IT4Innovations national supercomputing center

PRACE AUTUMN SCHOOL 2016
September 27-30, 2016
Hagenberg, Austria

PRACE Autumn School 2016
PETSc tutorial

Part V: Matrices

Václav Hapla
IT4Innovations
VSB– Technical University of Ostrava
Ostrava, Czech Republic

Matrix (Mat)

```
Mat A;   PetscInt m=2, n=3, M=8, N=12;
MatType type=MATMPIAIJ;
MPI_Comm comm=PETSC_COMM_WORLD;
```

- **create**: `MatCreate(comm, &A);`
- **layout**: `MatSetSizes(A,m,n,M,N);`
- **type**: `MatSetType(A,type);`
- **prealloc:** `MatMPIAIJSetPreallocation(A,5,PETSC_NULL,5,PETSC_NULL);`

- **setup:** `MatSetUp(A);`
- **options:** `MatSetFromOptions(A);`
- **dealloc**: `MatDestroy(&A);`

Matrix (Mat)


```
Mat A;    PetscInt m=2, n=3, M=8, N=12;
MatType type=MATMPIAIJ;                          // MATSEQAIJ,MATAIJ
MPI_Comm comm=PETSC_COMM_WORLD;                   // PETSC_COMM_SELF
```

- **create**: `MatCreate(comm, &A);`
- **layout**: `MatSetSizes(A,m,n,M,N);` `// MatSetSizes(v,M,N,M,N);`
- **type**: `MatSetType(A,type);`
- **prealloc:** `MatMPIAIJSetPreallocation(A,5,PETSC_NULL,5,PETSC_NULL);`
  `// MatSeqAIJSetPreallocation(A,5,PETSC_NULL);`

- **setup:** `MatSetUp(A);`
- **options:** `MatSetFromOptions(A);`
- **dealloc**: `MatDestroy(&A);`

Matrix (Mat)


```
Mat A;    PetscInt m=2, n=3, M=8, N=12;
MatType type=MATMPIAIJ;                          // MATSEQAIJ,MATAIJ
MPI_Comm comm=PETSC_COMM_WORLD;                   // PETSC_COMM_SELF
```

- **create**: `MatCreate(comm, &A);`
- **layout**: `MatSetSizes(A,m,n,M,N);` `// MatSetSizes(v,M,N,M,N);`
- **type**: `MatSetType(A,type);`
- **prealloc**: `MatMPIAIJSetPreallocation(A,5,PETSC_NULL,5,PETSC_NULL);`
  `// MatSeqAIJSetPreallocation(A,5,PETSC_NULL);`
- **all in one**: `MatCreateMPIAIJ(comm,m,n,M,N,5,PETSC_NULL,5,PETSC_NULL,&A);`
- **setup**: `MatSetUp(A);`
- **options**: `MatSetFromOptions(A);`
- **dealloc**: `MatDestroy(&A);`

Assembled matrix types

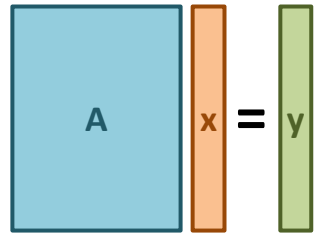
= explicit, element-wise, ... e.g. stiffness matrix, Jacobian, …

- `MATAIJ, MATSEQAIJ, MATMPIAIJ`
  - basic sparse format, known as compressed row format, CRS, Yale
  - `MATAIJ` means `MATSEQAIJ` with a single process communicator, and `MATMPIAIJ` otherwise.
- `MATBAIJ, MATSEQBAIJ, MATMPIBAIJ`
  - extension of the AIJ format described above
  - store matrix elements by small fixed-sized dense blocks (that fit into cache)
  - intended especially for use with PDEs
  - multiple DOFs per mesh node
- `MATDENSE, MATSEQDENSE, MATMPIDENSE`
  - plain dense matrices (stored column-wise like - FORTRAN compatible)

Parallel layout

matrix-vector product

MatMult(A,x,y);



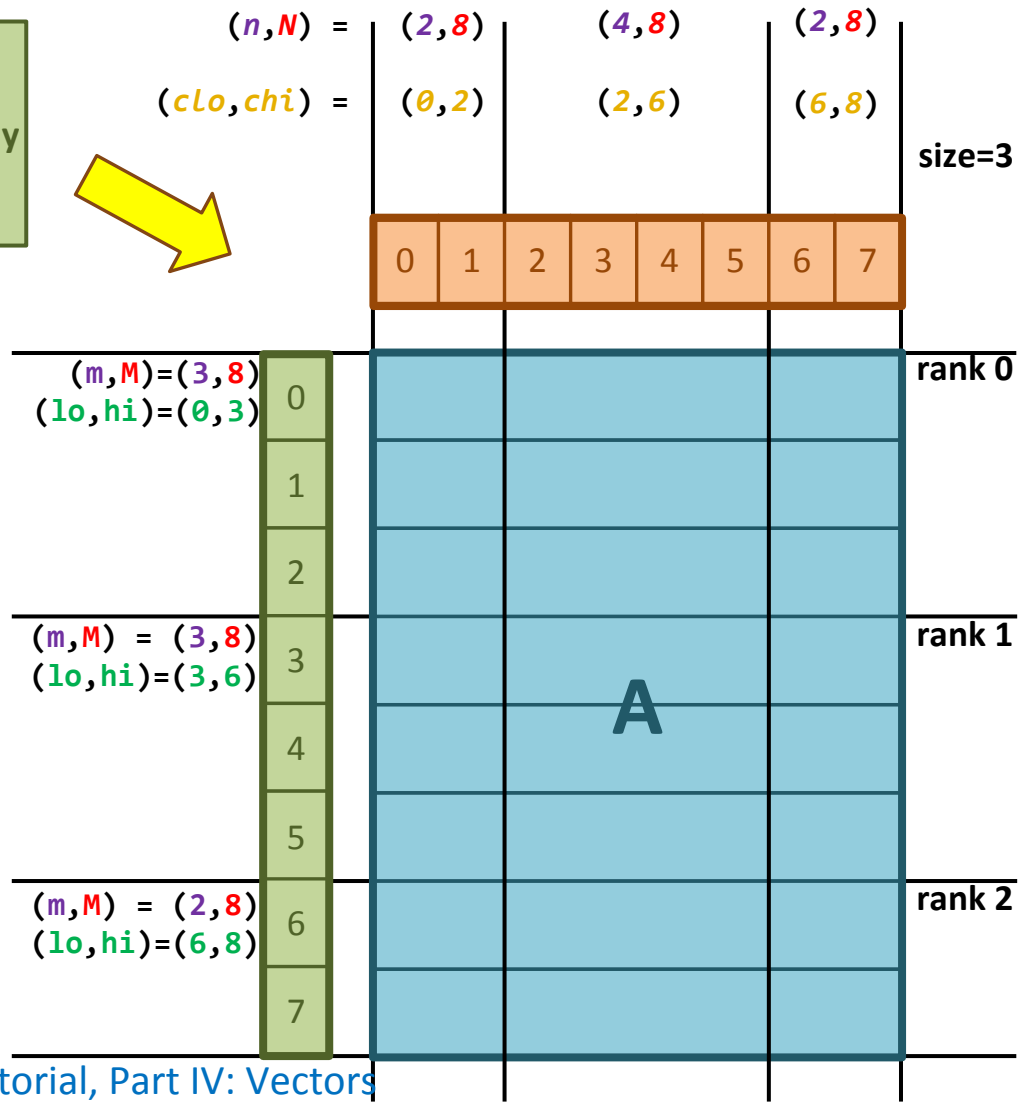
local sizes

global sizes

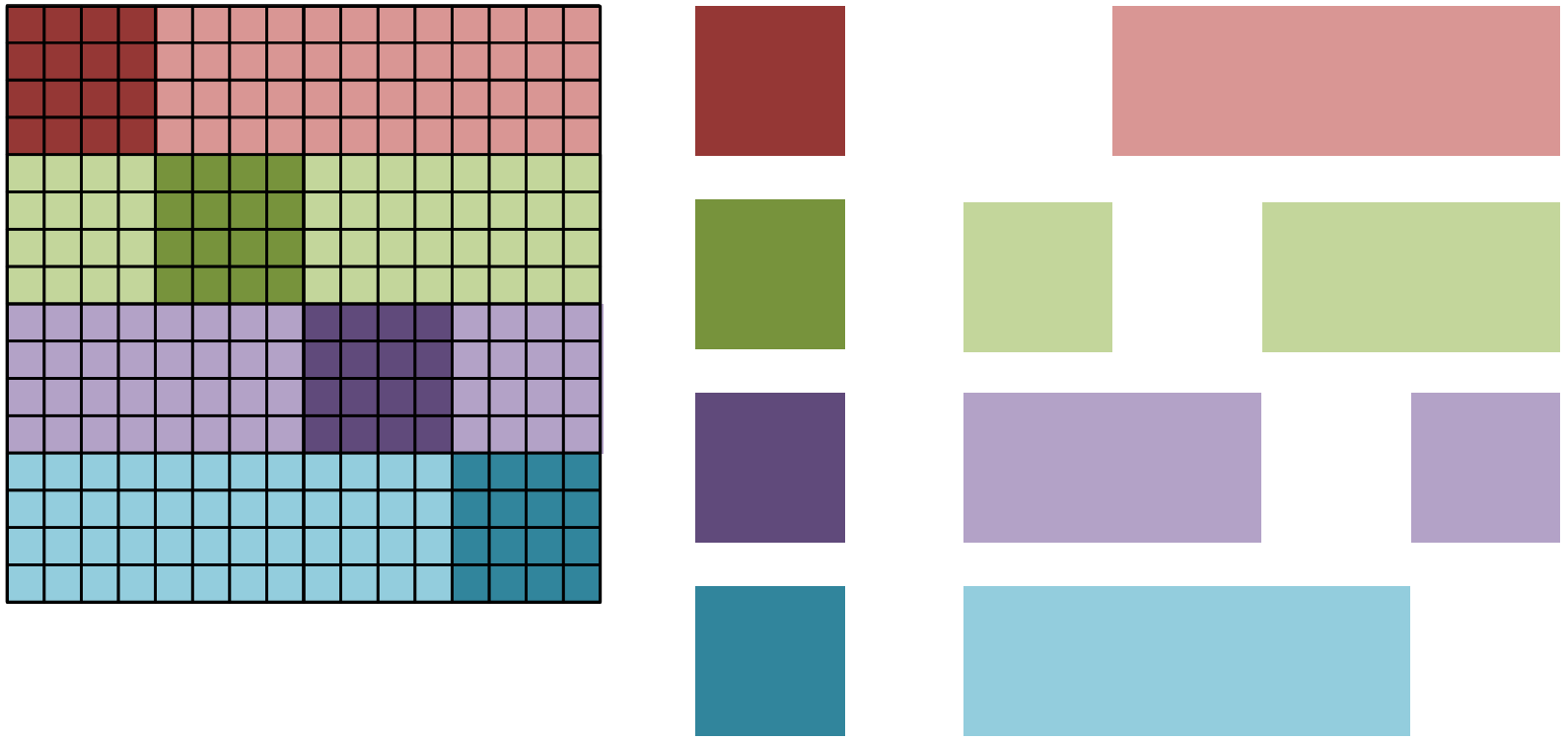
```
MatSetSizes(A,m,n,M,N);
  PetscSplitOwnership(comm,&m,&M);
  PetscSplitOwnership(comm,&n,&N);
MatGetSizes(A,&M,&N);
MatGetLocalSizes(A,&m,&n);

MatGetOwnershipRange(A,&lo,&hi);
VecGetOwnershipRange(y,&lo,&hi);
MatGetOwnershipRangeColumn(A,&clo,&chi);
VecGetOwnershipRange(x,&clo,&chi);

MatCreateVecs(A,&x,&y);
MatMult(A,x,y);
```



MPIAIJ storage



Matrix assembly (1)

```
Mat A;
```

- Set **all** (allocated) entries to **0**:
```
MatZeroEntries(A);
```
- Set an **individual** element (global indexing !):
```
PetscInt i = 1, j = 2;  PetscReal v = 3.14;
MatSetValue(A, i, j, v, INSERT_VALUES); // one value
    // or
MatSetValues(A, 1, &i, 1, &i, &v, INSERT_VALUES);
    // array of values
```

Matrix assembly (2)

- Set **multiple entries at once**:

```
PetscInt ii[2]={1, 2}, jj[2]={11, 12};
PetscReal vv[4]={1.3, 2.7, 3.1, 4.5};
MatSetValues(A, 2, ii, 2, jj, vv, INSERT_VALUES);
```

- The `last argument` can be
  - `INSERT_VALUES` **replace** original value **(=)**
  - `ADD_VALUES` **add** the new values to the original values **(+=)**

Matrix assembly (3)

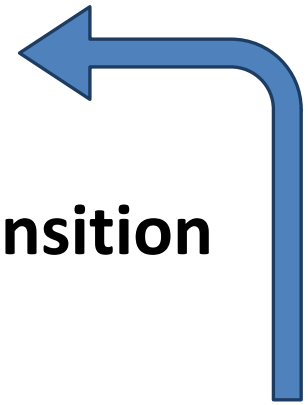
- `MatSetValues` is **purely local** with **no** inter-process **communication**
- values are just locally **cached**
- before using the vector, call **assembly function pair** to **exchange** values **between processors**:

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

Matrix assembly (3)

- `MatSetValues` is **purely local** with **no** inter-process **communication**
- values are just locally **cached**
- before using the vector, call **assembly function pair** to **exchange** values **between processors**:

```
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
//optional calculations not involving A
MatAssemblyEnd(  A,MAT_FINAL_ASSEMBLY);
```



- **computations** can be done **while** MPI **messages** are **in transition**
- this allows **overlapping communication** and **computation**

Matrix assembly (4)

- **cannot mix inserting** and **adding** values
- need to do **assembly** between

```
MatSetValues(A, ..., INSERT_VALUES);
MatAssemblyBegin(A, MAT_FLUSH_ASSEMBLY);
MatAssemblyEnd(  A, MAT_FLUSH_ASSEMBLY);
MatSetValues(A, ..., ADD_VALUES);
```

- `MAT_FINAL_ASSEMBLY` – final assembly to make A ready to use
- `MAT_FLUSH_ASSEMBLY` – cheaper,
  suffices for `INSERT_VALUES/ADD_VALUES` interleaving

Getting values

```
Mat A;
```

- get a **copy** of the 3x2 **local block** of `A` with **global row indices** `ii` and **global column indices** `jj` to an array `v`:

```
PetscInt ii[]={11, 22, 33};  PetscInt jj[]={12, 24};  PetscScalar v[6];
MatGetValues(A,3,ii,2,jj,v);
```

- get the **row** of the matrix `A`:

```
PetscInt nnz, *nzi;  PetscScalar *vals;
MatGetRow(    A, i, &nnz, &nzi, &vals);
// read the array vals (don't alter!)
MatRestoreRow(A, i, &nnz, &nzi, &vals);
```

Some more matrix operations

- **single matrix**

```
MatNorm(A, NORM_INFINITY, &norm);          //NORM_1,NORM_FROBENIUS
MatScale(A, 2.2);                          // A = 2.2 * A
MatTranspose(A,flg,&At);                   // At = A'
```

- **matrix-vector**

```
MatMult(A,x,y);                            // y = A*x
MatMultTranspose(A,x,y);                   // y = A'*x
MatMultAdd(A,x,y,z);                       // z = y + A*x
MatMultTransposeAdd(A,x,y,z);              // z = y + A'*x
```

- **matrix-matrix**

```
MatMatMult(          A,B,flg,fill,&C);     // C = A*B
MatMatTransposeMult(A,B,flg,fill,&C);      // C = A*B'
MatTransposeMatMult(A,B,flg,fill,&C);      // C = A'*B
                  //flg is either MAT_INITIAL_MATRIX or MAT_REUSE_MATRIX
```

Hands-on: ex4.c

1. `cd ~/petsc/exercises`
2. `make ex4`
3. `mpiexec -n 3 ./ex4`
4. look at the assembly code
5. try different views of the matrix
   - programatically - see documentation of `PetscViewerPushFormat`
   - from command-line

     ```
     -mat_view
     -mat_view ::ascii_dense
     -mat_view ::ascii_info
     -mat_view ::ascii_info_detail
     ```
6. change the type of matrix to dense at runtime (`-mat_type dense`)
7. try some basic operations (`MatMult, MatScale, …`)

Thanks for your attention.