Introduction to PETSc

Debugging, Profiling
Parallel Data Layout

Hong Zhang

2010

PETSc Programming Aids

- Correctness Debugging
  - Automatic generation of tracebacks
  - Detecting memory corruption and leaks
  - Optional user-defined error handlers
- Performance Debugging
  - Integrated profiling using -log_summary
  - Profiling by stages of an application
  - User-defined events

Integration

Debugging

Support for parallel debugging

- -start_in_debugger [gdb,dbx,noxterm]
- -on_error_attach_debugger [gdb,dbx,noxterm]
- -on_error_abort
- -debugger_nodes 0,1
- -display machinename:0.0

When debugging, it is often useful to place a breakpoint in the function PetscError( ).

Sample Error Traceback

Breakdown in ILU factorization due to a zero pivot

```
Buffers Files Tools Edit Search Mule Help
[dreamcast] mpirun -np 1 ex1
--------------------------------------------------------------------------
PETSc Version 2.1.0, Released April 11, 2001
       The PETSc Team    petsc-maint@mcs.anl.gov
 http://www.mcs.anl.gov/petsc/

See docs/copyright.html for copyright information.
See docs/changes.html for recent updates.
See docs/troubleshooting.html for hints about trouble shooting.
See docs/manualpages/index.html for manual pages.
-----------------------------------------------------------------------
ex1 on a linux named dreamcast.mcs.anl.gov by balay Thu Oct  4 15:25:11 2001
Libraries linked from /home/balay/software/petsc-2.1.0/lib/libg/linux
-----------------------------------------------------------------------
[0]PETSC ERROR: MatLUFactorNumeric_SeqAIJ() line 508 in src/mat/impls/aij/seq/aijfact.c
[0]PETSC ERROR:   Detected zero pivot in LU factorization!
[0]PETSC ERROR:   Zero pivot row 0!
[0]PETSC ERROR: MatLUFactorNumeric() line 1575 in src/mat/interface/matrix.c
[0]PETSC ERROR: PCSetUp_ILU() line 646 in src/sles/pc/impls/ilu/ilu.c
[0]PETSC ERROR: PCSetUp() line 783 in src/sles/pc/interface/precon.c
[0]PETSC ERROR: SLESSetUp() line 382 in src/sles/interface/sles.c
[0]PETSC ERROR: SLESSolve() line 483 in src/sles/interface/sles.c
[0]PETSC ERROR: main() line 195 in test/ex1.c
[0] MPI Abort by user Aborting program !
[0] Aborting program!
p0_5469:  p4_error: : 71
--1-:---F1  logs              (Text)--L3-- 2%-----------------------------------
```

debugging and errors

Profiling and Performance Tuning

Profiling:

- Integrated profiling using -log_summary
- User-defined events
- Profiling by stages of an application

Performance Tuning:

- Matrix optimizations
- Application optimizations
- Algorithmic tuning

profiling and
performance tuning

Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication
- Active if PETSc was configured with

  --with-debugging=1 (default)
  - Can also profile application code segments
- Print summary data with option: -log_summary
- Print redundant information from PETSc routines: -info [infofile]
- Print the trace of the functions called: -log_trace [logfile]

profiling and
performance tuning

Sample -log_summary

```
------------------------------------------------------------------------------------------------------------------------
Event                Count      Time (sec)     Flops/sec                         --- Global ---  --- Stage ---   Total
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   Avg len Reduct  %T %F %M %L %R  %T %F %M %L %R Mflop/s
------------------------------------------------------------------------------------------------------------------------

--- Event Stage 0: Main Stage

PetscBarrier           2 1.0 1.1733e-05 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0

--- Event Stage 1: SetUp

VecSet                 2 1.0 9.3448e-04 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatMultTranspose       1 1.0 1.8022e-03 1.0 1.85e+08 1.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0 57  0  0  0   185
MatAssemblyBegin       3 1.0 1.0057e-05 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatAssemblyEnd         3 1.0 2.0356e-02 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   5  0  0  0  0     0
MatFDColorCreate       2 1.0 1.5341e-01 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 4.6e+01  1  0  0  0 16  36  0  0  0 74     0

--- Event Stage 2: Solve

VecDot                 2 1.0 3.2985e-03 1.0 9.56e+07 1.0 0.0e+00 0.0e+00 2.0e+00  0  0  0  0  1   0  0  0  0  2    96
VecMDot               45 1.0 9.3093e-02 1.0 1.59e+08 1.0 0.0e+00 0.0e+00 1.5e+01  0  0  0  0  5   1  1  0  0 19   159
VecNorm              112 1.0 2.0851e-01 1.0 8.47e+07 1.0 0.0e+00 0.0e+00 5.2e+01  1  1  0  0 18   2  1  0  0 64    85
```

```
MatMultTranspose   1 1.0 1.8022e-03 1.0 1.85e+08 ...
VecNorm          112 1.0 2.0851e-01 1.0 8.47e+07 ... 5.2e+01 …
```

Adding a new Stage

```
int stageNum;
PetscLogStageRegister(&stageNum,"name");

PetscLogStagePush(stageNum);
[code to monitor]
PetscLogStagePop();
```

profiling and
performance tuning

Adding a new Event

```
static int USER_EVENT;
PetscLogEventRegister
  (&USER_EVENT,"name",CLASS_COOKIE);

PetscLogEventBegin(USER_EVENT,0,0,0,0);
[code to monitor]
PetscLogFlops(user_evnet_flops);
PetscLogEventEnd(USER_EVENT,0,0,0,0);
```

Parallel Data Layout and Ghost Values: Usage Concepts

Managing field data layout and required ghost values is the key to high performance of most PDE-based parallel programs.

Mesh Types

- Structured
  - DA objects
- Unstructured
  - VecScatter objects

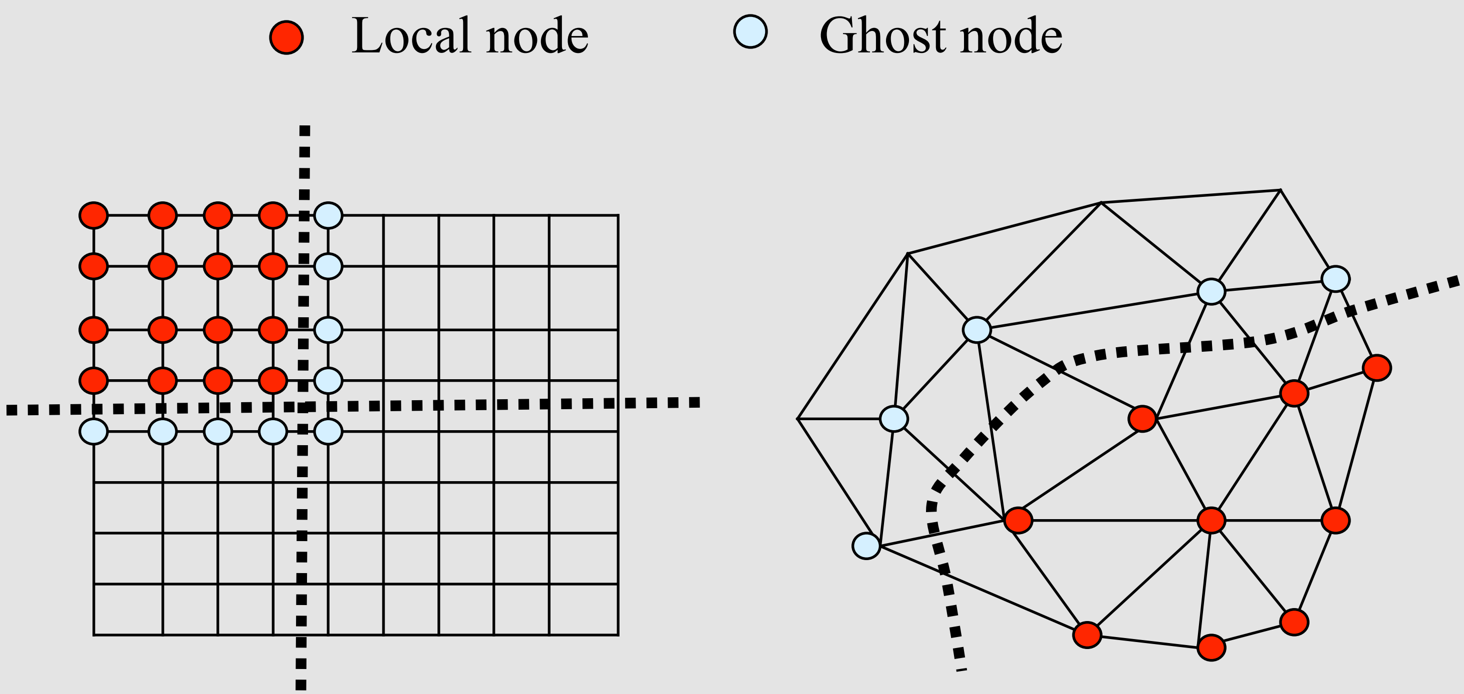
Usage Concepts

- Geometric data
- Data structure creation
- Ghost point updates
- Local numerical computation

important concepts

data layout

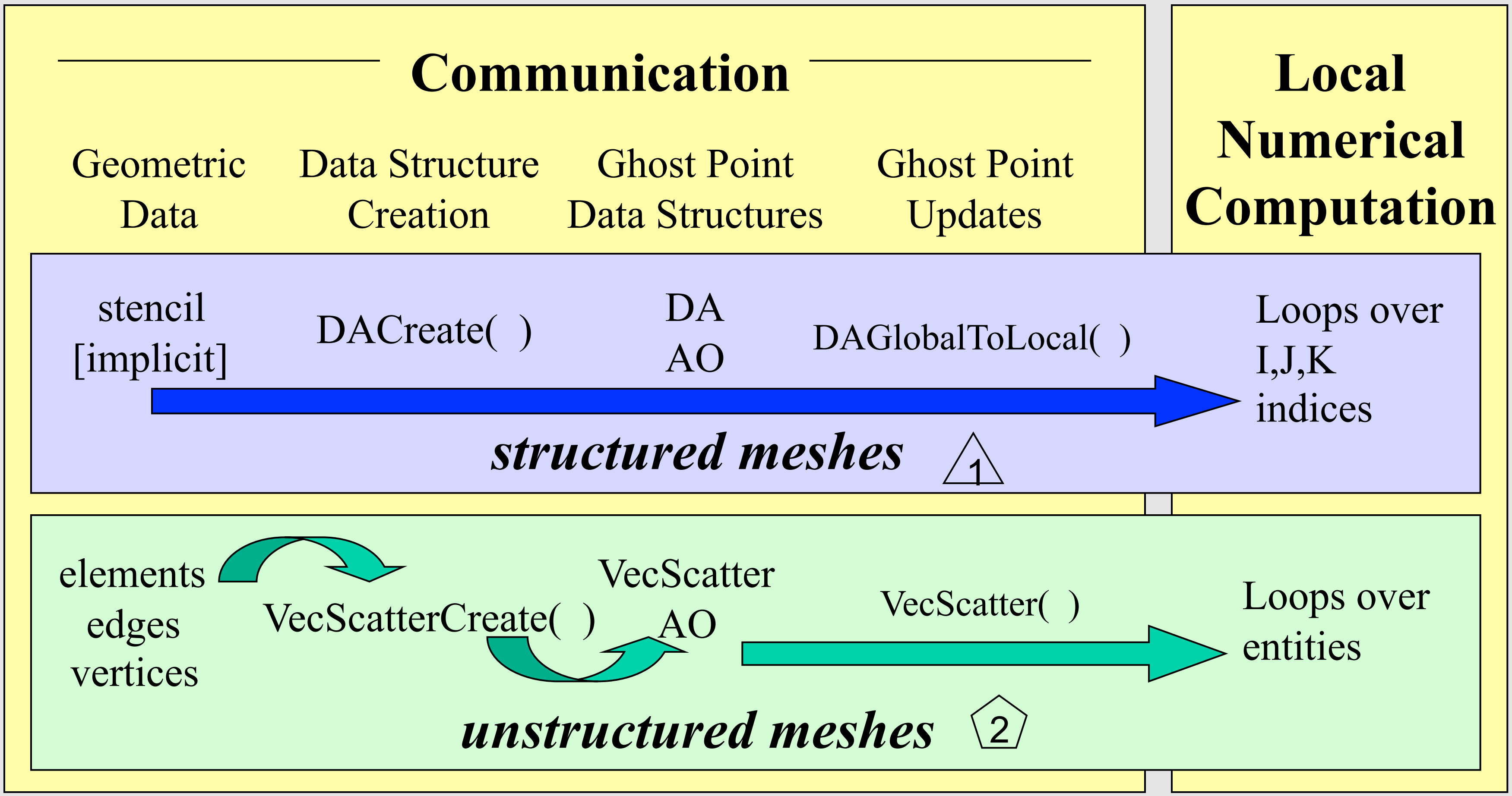
Ghost Values



Ghost values: To evaluate a local function *f(x)* , each process requires its local portion of the vector *x* as well as its **ghost values** – or bordering portions of *x* that are owned by neighboring processes.

data layout

Communication and Physical Discretization



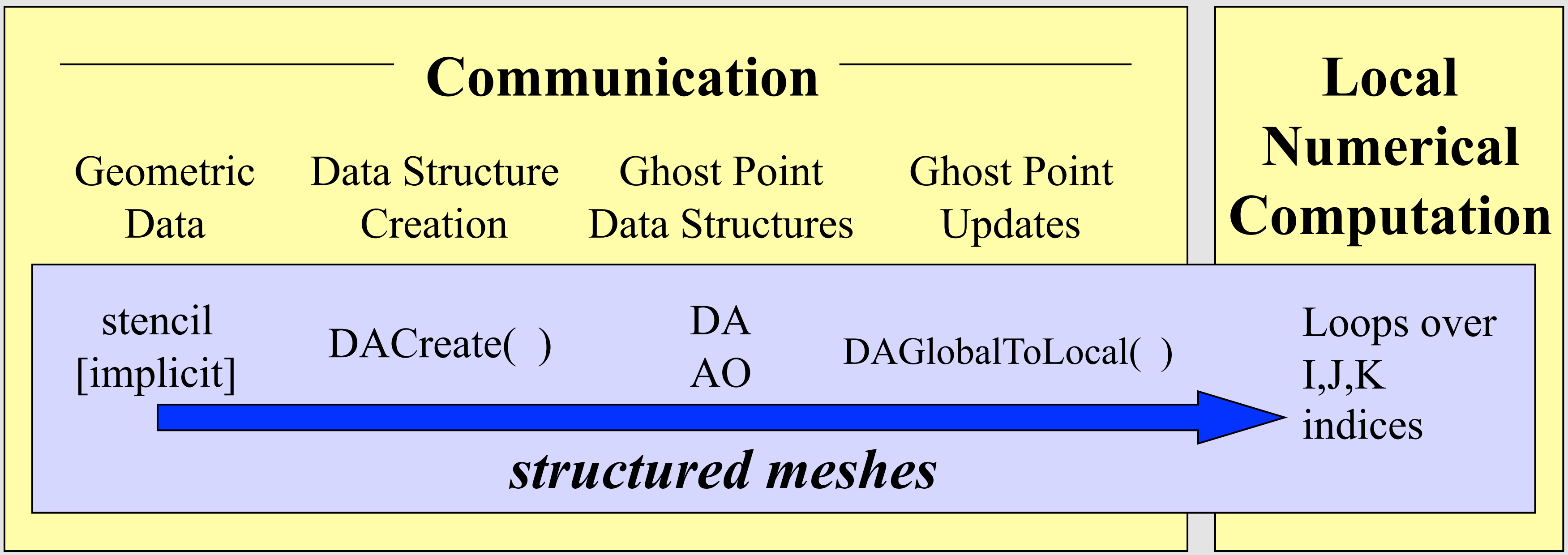
data layout

DA: Parallel Data Layout and Ghost Values for Structured Meshes

- Local and global indices
- Local and global vectors
- DA creation
- Ghost point updates
- Viewing

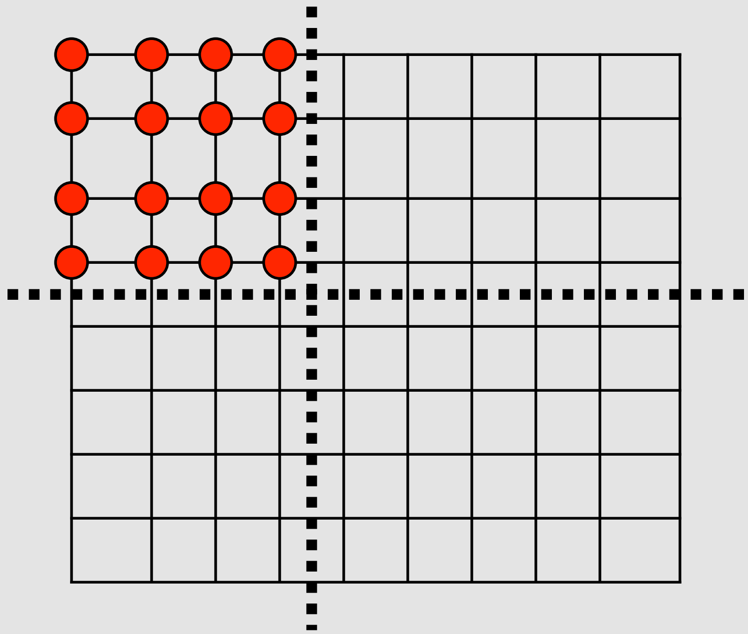
data layout:
distributed arrays

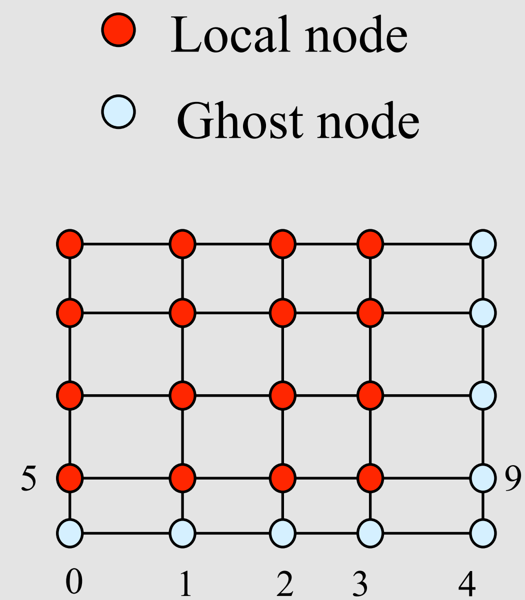
Communication and Physical Discretization: Structured Meshes



data layout:
distributed arrays

Global and Local Representations



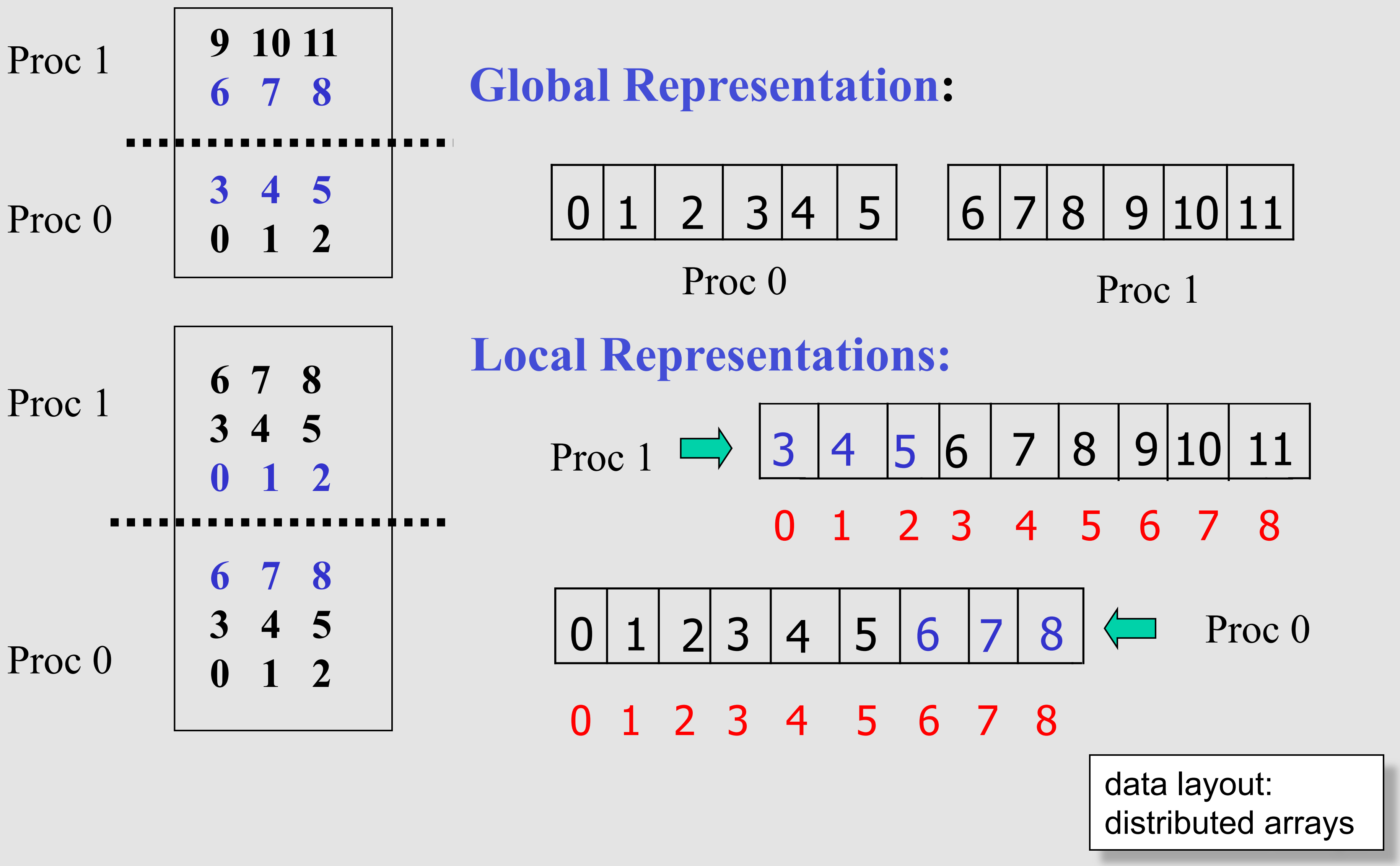


Global: each process stores a unique local set of vertices (and each vertex is owned by exactly one process)

Local: each process stores a unique local set of vertices *as well as* ghost nodes from neighboring processes

data layout:
distributed arrays

Global and Local Representations (cont.)



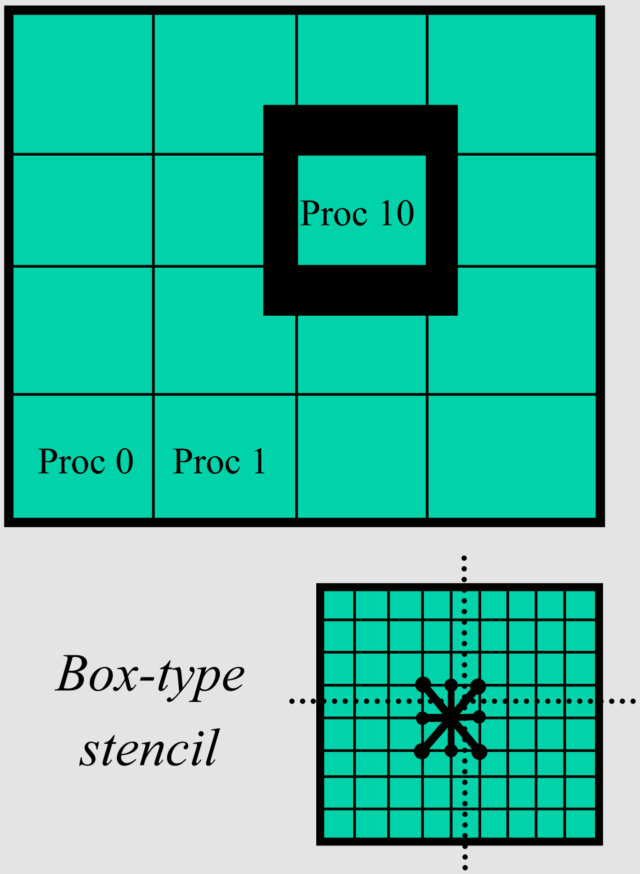
Logically Regular Meshes

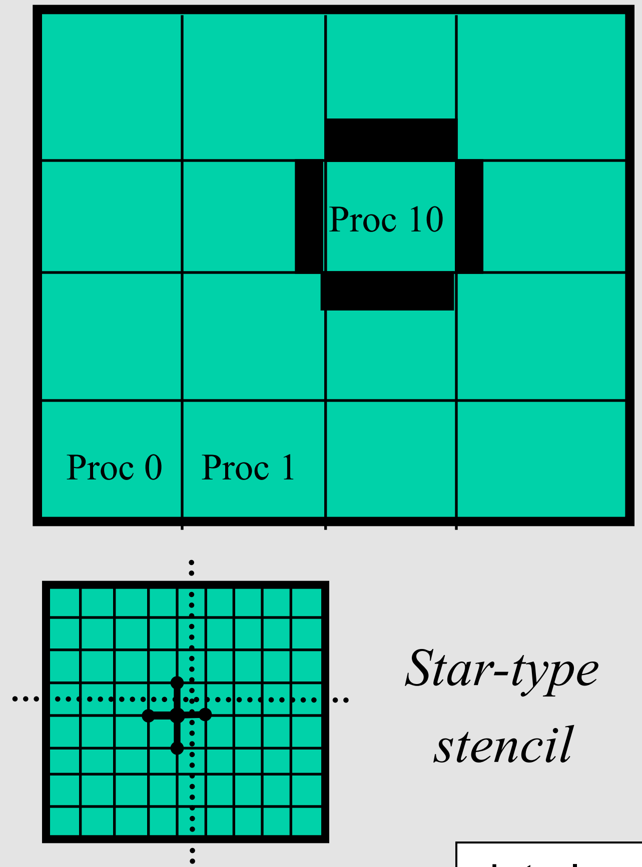
- DA - Distributed Array: object containing information about vector layout across the processes and communication of ghost values
- Form a DA
  - DACreate1d(….,DA *)
  - DACreate2d(….,DA *)
  - DACreate3d(….,DA *)
- Create the corresponding PETSc vectors
  - DACreateGlobalVector(DA, Vec *) or
  - DACreateLocalVector(DA, Vec *)
- Update ghostpoints (scatter global vector into local parts, including ghost points)
  - DAGlobalToLocalBegin(DA, …)
  - DAGlobalToLocalEnd(DA,…)

data layout:
distributed arrays

Distributed Arrays (DA)

Data layout and ghost values





data layout:
distributed arrays

Vectors and DAs

- The DA object contains information about the data layout and ghost values, but **not** the actual field data, which is contained in PETSc vectors
- Global vector: parallel
  - each process stores a unique local portion
  - DACreateGlobalVector(DA da,Vec *gvec);
- Local work vector: sequential
  - each process stores its local portion plus ghost values
  - DACreateLocalVector(DA da,Vec *lvec);
  - uses "natural" local numbering of indices (0,1,…*nlocal*-1)

data layout:
distributed arrays

DACreate1d(…,*DA)

- DACreate1d(MPI_Comm comm,DAPeriodicType wrap,
  int M,int dof,int s,int *lc,DA *inra)
  - MPI_Comm — processes containing array
  - DA_[NONPERIODIC,XPERIODIC]
  - number of grid points in x-direction
  - degrees of freedom per node
  - stencil width
  - Number of nodes for each domain
    - Use PETSC_NULL for the default

data layout:
distributed arrays

DACreate2d(…,*DA)

- DACreate2d(MPI_Comm comm,DAPeriodicType wrap,
  DAStencilType stencil_type, int M,int N,
  int m,int n,int dof,int s,int *lx,int *ly,DA *inra)
  - DA_[NON,X,Y,XY]PERIODIC
  - DA_STENCIL_[STAR,BOX]
  - number of grid points in x- and y-directions
  - processes in x- and y-directions
  - degrees of freedom per node
  - stencil width
  - Number of nodes for each domain
    - Use PETSC_NULL for the default

data layout: distributed arrays

Updating the Local Representation

Two-step process enables overlapping computation and communication

- DAGlobalToLocalBegin(DA, global_vec, insert,local_vec)
  - global_vec provides data
  - Insert is either INSERT_VALUES or ADD_VALUES
    - specifies how to update values in the local vector
  - local_vec is a pre-existing local vector
- DAGlobalToLocalEnd(DA,…)
  - Takes same arguments

data layout:
distributed arrays

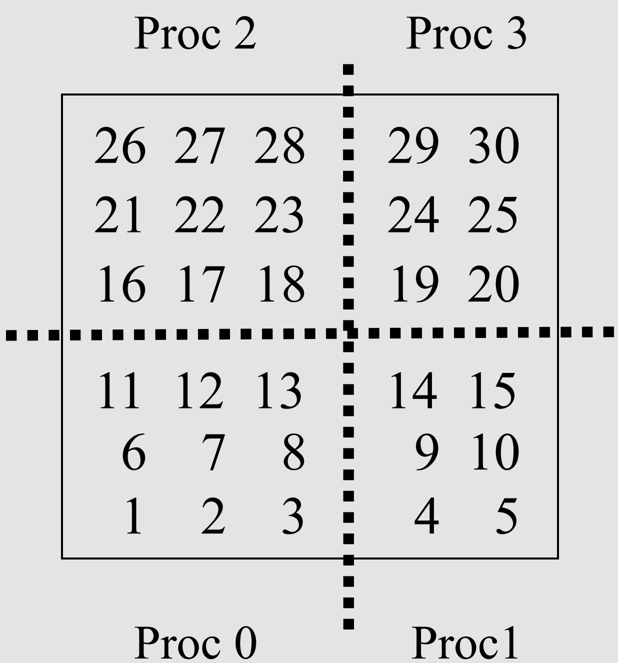
Ghost Point Scatters:

Burger's Equation Example

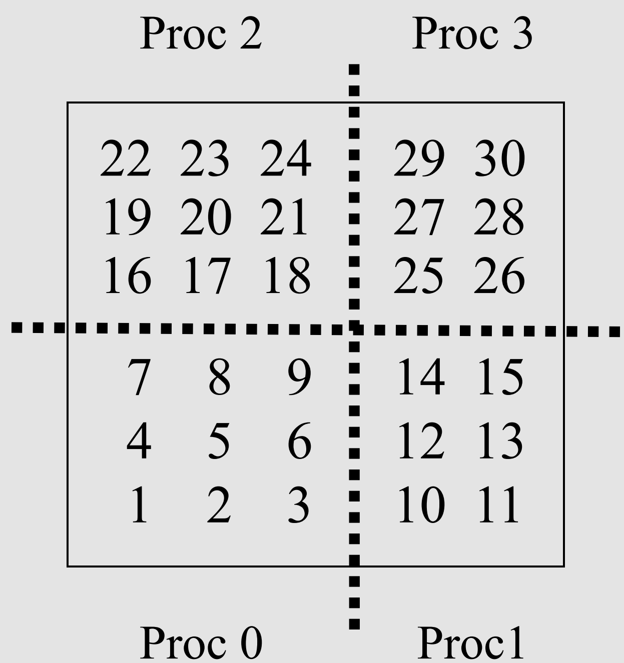
```
      call DAGlobalToLocalBegin(da,u_global,INSERT_VALUES,ulocal,ierr)
      call DAGlobalToLocalEnd(da,u_global,INSERT_VALUES,ulocal,ierr)

      call VecGetArray( ulocal, uv, ui, ierr )
#define u(i) uv(ui+i)
C  Do local computations (here u and f are local vectors)
      do 10, i=1,localsize
         f(i) = (.5/h)*u(i)*(u(i+1) - u(i-1)) + (e/(h*h))*(u(i+1) - 2.0*u(i) + u(i-1))
10  continue
      call VecRestoreArray( ulocal, uv, ui, ierr )
      call DALocalToGlobal(da,f,INSERT_VALUES,f_global,ierr)
```

data layout:
distributed arrays

Global Numbering used by DAs



Natural numbering, corresponding to the entire problem domain



PETSc numbering used by DAs

data layout:
distributed arrays

Mapping Between Global Numberings

- Natural global numbering
  - convenient for visualization of global problem, specification of certain boundary conditions, etc.
- Can convert between various global numbering schemes using AO (Application Orderings)
  - DAGetAO(DA da, AO *ao);
  - AO usage explained in next section
- Some utilities (e.g., VecView()) automatically handle this mapping for global vectors attained from DAs

Distributed Array Example

- Files: src/snes/examples/tutorial/ex5.c, ex5f.F
  – Functions that construct vectors and matrices use a naturally associated DA
    - DAGetMatrix()
    - DASetLocalFunction()
    - DASetLocalJacobian()

data layout:
distributed arrays

The Bratu Equation I
SNES Example 5

- Create SNES and DA
- Use DASetLocalFunction()
  - Similarly DASetLocalJacobian()
- Use SNESDAFormFunction() for SNES
  - Could also use FormFunctionMatlab()
- Similarly SNESDAComputeJacobian()
  - Use DAGetMatrix() for SNES Jacobian
  - Could also use SNESDefaultComputeJacobian()

The Bratu Equation II
SNES Example 5

- int FormFunctionLocal(DALocalInfo *info,
  PetscScalar **x, PetscScalar **f,
  void *ctx)

```
for(j = info->ys; j < info->ys + info->ym; j++) {
  for(i = info->xs; i < info->xs + info->xm; i++) {
    if (i == 0 || j == 0 || i == info->mx-1 || j == info->my-1) {
      f[j][i] = x[j][i];
    } else {
      u       = x[j][i];
      u_xx    = -(x[j][i+1] - 2.0*u + x[j][i-1])*(hy/hx);
      u_yy    = -(x[j+1][i] - 2.0*u + x[j-1][i])*(hx/hy);
      f[j][i] = u_xx + u_yy - hx*hy*lambda*PetscExpScalar(u);
    }
  }
}
```

The Bratu Equation III
SNES Example 5

- int FormJacobianLocal(DALocalInfo *info, PetscScalar **x, Mat jac, void *ctx)

```
for(j = info->ys; j < info->ys + info->ym; j++) {
  for(i = info->xs; i < info->xs + info->xm; i++) {
    row.j = j; row.i = i;
    if (i == 0 || j == 0 || i == info->mx-1 || j == info->my-1) {
      v[0] = 1.0;
      MatSetValuesStencil(jac,1,&row,1,&row,v,INSERT_VALUES);
    } else {
      v[0] = -(hx/hy); col[0].j = j-1; col[0].i = i;
      v[1] = -(hy/hx); col[1].j = j;   col[1].i = i-1;
      v[2] = 2.0*(hy/hx+hx/hy) - hx*hy*lambda*PetscExpScalar(x[j][i]);
      v[3] = -(hy/hx); col[3].j = j;   col[3].i = i+1;
      v[4] = -(hx/hy); col[4].j = j+1; col[4].i = i;
      MatSetValuesStencil(jac,1,&row,5,col,v,INSERT_VALUES);
    }
  }
}
```