IT4Innovations national supercomputing center

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

PRACE Autumn School 2016
PETSc tutorial

Part IV: Vectors

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

Vector (Vec)

```
Vec v;  PetscInt m=2, M=8;
VecType type=VECMPI;
MPI_Comm comm=PETSC_COMM_WORLD;
```

- **create**: `VecCreate(comm, &v);`
- **layout**: `VecSetSizes(v,m,M);`
- **type**: `VecSetType(v,type);`

- **options:** `VecSetFromOptions(v);`
- **dealloc**: `VecDestroy(&v);`

Vector (Vec)

sequential alternative

```
Vec v;  PetscInt m=2, M=8;
VecType type=VECMPI;                  // VECSEQ,VECSTANDARD
MPI_Comm comm=PETSC_COMM_WORLD;    // PETSC_COMM_SELF
```

- **create**: `VecCreate(comm, &v);`
- **layout**: `VecSetSizes(v,m,M);` `// VecSetSizes(v,M,M);`
- **type**: `VecSetType(v,type);`

- **options:** `VecSetFromOptions(v);`
- **dealloc**: `VecDestroy(&v);`

Vector (Vec)

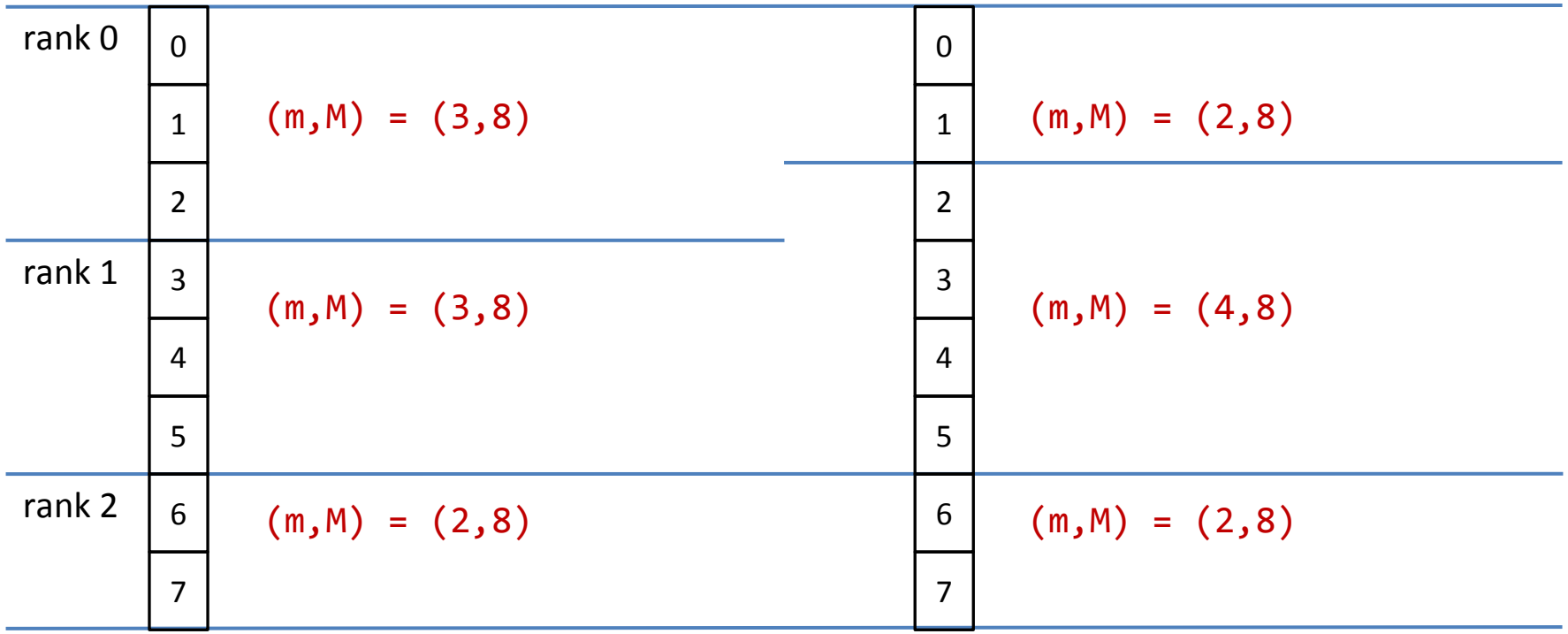
sequential alternative

```
Vec v;  PetscInt m=2, M=8;
VecType type=VECMPI;                  // VECSEQ,VECSTANDARD
MPI_Comm comm=PETSC_COMM_WORLD;       // PETSC_COMM_SELF
```

- **create**: `VecCreate(comm, &v);`
- **layout**: `VecSetSizes(v,m,M);` `// VecSetSizes(v,M,M);`
- **type**: `VecSetType(v,type);`
- **all in one**: `VecCreateMPI(comm,m,M,&v);//VecCreateSeq(comm,M,&v);`
- **options:** `VecSetFromOptions(v);`
- **dealloc**: `VecDestroy(&v);`

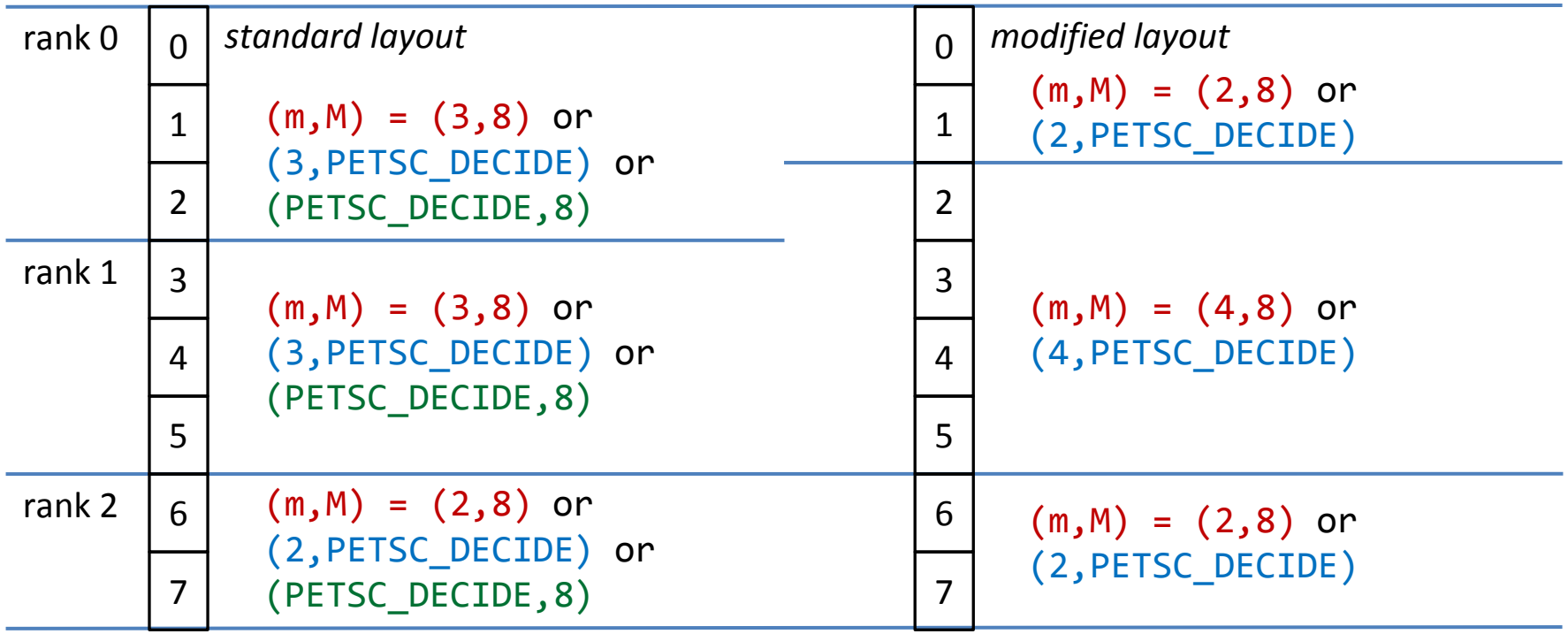
Parallel layout (1)

- consider the vector v with local size `m`, global size `M`, distributed across 3 processes
- call `VecSetSizes(v,m,M)` to set the sizes



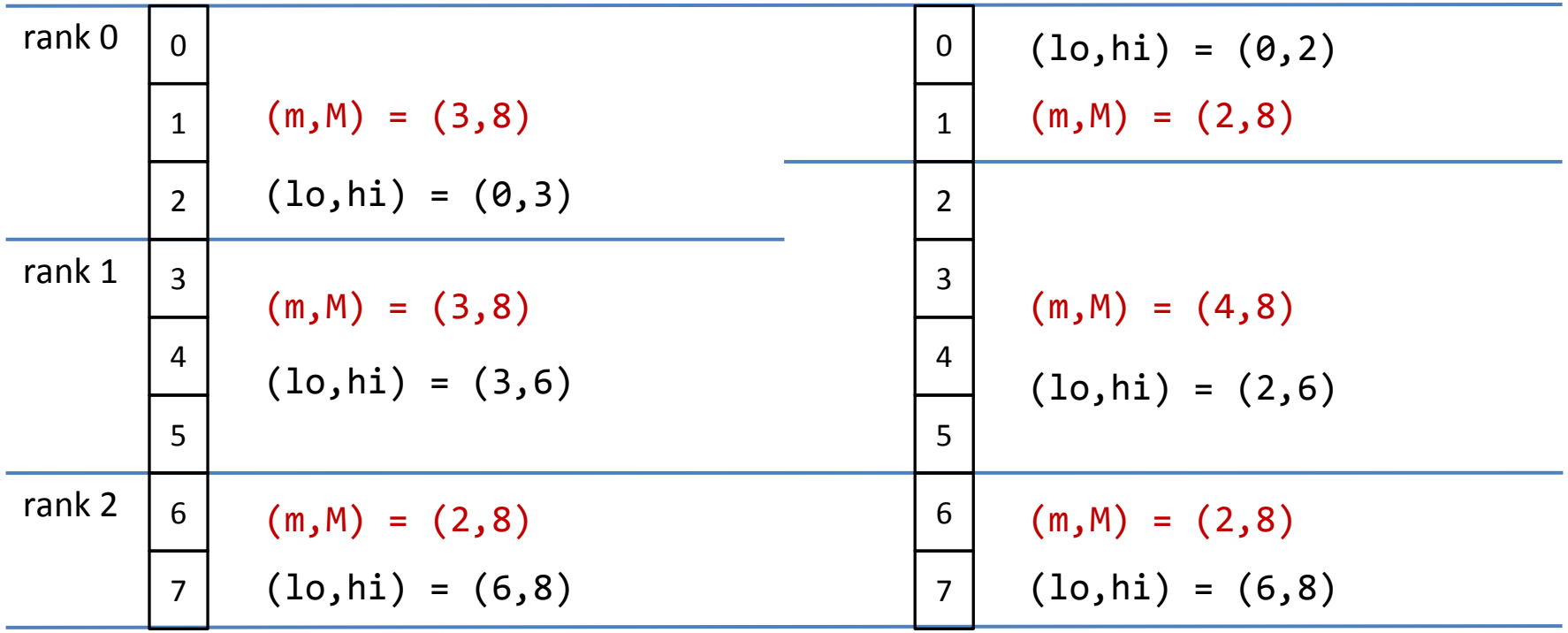
Parallel layout (2)

- set either `m` or `M` to `PETSC_DECIDE` (enforcing **standard layout**)
- get this standard layout across `comm`: `PetscSplitOwnership(comm,&m,&M)`



Querying layout

- **local** and **global sizes:** `VecGetLocalSize(v,&m)` and `VecGetSize(v,&M)`
- **global indices** of the **first** and **last elements** of the **local portion**: `VecGetOwnershipRange(v,&lo,&hi)`



Vector assembly (1)

```
Vec x;
```

- Set **all** entries to **constant** value:
```
VecSet(x, 1.0);
```
- Set **all** entries to **0**:
```
VecZeroEntries(x);
```
- Set an **individual** element (global indexing !):
```
PetscInt i = 10;  PetscReal v = 3.14;
VecSetValue(x, i, v, INSERT_VALUES);
// or
VecSetValues(x, 1, &i, &v, INSERT_VALUES);
```

Vector assembly (2)

- Set **multiple entries at once**:

```
PetscInt ii[]={1, 2}; PetscReal vv[]={2.7, 3.1);
VecSetValues(x, 2, ii, vv, INSERT_VALUES);
```

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

Vector assembly (3)

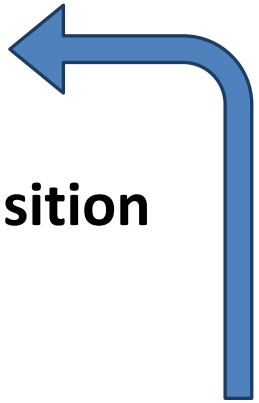
- `VecSetValues` is **purely local** with **no** inter-process **communication**
- values are just locally **cached**
- before using the vector,
  call **assembly function pair** to **exchange** values **between processors**:

```
VecAssemblyBegin(x);
VecAssemblyEnd(x);
```

Vector assembly (3)

- `VecSetValues` 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**:

```
VecAssemblyBegin(x);
//optional calculations not involving x
VecAssemblyEnd(x);
```



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

Getting values

```
Vec x;
```

- get a **copy** of 2 **local entries** of x with **global indices** ix to an array y

```
PetscInt ix[]={10, 20};
PetscScalar v[2];
VecGetValues(x, 2, ix, v);
```

- get the **pointer** to the whole **local internal array**

```
PetscScalar *a;
VecGetArray(x, &a);
// read and/or modify the array a
VecRestoreArray(x, &a);
```

Duplicate, copy, initialize

- **Create** another Vec with the **same type & layout**

```
Vec v, w;
VecDuplicate(v,&w);      // w contains undefined values
```

- **Initialize** values of Vec

```
VecZeroEntries(v);       // zeros
VecSet(v,1.0);           // ones
VecSetRandom(v,NULL);    // pseudo-random values

PetscRandom r;
PetscRandomCreate(PETSC_COMM_WORLD,&r);
VecSetRandom(x,r);
PetscRandomDestroy(&r);  // set pseudo-random values using explicit seed
```

- **Copy** the entries from v to w

```
VecCopy(v,w);            // v = w, w must be allocated
```

Some more vector functions

VecAXPY(Vec y,PetscScalar a,Vec x);	$y = y + a * x$
VecAYPX(Vec y,PetscScalar a,Vec x);	$y = x + a * y$
VecWAXPY(Vec w,PetscScalar a,Vec x,Vec y);	$w = a * x + y$
VecAXPBY(Vec y,PetscScalar a,PetscScalar b,Vec x);	$y = a * x + b * y$
VecScale(Vec x, PetscScalar a);	$x = a * x$
VecDot(Vec x, Vec y, PetscScalar *r);	$r = \bar{x}' * y$
VecTDot(Vec x, Vec y, PetscScalar *r);	$r = x' * y$
VecNorm(Vec x,NormType type, PetscReal *r);	$r = \|\|x\|\|_{type}$
VecSum(Vec x, PetscScalar *r);	$r = \sum x_i$
VecCopy(Vec x, Vec y);	$y = x$
VecSwap(Vec x, Vec y);	$y = x$ while $x = y$
VecPointwiseMult(Vec w,Vec x,Vec y);	$w_i = x_i * y_i$
VecPointwiseDivide(Vec w,Vec x,Vec y);	$w_i = x_i / y_i$
VecMDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = \bar{x}' * y[i]$
VecMTDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = x' * y[i]$
VecMAXPY(Vec y,int n, PetscScalar *a, Vec x[]);	$y = y + \sum_i a_i * x[i]$
VecMax(Vec x, int *idx, PetscReal *r);	$r = \max x_i$
VecMin(Vec x, int *idx, PetscReal *r);	$r = \min x_i$
VecAbs(Vec x);	$x_i = \|x_i\|$
VecReciprocal(Vec x);	$x_i = 1/x_i$
VecShift(Vec x,PetscScalar s);	$x_i = s + x_i$
VecSet(Vec x,PetscScalar alpha);	$x_i = \alpha$

Index Set (IS)

- `PetscObject` used for integer vectors (not necessarily ”index sets”)
- distributed in the same manner as `Vec`
  - local and global size, ownership range
- `ISCreateStride(comm, n, first, step, &is);`
  produces `first:step:first+n-1` stride
- `ISCreateGeneral(comm, n, idx, mode, &is);`
  wraps integer array `PetscInt idx[]` into `IS`
  - `PetcsCopyMode mode` is one of `PETSC_COPY_VALUES`, `PETSC_OWN_POINTER`, `PETSC_USE_POINTER`, specifying handling of the input array
- useful functions:
  `ISConcatenate, ISDifference, ISExpand, ISMinMax, ISSort, …`

Hands-on: ex3.c

1. `cd ~/petsc/exercises`
2. `make ex3 && mpirun -n 3 ./ex3  // what's wrong?`
3. set local size to `rank+1`, let global size be computed by PETSc (`PETSC_DECIDE`)
4. `make ex3 && mpirun -n 3 ./ex3  // what's wrong?`
5. fix using `VecAssemblyBegin/End`
6. `make ex3 && mpirun -n 3 ./ex3`
7. look at the output of `VecView`
8. all values of vector `x` are equal to `commsize+1`, why?
9. get the ownership range into `lo` and `hi` variables
10. set each process' values of `x` to the value `rank+1` (e.g. rank 1 sets all its local values of `x` to 3) just by changing the lower limit of the original loop to `lo`
11. do the same, avoiding communication (loop from `lo` to `hi`, use `INSERT_VALUES` and `rank`)

Thanks for your attention.