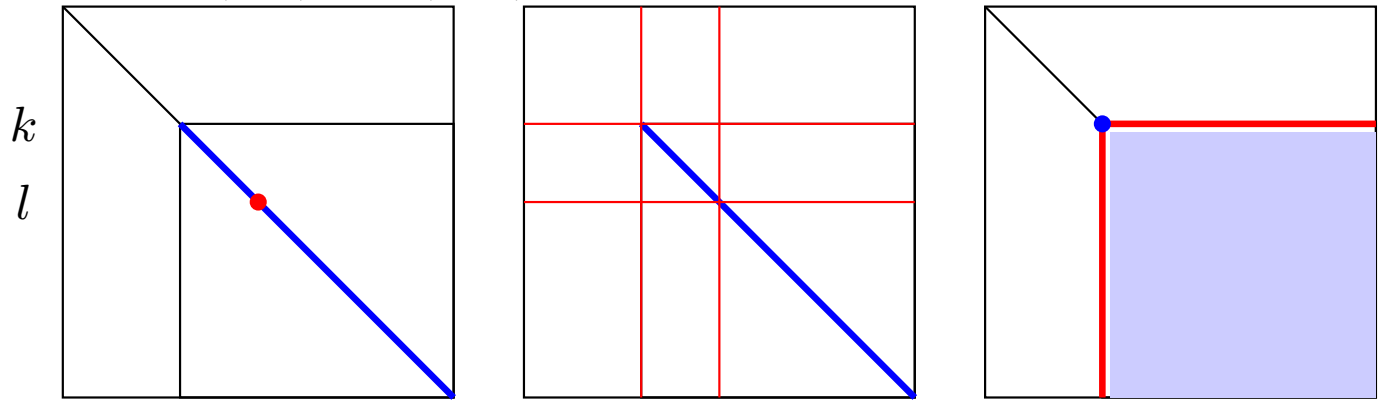
find max$|A(l,l)|$ with $k < l \leq n$,

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

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

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

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



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/should specify perturbation parameter for unsymmetric matrix (default = 10^{-13} for `Pardiso`)

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
for (int i = 0; i < 64; i++)
   pt[i] = (void *)0; // zero clear
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
pardiso(pt, &maxfct, &mnum, &mtype, &phase, &n, ...
phase = 33;  // Fw/Bw substitution
pardiso(pt, &maxfct, &mnum, &mtype, &phase, &n, ...
phase = -1;  // free working data
pardiso(pt, &maxfct, &mnum, &mtype, &phase, &n, ...
```

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
dmumps_c(&id);
```