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.

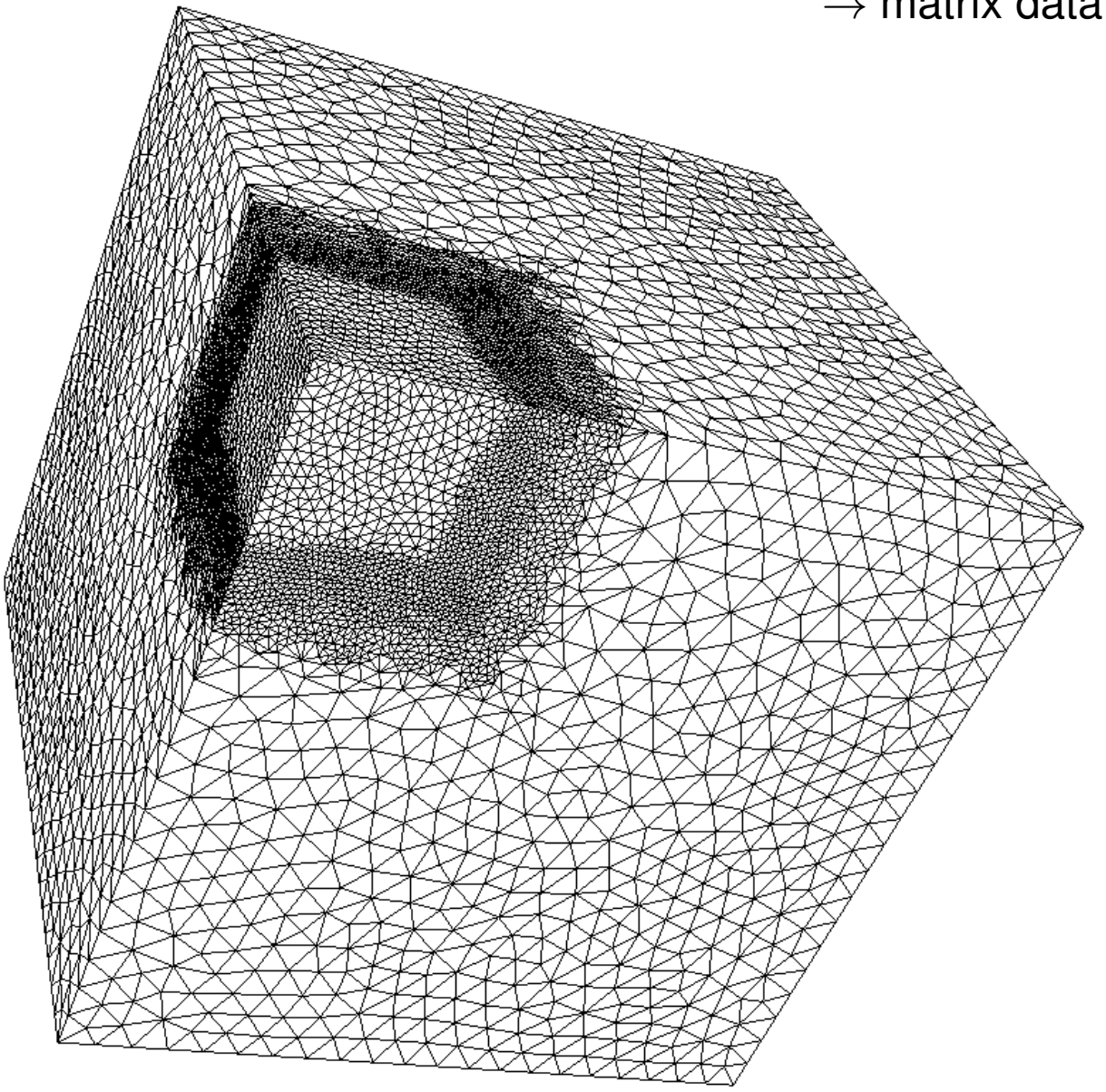
RAS preconditioner

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

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, $N = 677,163$, $nnz = 28,853,844$
$\rightarrow$ matrix data `Navier3DmeshP2`



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

`Navier3D.32.P2`

overlap	# iteration	elapsed time (sec.) 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)

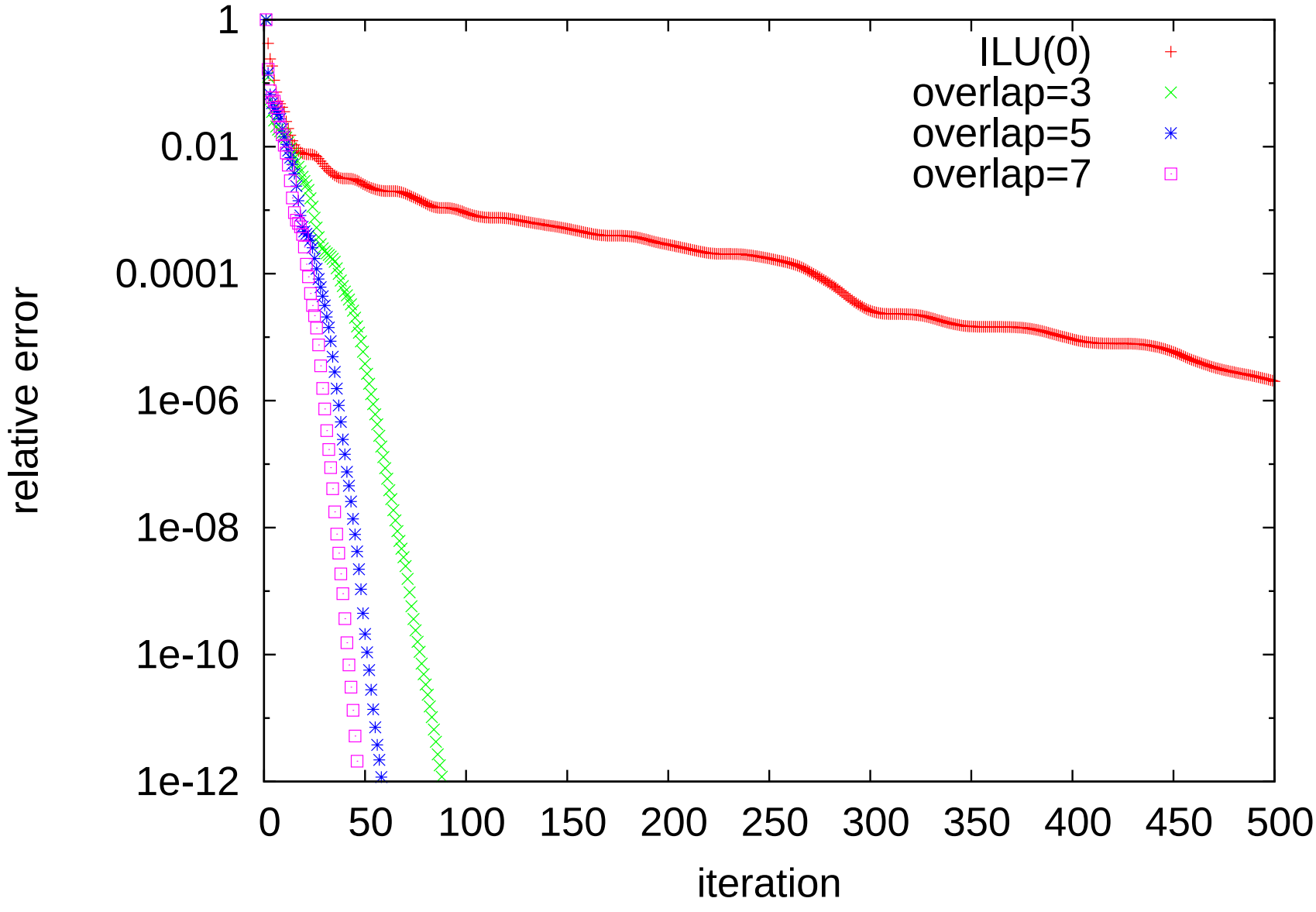
`Navier3DmeshP2`

overlap	# iteration	elapsed time (sec.) 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.

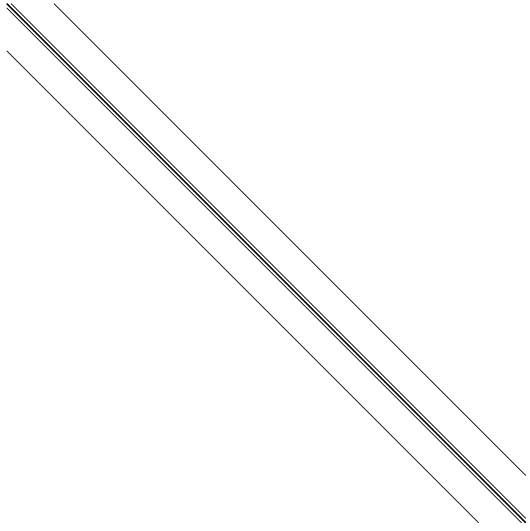
A. Suzuki, F.-X. Roux, A dissection solver with kernel detection for symmetric finite element matrices on shared memory computers,
Int. J. Numer. Meth. in Engng, 100 (2014) 136–164.

ordering of sparse matrix

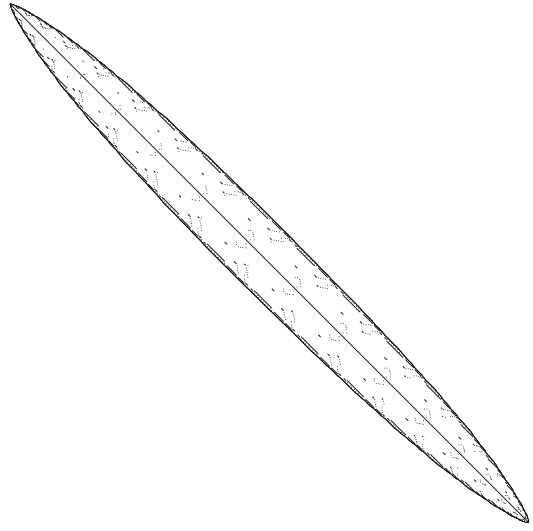
sparse matrix needs to be re-ordered

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

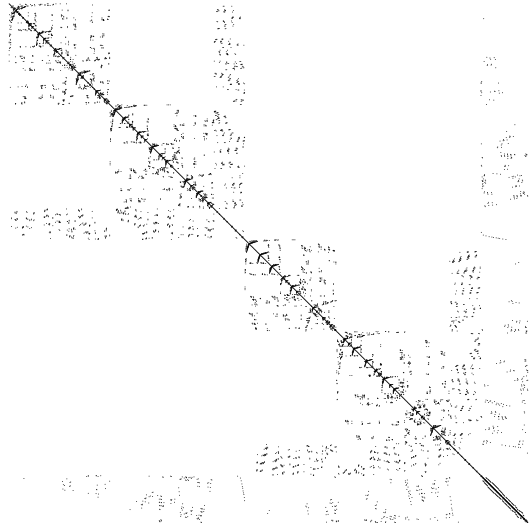
example:
7 stencil of Poisson equation in 3D, 11^3 nodes.



original matrix
band matrix

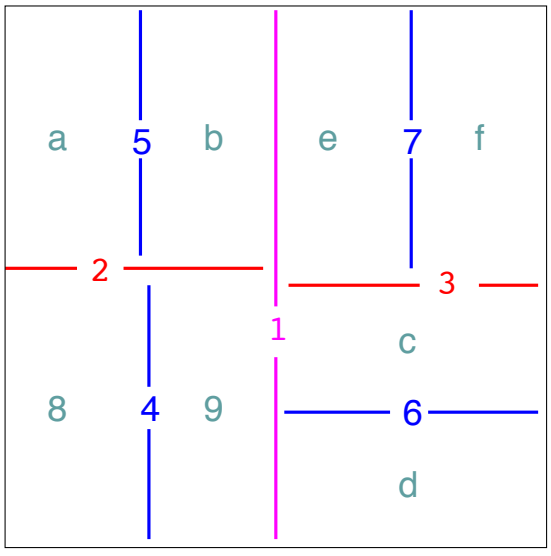


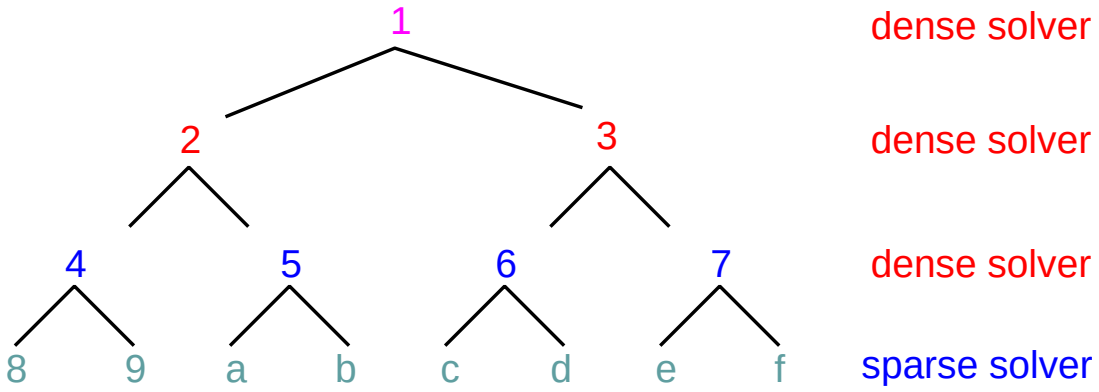
reverse Cuthill-McKee
sequential computation



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:

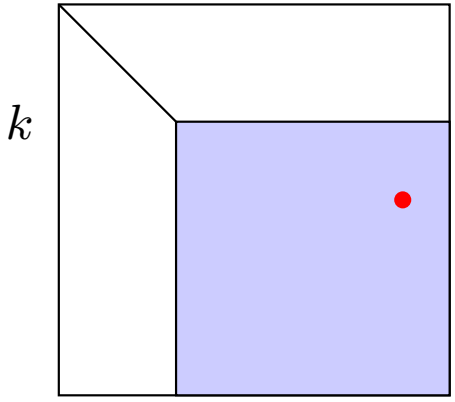
METIS : V. Kumar, G. Karypis, A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. 20 (1998) 359–392.

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.

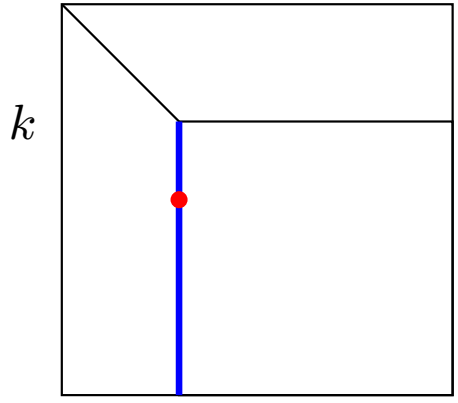
- each leaf can be computed in parallel $\Leftarrow$ multi-front

pivoting strategy

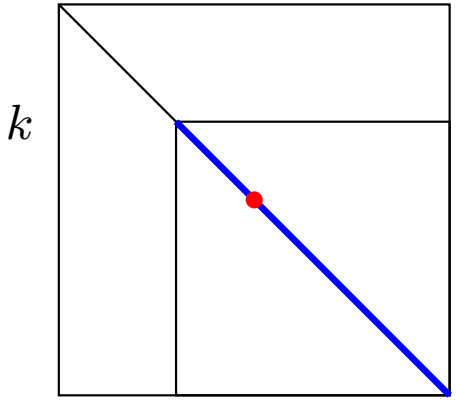
full pivoting : $A = \Pi_L^T L U \Pi_R$
find max$_{k<i,j\leq n}|A(i,j)|$



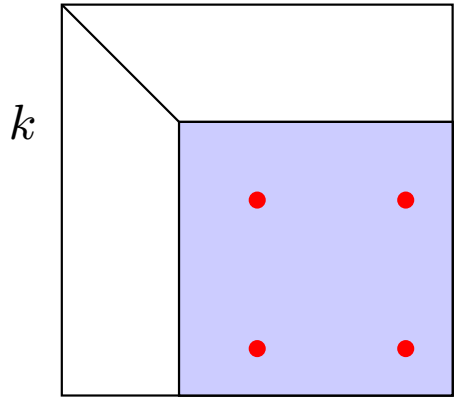
partial pivoting : $A = \Pi L U$
find max$_{k<i\leq n}|A(i,k)|$



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 $k = 1, \cdots, N$

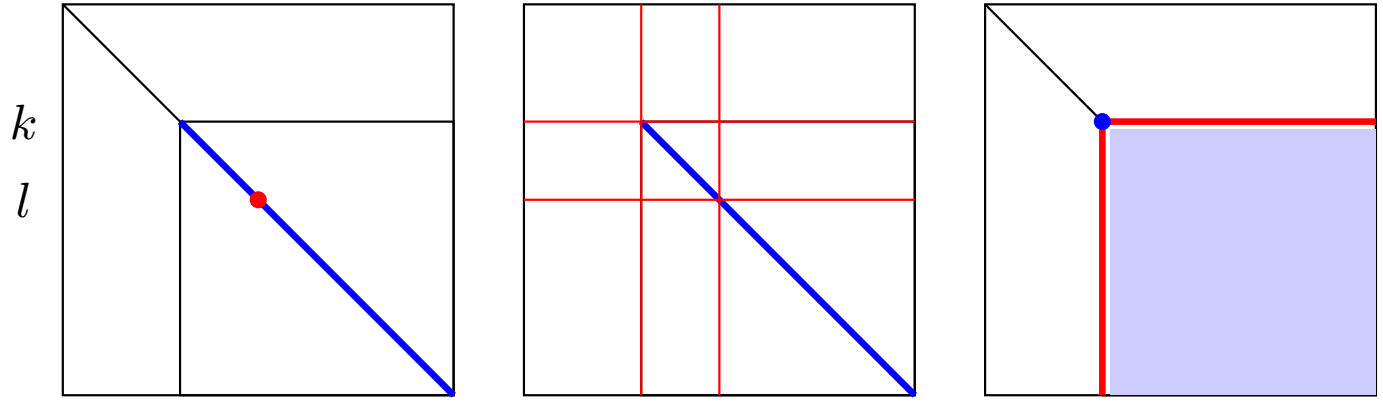
find max$|A(l,l)|$ with $k < l \leq n$,

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

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

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

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



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`)

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
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
```

non overlapping decomposition of the matrix by METIS

```
int nrow, nnz;
int xadj[nrow];          // connectivity of the sparse matrix
int adjcy[nnz - nrow]; // excluding diagonal from CSR
int part[nrow];
int ncon = 1, objval;
idx_t options[METIS_NOPTIONS] = {0} ;
METIS_SetDefaultOptions(options);
options[METIS_OPTION_NUMBERING] = 0;
options[METIS_OPTION_DBGLVL] = METIS_DBG_INFO;
METIS_PartGraphRecursive(&nrow, &ncon, xadj, adjcy,
                         NULL, NULL, NULL,
                         &nparts,
                         NULL, NULL,
                         options, &objval, part);
```

index $\Lambda = \{1, 2, \cdots, N\}$ is decomposed into a union of non-overlapping indices Λ_p

$$\Lambda = \oplus_{1 \leq q \leq P} \Lambda_p \quad \Lambda_p \cap \Lambda_p = \emptyset$$

array `part[`k`]` $1 \leq k \leq N$ contains subdomain index $1 \leq q \leq Q$

$$\Lambda_p = \{q \in \{1, \cdots, N\}\,;\, \exists k \ \texttt{part[}k\texttt{]} = q\}$$

creation of overlapping decomposition

starting from non-overlapping decomposition $\Lambda_p^{(0)} = \Lambda_p$ satisying
$\Lambda = \oplus_{1 \le p \le P} \Lambda_p \quad \Lambda_p \cap \Lambda_q = \emptyset$,

$$\Lambda_p^{(l+1)} = \{ j \, ; \, [A]_{i\,j} \neq 0 \;\; i \in \Lambda_p^{(l)} \}$$

$\Lambda_p = \Lambda_p^{(0)} \subset \Lambda_p^{(1)} \subset \cdots$ are generated
updating of mask vector is performed as

```
structure CSRformat acsr;
std::vector<std::vector<int > > mask[nparts];
for (int n = 0; n < nparts; n++) mask[n].resize(nrow, 0);

for (int n = 0; n < nparts; n++) {
  for (int i = 0; i < nrow; i++)
     mask[part[n][i]] = 1;
  for (int ll = 0; ll < noverlap; ll++) {
    std::vector<int> itmp(mask[n]);
    for (int i = 0; i < nrow; i++) {
      if (itmp[i] == 1) {
        for (int k = acsr.ptrow[i]; k < acsr.ptrow[i + 1]; k++)
          mask[n][acsr.indcol[k]] = 1;
      }
    }
  }
}
```

creation of a partition of the unity

dsicrete partition of the unity

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

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

from the `mask` array, restriction R_p and weight D_p are generated as

```
std::vector<std::vector<int> > local2global(nparts);
std::vector<double> weightPTU(nrow, 0.0);

for (int n = 0; n < nparts; n++) {
  for (int i = 0; i < nrow; i++) {
    if (mask[n][i] == 1) {
      local2globall[n].push_back(i);
      weightPTU[i] += 1.0;
    }
  }
}
for (int i = 0; i < nrow; i++) {
  weightPTU[i] = 1.0 / weightPTU[i];
}
```

extraction of local matrix is performed by using `mask[n]` and `local2global[n]`

data distribution of sparse matrix with overlapping subdomains : 1/4

form restriction operator R_p, local matrix is defined as

$$A_p = R_p A R_p^T, \quad A = \sum_p R_p^T D_p A_p R_p$$

matrix-vector product is performed as

$$\begin{aligned} \vec{y} = A\vec{x} &= \sum_p R_p^T D_p A_p R_p \vec{x} \\ &= \sum_p R_p^T D_p A_p \vec{x}_p \end{aligned}$$

$\vec{x}_p$: restricted (local) vector with index Λ_p

- local SpMV $\vec{y}_p = A_p \vec{x}_p$
- gathering operation with weight D_p, $\vec{y} = \sum_p R_p D_p \vec{y}_p$

gathering operation requires some communication among subdomains assigned to different processors

data distribution of sparse matrix with overlapping subdomains : 2/4

simplest implementation to perform gathering operation $\vec{y} = \sum_p R_p D_p \vec{y}_p$

- $\vec{y}_p = A_p \vec{x}_p$ is performed locally
- prepare global array $\vec{z}_p \in \mathbb{R}^N$ with zero padding outside of index Λ_p

$$
\begin{aligned}
[\vec{z}_p]_{i \neq \Lambda_p} &= 0 \\
[\vec{z}_p]_{\lambda_p(j)} &= [\vec{y}_p]_j \quad \lambda_p : \{1, \cdots, N_p\} \to \Lambda_p \subset \{1, \cdots, N\}
\end{aligned}
$$

perform reduction operation for all vectors $\vec{z}_p$

```
MPI_Allreduce(z_p, y, nrow, MPI_DOUBLE, MPI_SUM,
              MPI_COMM_WORLD);
```

values at index only belong to interior of the subdomain

$\Lambda_{p,I} \cap \Lambda_q = \emptyset \ (\forall q \neq p)$ receives accumulation of zero values from other subdomain due to zero padding

data distribution of sparse matrix with overlapping subdomains : 3/4

eliminating unnecessary operation with adding zero values
decomposition of index into interior and interface of the subdomain

$$\Lambda_p = \Lambda_{p,I} \oplus \Lambda_{p,B} \quad \Lambda_{p,I} \cap \Lambda_q = \emptyset \quad \forall q \neq p \ \exists r \Lambda_{p,B} \cap \Lambda_r \neq \emptyset$$

by this decomposition, gathering operation can avoid zero addition

$$\begin{aligned} \vec{y} &= \sum_p R_p^T D_p \vec{y}_p \\ &= \left(\{ R_{p,I}^T D_{p,I} \vec{y}_{p,I} \}, \sum_q R_{q,B}^T D_{q,B} \vec{y}_{q,B} \right) \end{aligned}$$

separation of $\Lambda_p = \Lambda_{p,I} \cap \Lambda_{p,B}$ is obtained from the weight of the partition of the unity

$$\begin{aligned} i \in \Lambda_{p,I} &\Leftrightarrow [D_p]_i = 1 \\ i \in \Lambda_{p,B} &\Leftrightarrow [D_p]_i < 1 \end{aligned}$$

```
std::vector<std::vector<int> > localI2global(nparts);
std::vector<std::vector<int> > localB2global(nparts);
// n fixed
for (std::vector<int>::iterator it = local2global[n].begin();
  it != local2global[n].end(); ++it) {
  if (weightPTU((*it) == 1.0)
    localI2global[n].push_back((*it));
  else
    localB2global[n].push_back((*it));
}
```

data distribution of sparse matrix with overlapping subdomains : 4/4

calculation of inner product does not require data transfer on vectors between subdomain

$$
\begin{aligned}
(\vec{x}, \vec{y}) =& (\vec{x}, \sum_p R_p^T D_p \vec{y}_p) \\
=& \sum_p (R_p \vec{x}, D_p \vec{y}_p) \\
=& \sum_p (\vec{x}_p, D_p \vec{y}_p) \\
=& \sum_p (\vec{x}_{p,I}, \vec{y}_{p,I}) + \sum_p (\vec{x}_{p,B}, D_{p,B} \vec{y}_{p,B})
\end{aligned}
$$

```
MPI_Allreduce(local, global, 1, MPI_DOUBLE, MPI_SUM,
              MPI_COMM_WORLD);
```

exercise : implement GMRES/CG using IML++

GMRES is written by `C++` template in `IML++`,
`https://math.nist.gov/iml++/gmres.h.txt`
using additive Schwarz preconditoner for `const Preconditioner &M`
replacing Operator, Vector, and Matrix classes by simpler ones

- `Real` by `double`
- `Vector` by `std::vector<double>`
- `Operator` by `structure CSRformat`
- `Matrix` by `std::vector<double>` as one-dimensionalized array

matrix vector product to compute the residual as `r = b - A * x;` will be replaced by

```
std::vector<double> r(b);
SparseGEMV(A, (-1.0), x, 1.0, r);
```

first version in sequential
second version in parallel

```
int mpi_id, mpi_procs;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &mpi_id);
MPI_Comm_size(MPI_COMM_WORLD, &mpi_procs);

MPI_Allreduce(&tmps[0], &rhs[0], nrow, MPI_DOUBLE, MPI_SUM,
              MPI_COMM_WORLD);

MPI_Finalize();
```

exercise : matrix data

FEM matrix data are located in `/work/gt90/t90044/matrix-data/`
stopping criteria 10^{-12}, ovelap $\ell = 3$

data	Poisson3D.data	Stokes3D.data	RayleighBenard3D.data
matrix	positive sym.	indefinite sym.	indefinite unsym.
N	499,280	187,218	68,082
nnz	13,854,290	17,195,764	7,542,968
# iter	27	55	72
error	9.19315×10^{-13}	3.47688×10^{-8}	1.43101×10^{-8}
residual	8.40642×10^{-13}	5.32961×10^{-13}	7.66600810^{-13}
total time	18.625	26.515	9.3780
MPI comm.	3.686	0.2110	0.3613

PETSc library : 1/5

PETSc : a suite of data structures and routines for scalable parallel solution of linear/nonliear system for partial differential equations developed by Argonne National Laboratory

- KSP : Krylov subspace method with various preconditioners

ksp/tutorial/ex1.c

```
#include <petscksp.h>

int main(int argc, char **args)
{
  Vec         x, b, u; /* approx solution, RHS, exact solution */
  Mat         A;       /* linear system matrix */
  KSP         ksp;     /* linear solver context */
  PC          pc;      /* preconditioner context */
  PetscInt    i, n = 10, col[3], its;
  PetscMPIInt size;
  PetscScalar value[3];

  PetscInitialize(&argc, &args, (char *)0, help);
  MPI_Comm_size(PETSC_COMM_WORLD, &size);

  PetscOptionsGetInt(NULL, NULL, "-n", &n, NULL);
  // set up for vectors
  MatCreate(PETSC_COMM_SELF, &A);
  MatSetSizes(A, PETSC_DECIDE, PETSC_DECIDE, n, n);
  MatSetFromOptions(A);
  MatSetUp(A);
```

PETSc library : 2/5

ksp/tutorial/ex1.c matrix set up and solver options

```
value[0] = -1.0;
value[1] = 2.0;
value[2] = -1.0;
for (i = 1; i < n - 1; i++) {
  col[0] = i - 1;
  col[1] = i;
  col[2] = i + 1;
  MatSetValues(A, 1, &i, 3, col, value, INSERT_VALUES);
}
// two column data for i = 0 and i = n
MatSetValues(A, 1, &i, 2, col, value, INSERT_VALUES);
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);

VecSet(u, 1.0);
MatMult(A, u, b);

KSPCreate(PETSC_COMM_SELF, &ksp);
KSPSetOperators(ksp, A, A);
KSPGetPC(ksp, &pc);
PCSetType(pc, PCJACOBI);
KSPSetTolerances(ksp, 1.e-5, PETSC_DEFAULT, PETSC_DEFAULT, PETSC_DEF
KSPSetFromOptions(ksp);
KSPSolve(ksp, b, x);
```

PETSc library : 3/5

- after end of `MatSetVaules()`, `MatAssenblyBegin()` and `MatAssenblyEnd()` need to be called
- `PCSetType` sets preconditioner as `PCType`
- `PCType` provides several kinds of precondioners `PCJACOBI`, `PCILU`, `PCSOR`, `PCGAMG`, `PCASM`
- KSPSetFromOptions set parameters of Krylov subspace method from command line

```
-ksp_type <type> -pc_type <type> -ksp_monitor -ksp_rtol
<rtol>
```

PETSc library : 4/5

ksp/tutorial/ex8.c an example for additive Schwarz preconditioner

```
 #include <petscksp.h>

int main(int argc, char **args)
{
  Vec x, b, u;          /* approx solution, RHS, exact solution */
  Mat A;                /* linear system matrix */
  KSP ksp;              /* linear solver context */
  PC  pc;               /* PC context */
  IS  *is, *is_local;  //
  PetscInt    overlap = 1;    /* width of subdomain overlap */
  PetscInt    Nsub;           /* number of subdomains */
  PetscInt    m = 15, n = 17; /* mesh dimensions in x- and y- directio
  PetscInt    M = 2, N = 1;   /* number of subdomains in x- and y- dir
  PetscInt    i, j, Ii, J, Istart, Iend;
  PetscMPIInt size;

  PetscInitialize(&argc, &args, (char *)0, help);
  MPI_Comm_size(PETSC_COMM_WORLD, &size);

  MatCreate(PETSC_COMM_WORLD, &A);
  MatSetSizes(A, PETSC_DECIDE, PETSC_DECIDE, m * n, m * n);
  MatSetFromOptions(A);
  MatSetUp(A);
  MatGetOwnershipRange(A, &Istart, &Iend);
  for (Ii = Istart; Ii < Iend; Ii++) {
    //  MatSetValues(A, 1, &Ii, 1, &J, &v, INSERT_VALUES);
```

PETSc library : 5/5

ksp/tutorial/ex8.c an example for additive Schwarz preconditioner

```
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);

KSPCreate(PETSC_COMM_WORLD, &ksp);
KSPSetOperators(ksp, A, A);
KSPGetPC(ksp, &pc);
PCSetType(pc, PCASM);
PCASMSetOverlap(pc, overlap);
KSPSetFromOptions(ksp);
KSPSolve(ksp, b, x);
```

user defined decomposition of the matrix

- 2D decomposition with `M×N` for `m×n` grid

```
PCASMCreateSubdomains2D(m, n, M, N, 1, overlap, &Nsub,
                        &is, &is_local);
PCASMSetLocalSubdomains(pc, Nsub, is, is_local);
```

- user gives

```
PCASMSetLocalSubdomains(pc, Nsub, is, is_local);
```

number of subdomains `Nsub`
index for overlapping subdomain `IS *is`
index for interior subdomain without overlap `IS *is_local`

CSR data format and METIS decomposition for PETSc : 1/2

- KSP : Krylov subspace method with various preconditioners
- PCASM : additive Schwarz preconditioner

modification of ksp/tutorial/ex8.c
to use CSR data and overlapping subdomains by METIS

```
struct csr_matrix {
  PetscInt nrow, nnz;
  std::vector<PetscInt> ia, ja;
  std::vector<PetscReal> coefs;
};

// read OCC data from matrix market file format
// convert from COO to csr_matrix a

  PetscCall(MatCreateSeqAIJWithArrays(PETSC_COMM_SELF, nrow, nrow,
                                      &a.ia[0], &a.ja[0], &a.coefs[0], &A));

  PetscCall(MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY));
  PetscCall(MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY));
```

- `MatCreateSeqAIJWithArrays()` directly creates matrix `Mat *mat` from CSR format data `PetscInt i[]`, `PetscInt j[]`, `PetscScalar a[]`

CSR data format and METIS decomposition for PETSc : 2/2

modification of ksp/tutorial/ex8.c
to use CSR data and overlapping subodamins by METIS

```
// creataion of non-overlapping decomposition by METIS
// creation of overlapping decomposition by adding layers

  PetscInt    Nsub;            // number of subdomains

  std::vector<std::vector<PetscInt> >local2glob0(Nsub);
  std::vector<std::vector<PetscInt> >local2glob(Nsub);

  is = new IS[Nsub];
  is_local = new IS[Nsub];
  for (PetscInt i = 0; i < Nsub; i++) {
    PetscCall(ISCreateGeneral(PETSC_COMM_SELF, local2glob[i].size(),
                      &local2glob[i][0], PETSC_COPY_VALUES, &is[i]));
    PetscCall(ISCreateGeneral(PETSC_COMM_SELF, local2glob0[i].size(),
              &local2glob0[i][0], PETSC_COPY_VALUES, &is_local[i]));

  }
  PetscCall(PCASMSetLocalSubdomains(pc, 1, is, is_local));
```

- ISCreateGeneral converts `PescInt idx[]` array to PETSc data structure for an index set `IS`