parallel implementation of additive Schwarz preconditioner

Atsushi Suzuki[1]

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

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 by enlarging procedure,

$$\Lambda = \oplus_{1 \le p \le P} \Lambda_p \quad \Lambda_p \cap \Lambda_q = \emptyset \ (p \le q)$$

array `part[`k`]` $1 \le k \le N$ contains subdomain index $1 \le p \le P$

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

creation of overlapping decomposition

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

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

$\Lambda_p = \Lambda_p^{(0)} \subset \Lambda_p^{(1)} \subset \cdots$ 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[n][part[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 : 1/5

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, $\vec{y} = \sum_p R_p^T D_p \vec{y}_p$

gathering operation requires some communication among subdomains assigned to different processors

data distribution of sparse matrix with overlapping subdomains : 2/5

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

- $\vec{y}_p = A_p \vec{x}_p$ 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 : 3/5

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

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

by this decomposition, gathering operation can avoid zero addition

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

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 : 4/5

subdomain-wise data update with neighboring communication

$$\Lambda_p = \Lambda_{p,I} \oplus \Lambda_{p,B} \quad \Lambda_{p,I} \cap \Lambda_q = \emptyset \quad \forall q \neq p \text{ and } \exists r \; \Lambda_{p,B} \cap \Lambda_r \neq \emptyset$$
$$\Lambda_{p,r} = \Lambda_{p,B} \cap \Lambda_{r,B} \quad \text{data communication between } p \text{ and } r$$

map between index set for local subdomain and its boudnary

$$\{1, \cdots, N_p\} \ni i \mapsto \lambda_p(i) \in \Lambda_p \subset \Lambda \qquad \text{global index}$$
$$\{1, \cdots, N_{p,B}\} \ni i \mapsto \lambda_{p,B}(i) \in \Lambda_{p,B} \subset \Lambda_p \qquad \text{global index}$$
$$\{1, \cdots, N_{p,B}\} \ni i \mapsto \widehat{\lambda}_{p,B}(i) \in \{1, \cdots, N_p\} \quad \widehat{\Lambda}_{p,B} = \{\widehat{\lambda}_{p,B}(i)\,;\, i \in \{1, \cdots, N_{p,B}\}\}$$

$\widehat{\Lambda}_{p,r} \subset \widehat{\Lambda}_{p,B}$: local index set on the boundary which shares data with $\widehat{\Lambda}_{r,B}$

- on domain p, packing data with $\widehat{\Lambda}_{p,r}$
- send packed data to domain r
- receive packed data from domain r
- on domain p, unpacking data with $\widehat{\Lambda}_{p,r}$

```
MPI_Isend( &senddata[0], nsenddata, MPI_DOUBLE, destrank, 0,
           MPI_COMM_WORLD, &requests[0]);
MPI_Irecv( &recvdata[0], nsenddata, MPI_DOUBLE, srcrank, 0,
           MPI_COMM_WORLD, &requests[1]);
MPI_Waitall( 2, &requests[0], &statuses[0]);
```

data distribution of sparse matrix with overlapping subdomains : 5/5

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

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

summing up scalar data in all processors

```
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` → `double`
- `Vector` → `std::vector<double>`
- `Operator` → `structure CSRformat`
- `Matrix` → `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

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 gmres -pc_type ilu -ksp_monitor -ksp_rtol
1.0e-6
```

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`