Intermediate MPI: A Practical Approach (Day 3)

Dr. David Ennis

Ohio Supercomputer Center

and

Dr. Kadin Tseng (kadin@bu.edu)

http://scv.bu.edu/~kadin/intermediate-mpi/

Boston University

http://webct.ncsa.uiuc.edu:8900/

MPI Reduction Operations

MPI Reduction Operations:

MPI_Reduce, MPI_Allreduce, MPI_Scan

User must specify a reduction operator when the above reduction operation functions are called. There are two types of reduction operators:

- MPI-defined operators

  MPI_SUM, MPI_MAX, MPI_MIN, …

- User-defined operators

  Subject of this section. But first, a quick review of reduction operations …

Review – Predefined Reduction Operations

Example: Compute $s = \sum_{i=0}^{p-1} i$

Fortran:

```
call MPI_Comm_rank(MPI_COMM_WORLD, i, ierr)

call MPI_Reduce(i, s, 1, MPI_INTEGER, MPI_SUM, dest,  &
                    MPI_COMM_WORLD, ierr)
```

C:

```
MPI_Comm_rank(MPI_COMM_WORLD, &i);

MPI_Reduce(i, &s, 1, MPI_INT, MPI_SUM, dest,
                    MPI_COMM_WORLD);
```

User-defined Reduction Operations

There are 3 steps to the creation of a user-defined reduction operation . . .

User-defined Reduction … Step 1

Step 1. The operation, say □, must satisfy the following rules:

- □ must satisfy associative rule   *a □ (b □ c) = (a □ b) □ c*
  - Addition, “+”, satisfies associative rule.
  - Subtraction, “-”, does not.

- □ *optionally* satisfies the commutative rule   *a □ b = b □ a*
  - Multiplication, “*”, satisfies both associative and commutative rules.
  - Division, “/”, satisfies neither.

http://webct.ncsa.uiuc.edu:8900/

User-defined Reduction … Step 2

Step 2. Implement operator into a reduction function following rules:

Fortran:

```
FUNCTION MYFUNC(IN, INOUT, LEN, DATATYPE)
INTEGER LEN, DATATYPE
<type> IN(LEN), INOUT(LEN)
```

<type> is one of REAL, INTEGER, COMPLEX, …

DATATYPE is defined by the MPI data type declared in the MPI reduction function call. It should be consistent with <type>.

C:

```
function myfunc(void *in, void *inout, int *len, MPI_Datatype *datatype)
```

User-defined Reduction … (cont'd)

Step 3. Register MYFUNC with MPI (*Fortran*)   and

declare EXTERNAL MYFUNC

```
    EXTERNAL MYFUNC        ! Declare MYFUNC an external function
    INTEGER MYOP           ! MPI handle for MYFUNC
    LOGICAL COMMUTE
        . . .
    COMMUTE = .TRUE.       ! If operator is commutative; else .FALSE.
! Registers MYFUNC with MPI to obtain operator handle MYOP
    CALL MPI_OP_CREATE(MYFUNC, COMMUTE, MYOP, IERR)
    CALL MPI_REDUCE( …, MYOP, …)    ! Use MYOP, not MYFUNC
```

User-defined Reduction … (cont'd)

Step 3. Register myfunc with MPI (*C*)

```
/* Unlike fortran coding, no external declaration for myfunc necessary */
    int commute, myop;
    commute = 1;         /* if operator is commutative, else 0 */
/* registers myfunc with MPI to obtain operator handle myop */
    MPI_Op_create(myfunc, commute, &myop);
    MPI_Reduce( …, myop, …) ;   /* use myop, not myfunc */
```

Example 1.

A user-implementation of MPI_SUM (*Fortran*)

```
  FUNCTION MYFUNC(IN, INOUT, LEN, DATATYPE)
  INTEGER LEN, DATATYPE, IERR
  REAL  IN(LEN), INOUT(LEN)
  INCLUDE ‘MPIF.H’
  IF (DATATYPE .NE. MPI_REAL)
&           CALL MPI_ABORT(MPI_COMM_WORLD, 1, IERR)
  DO I=1,LEN
    INOUT(I) = INOUT(I) + IN(I)
  ENDDO
  END
```

Example 1. (cont'd)

A user-implementation of MPI_SUM (*C*)

```
function myfunc(void *in, void *inout, int *len, MPI_Datatype *datatype) {
  float  *in2, *inout2;
   #include <mpi.h>
   in2 = (float*) in;   inout2 = (float*) inout;
   if (*datatype != MPI_FLOAT) MPI_Abort(MPI_COMM_WORLD, 1);
     for (i=0; i<*len; i++) {
        *inout2 += *in2;    inout2++;    in2++;
     }
   }
   in = in2;    inout = inout2;
}
```

Example 2. One-norm

Various norms are often used to measure the convergence history of numerical solutions. One-norm is defined as

$$N_1(\vec{x}) = \sum_{j=0}^{p-1} | x_j |$$

- "+" is the reduction operator
- MPI_SUM could be used with MPI_Reduce to achieve the effect of one-norm
- Will implement one-norm to highlight the computing procedure of processes

Example 2. One-norm subroutine

A user-implementation of one-norm (*Fortran*)

```
FUNCTION ONENORM(IN, INOUT, LEN, DATATYPE)
INTEGER LEN, DATATYPE
REAL  IN(LEN), INOUT(LEN)

DO I=1,LEN
   INOUT(I) = ABS(INOUT(I)) + ABS(IN(I))
ENDDO
END
```

Example 2. One-norm calling program

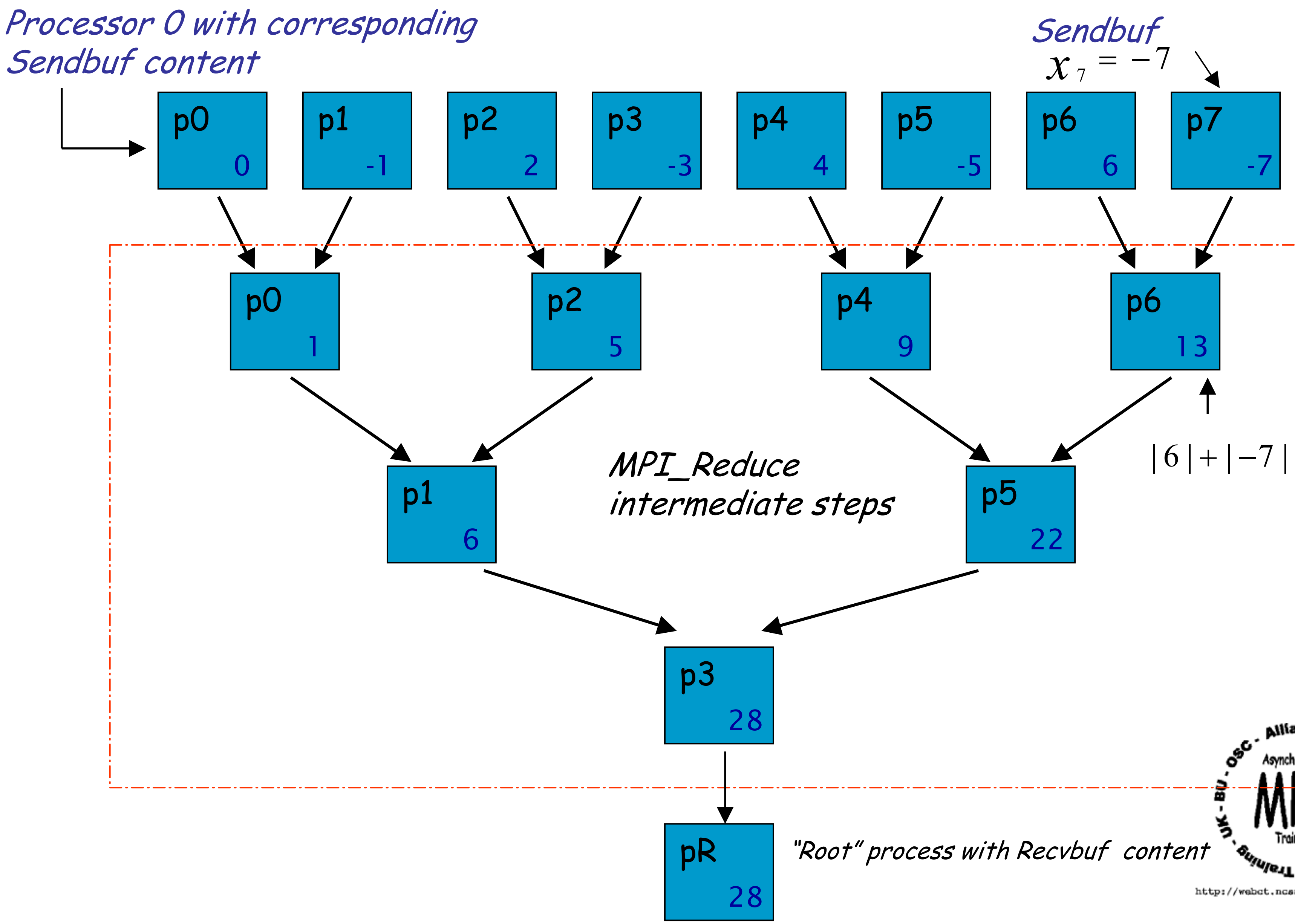
```
      EXTERNAL ONENORM          ! Declare ONENORM an external function
      INTEGER MYOP              ! MPI handle for ONENORM
      LOGICAL COMMUTE           ! Commutation allowed ?
        . . .
      COMMUTE = .TRUE.          ! operator is commutative
      CALL MPI_COMM_RANK(MPI_COMM_WORLD, J, IERR)
      XJ = J*(-1)**J            ! X = 0, -1, 2, -3, …
! Registers ONENORM with MPI to obtain operator handle MYOP
      CALL MPI_OP_CREATE(ONENORM, COMMUTE, MYOP, IERR)
      CALL MPI_REDUCE(XJ, N1, 1, MPI_REAL, MYOP, …)     ! N1 is one-norm
```

Alternatively,

```
      CALL MPI_REDUCE(ABS(XJ), N1, 1, MPI_REAL, MPI_SUM, …)
```

http://webct.ncsa.uiuc.edu:8900/

One Norm Example using 8 Processes



Processor 0 with corresponding Sendbuf content
Sendbuf
$x_7 = -7$
p0 0
p1 -1
p2 2
p3 -3
p4 4
p5 -5
p6 6
p7 -7
p0 1
p2 5
p4 9
p6 13
$|6|+|-7|$
p1 6
MPI_Reduce intermediate steps
p5 22
p3 28
pR 28
"Root" process with Recvbuf content

http://webct.ncsa.uiuc.edu:8900/

Virtual Topologies

Two different topologies available in MPI:

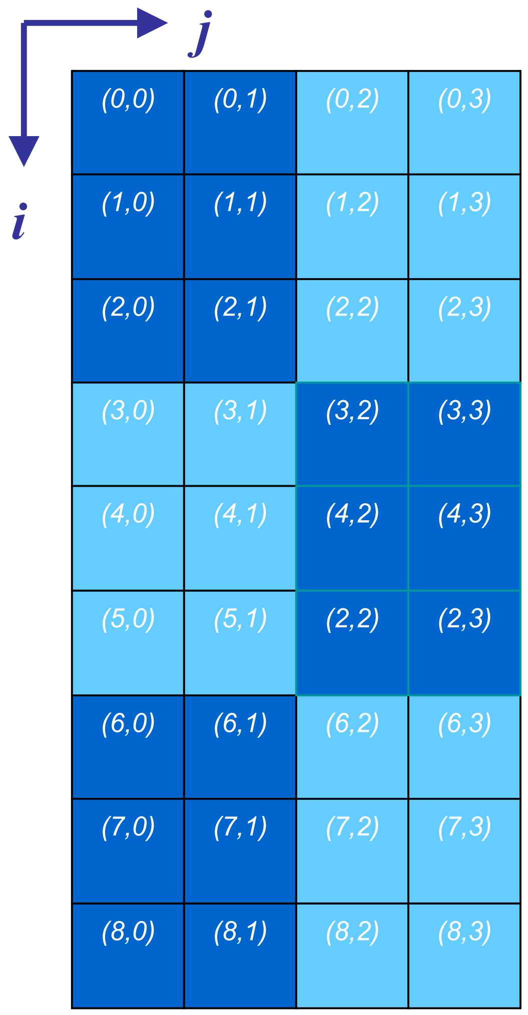
- Cartesian Topology
- Graph Topology

Virtual Topologies

First, a quick review of Cartesian Topology …

Will demonstrate usage of Cartesian Topology at the end.

Example: A 9 x 4 Array

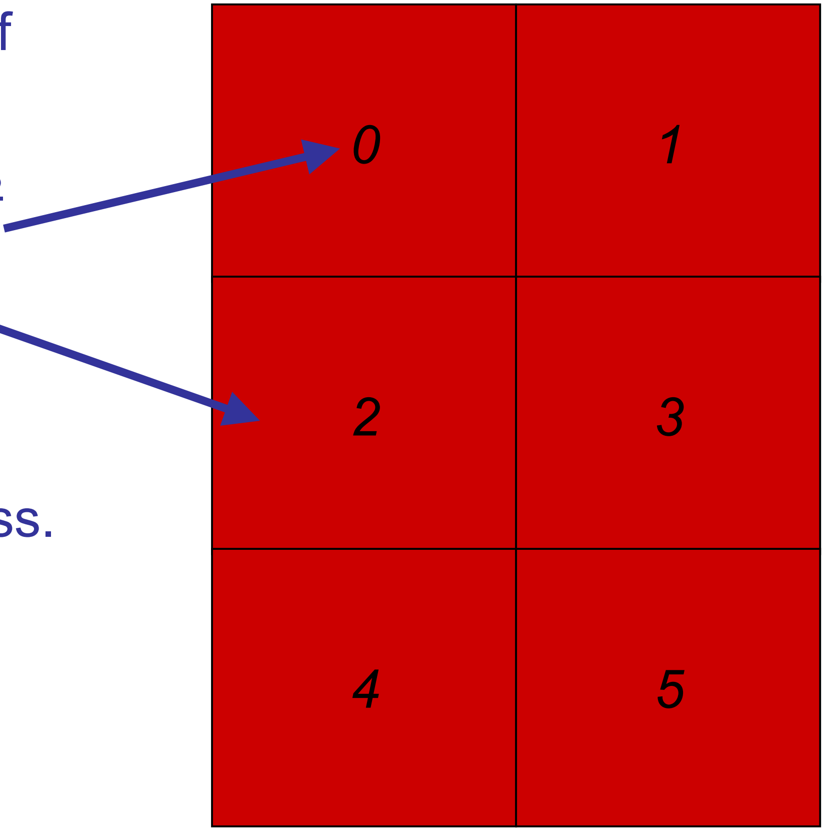


- Consider a *9x4* matrix.
- The parenthesized number-pairs, *(i, j)*, denote array row and column indexes, respectively.
- Assume six processes used for parallel computation.
- A 2D domain decomposition leads to six *3x2* submatrices as shown.

http://webct.ncsa.uiuc.edu:8900/

Domain Decomposition

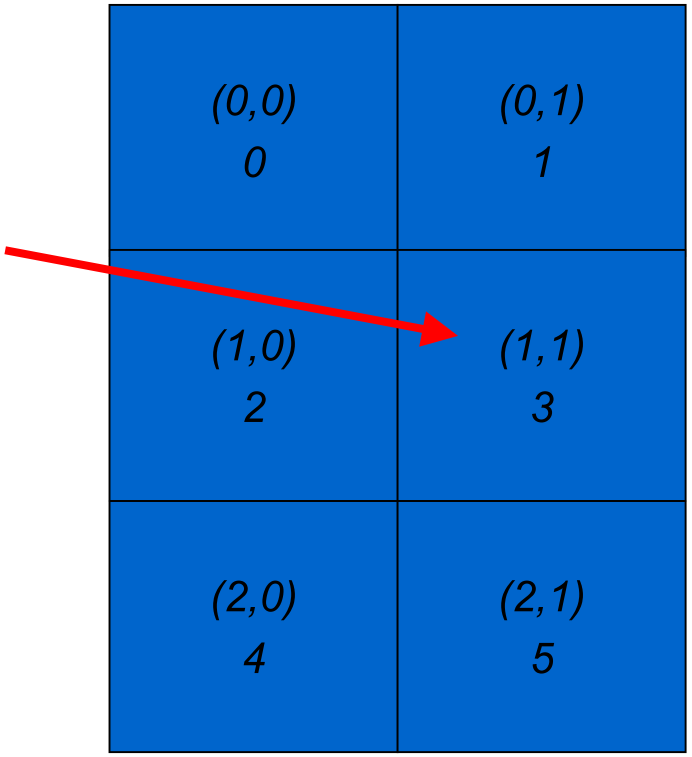
- Domain decomposition yields linear order representation of process topology.
- Number in each cell denotes process number.
- Each cell represents a *3x2* array out of the *9x4* array.
- Work within each cell is performed by a single process.



http://webct.ncsa.uiuc.edu:8900/

2D Cartesian Topology

- More convenient and intuitive to map linear rank order into a 2D Cartesian topology *(i,j)* via MPI function call.
- Example: linear rank *3* can be addressed by 2D Cartesian coordinates, *(1,1)*.
- Each cell represents a *3x2* matrix block whose work will be performed by the indicated process rank.
- MPI rank index starts from *0*.
- Ranks map into MPI Cartesian topology following row-major convention.

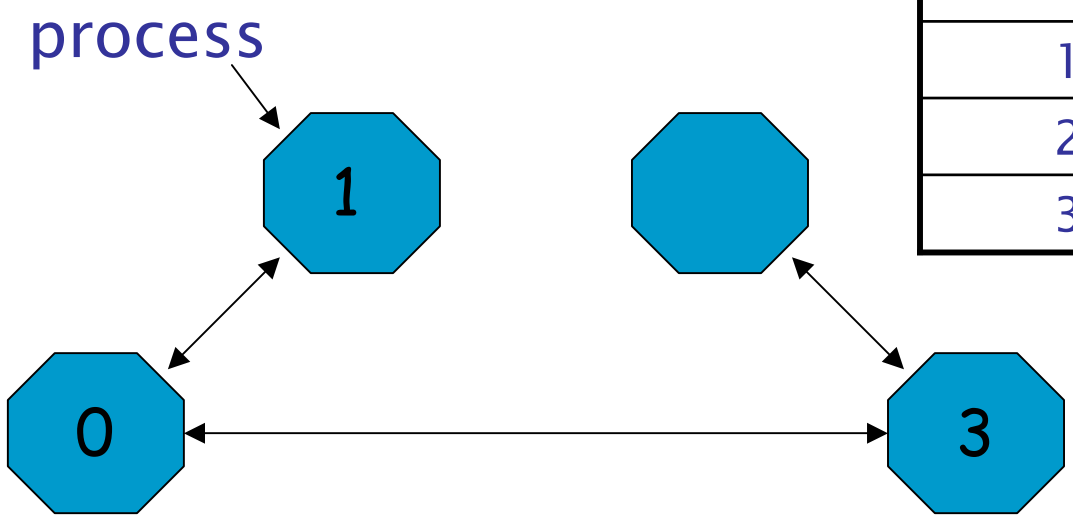


http://webct.ncsa.uiuc.edu:8900/

Graph Topology

- Graph Topology provides a mechanism for user to define arbitrary connections among processes
- Cartesian Topology maps linear ranks to Cartesian coordinate ranks

Graph Topology Essentials



Node	Nneighbors	index	edges
0	2	2	1,3
1	1	3	0
2	1	4	3
3	2	6	0,2

Lines connecting processes denote user-defined communication links (neighbors); arrows show link origins and destinations

Graph Topology Notes

- Given a graph, communication speed may be improved if logical/physical process mapping reordered by system.
- Reorder is not implemented on some systems …
- One node may be declared as neighbor of another without the opposite being true, *i.e.,* asymmetric. If reorder is true, communication efficiency may not be optimal.
- Reorder is implemented on IBM. Graph topology, *i.e., edges* array, must be symmetric. If x is neighbor of y, then y is neighbor of x.
- Graph topology cannot be used in inter-communicators.
- Number of graph nodes must not exceed processors in group.

Graph Topology Routines

- MPI_GRAPH_CREATE -- creates communicator with user-defined graph topology
- MPI_GRAPH_NEIGHBORS_COUNT – returns a given rank's # of neighbors
- MPI_GRAPH_NEIGHBORS -- returns the edges associated with a given rank
- MPI_GRAPH_GET -- returns arrays *index, edges*
- MPI_GRAPHDIMS_GET – returns # nodes, # edges for graph
- MPI_GRAPH_GET – returns arrays *index, edges* of graph
- MPI_TOPO_TEST – returns topology type, *i.e.,* cartesian, graph, or undefined

MPI_Graph_create Usage Example

Fortran :

```
include "mpif.h"
integer graph_comm, nnodes, ierr, index(4), edges(6)
logical reorder
data nnodes/4/,  index/2,3,4,6/,  edges/1,3,0,3,0,2/,  reorder/.true./
call MPI_GRAPH_create(MPI_COMM_WORLD, nnodes, index, &
                      edges, reorder, graph_comm, ierr)
```

.
.
.

Node	*Nneighbors*	*index*	*Edges*
0	2	2	1,3
1	1	3	0
2	1	4	3
3	2	6	0,2

MPI_Graph_create Usage Example

Node	Nneighbors	index	edges
0	2	2	1,3
1	1	3	0
2	1	4	3
3	2	6	0,2

C :

```
#include "mpi.h"
MPI_Comm graph_comm;
int nnodes = 4;          /* number of nodes */
int index[4] = {2, 3, 4, 6};   /* index definition */
int edges[6] = {1, 3, 0, 3, 0, 2};   /* edges definition */
int reorder = 1;         /* allows processes reordered for efficiency */
MPI_Graph_create(MPI_COMM_WORLD, nnodes, index, edges, reorder,
                         graph_comm);
```

MPI_Graph_neighbors_count, MPI_Graph_neighbors

Fortran :

```
integer my_neighbors, my_edges(2)
integer node
          .
          .
call MPI_Comm_rank(graph_comm, node, ierr)
          .
          .
call MPI_Graph_neighbors_count(graph_comm, node, my_neighbors, ierr)
call MPI_Graph_neighbors(graph_comm, node, my_neighbors, my_edges, ierr)
```

Node	*Nneighbors*	*index*	*edges*
0	2	2	1,3
1	1	3	0
2	1	4	3
3	2	6	0,2

MPI_Graph_neighbors_count, MPI_Graph_neighbors

C :

```
int node, my_neighbors, my_edges(2);

        .
        .
MPI_Comm_rank(graph_comm, &node);
        .
        .
MPI_Graph_neighbors_count(graph_comm, node, &my_neighbors);
MPI_Graph_neighbors(graph_comm, node, Nneighbors, my_edges);
```

Node	*Nneighbors*	*index*	*edges*
0	2	2	1,3
1	1	3	0
2	1	4	3
3	2	6	0,2

MPI_Graphdims_get, MPI_Graph_get

Fortran :

```
integer nnodes, nedges, index(4), edges(6)
          .
          .
call MPI_Graphdims_get(graph_comm, nnodes, nedges, ierr)
call MPI_Graph_get(graph_comm, nnodes, nedges, index, edges, ierr)
```

Node	*Nneighbors*	*index*	*edges*
0	2	2	1,3
1	1	3	0
2	1	4	3
3	2	6	0,2

MPI_Graphdims_get, MPI_Graph_get

C :

```
int nnodes, nedges, index[4], edges[6];
        .
        .
MPI_Graphdims_get(graph_comm, &nnodes, &nedges);
MPI_Graph_get(graph_comm, nnodes, nedges, index, edges);
```

Node	*Nneighbors*	*index*	*edges*
0	2	2	1,3
1	1	3	0
2	1	4	3
3	2	6	0,2

Graph Topology Example - Reduction Operation

Let $\oplus$ be an associative, and optionally commutative, reduction operator and let $x_i, i = 0, p-1$ be a set of inputs. Applying this operation on x produces $y = x_0 \oplus x_1 \oplus x_2 \oplus \ldots \oplus x_{p-1}$

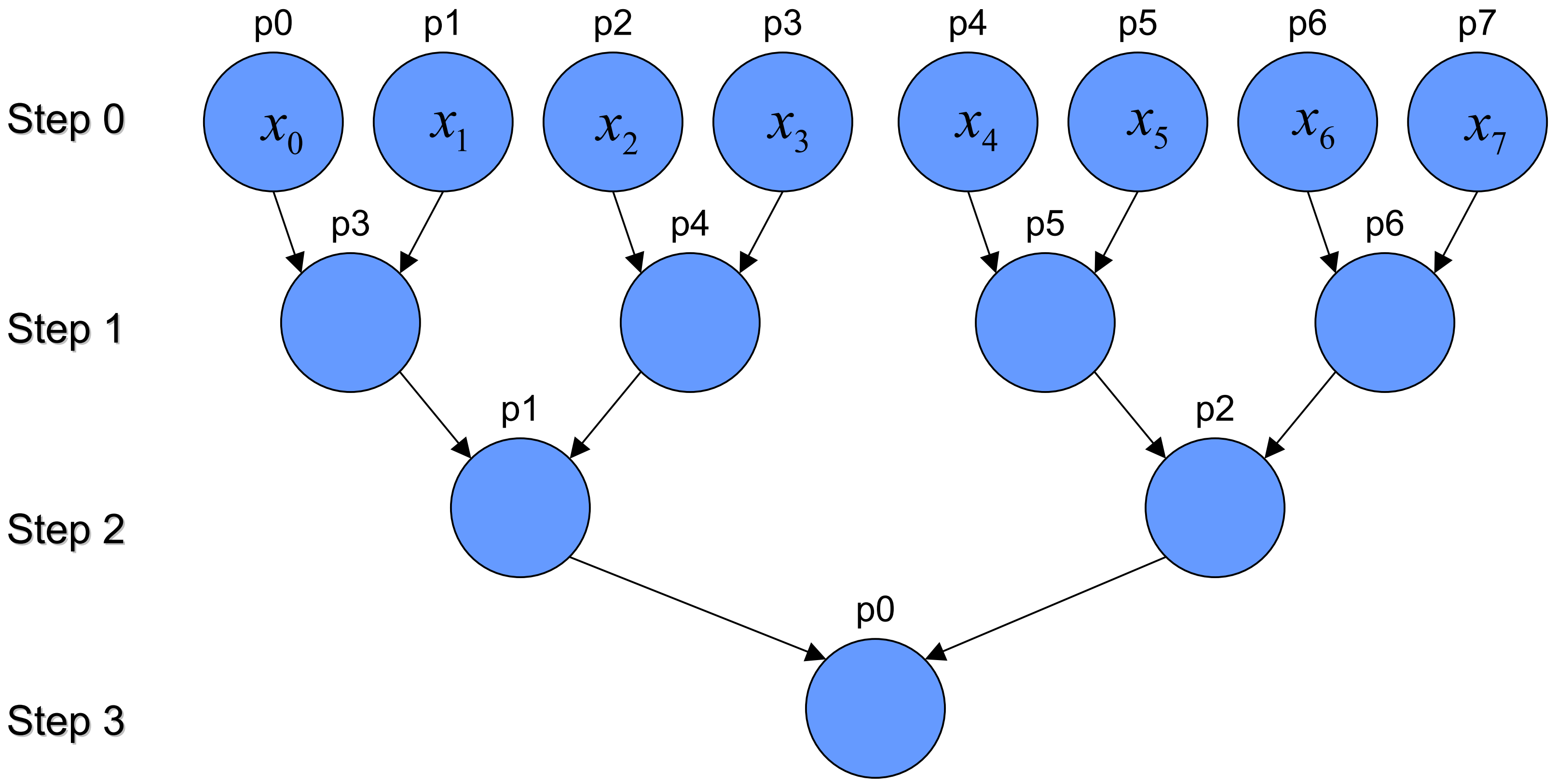
Examples of above operation:

- Numerical integration: $I = \int f(x)dx = \sum_{i=0}^{p-1} f(x_i) * \Delta x$
- Reduction: sum, product, min, max
- Reduction: user-defined operator

This operation may be parallelized, for instance, with a binary tree algorithm. It takes $n = log_2(p)$ steps to complete task.

Binary-tree Parallel Algorithm

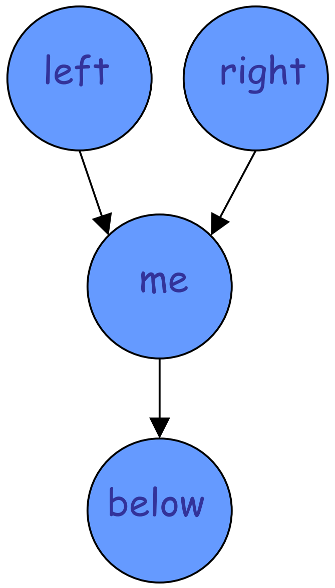
pk denotes process number



$$Total\ \ steps = n = log_2(8) = 3$$

Reduction Operations – Binary Algorithm

For *p* inputs, layout a binary tree with *p* leave nodes (nodes at step 0).

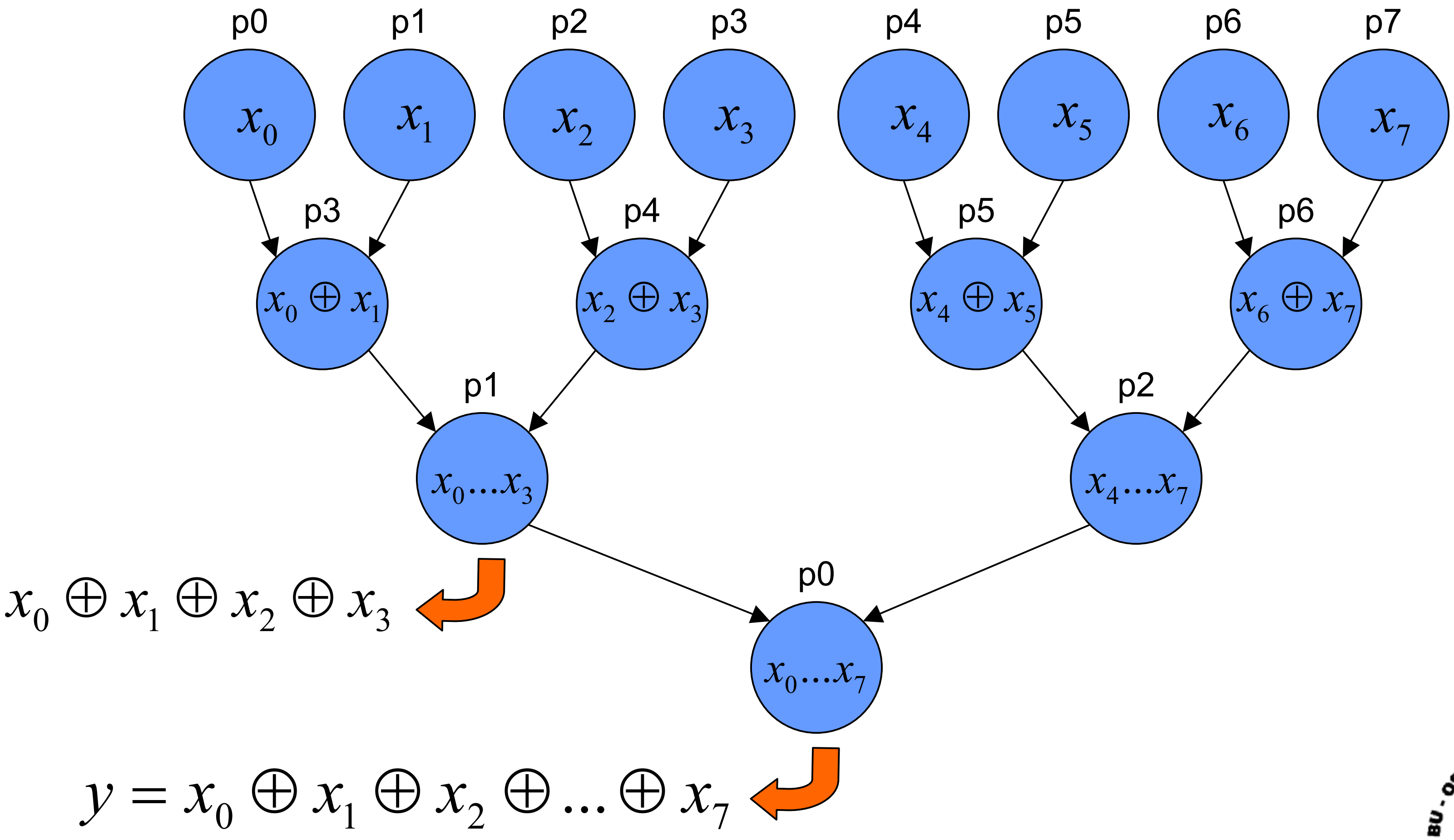


- *Step 0:* Load x_i into the leave nodes and send them to nodes that expect them in Step 1.
- *Step i:* For each node me, performs $\oplus$ on buffers received from nodes left and right (above), sends computed result down to node below. For example, at Step 1, with me = p4, left, right and below are p2, p3, and p1, respectively.
- *Step n:* Node at the bottom performs $\oplus$ to yield y

http://webct.ncsa.uiuc.edu:8900/

Reduction Operations Example

Procedure takes 3 steps to yield reduction solution



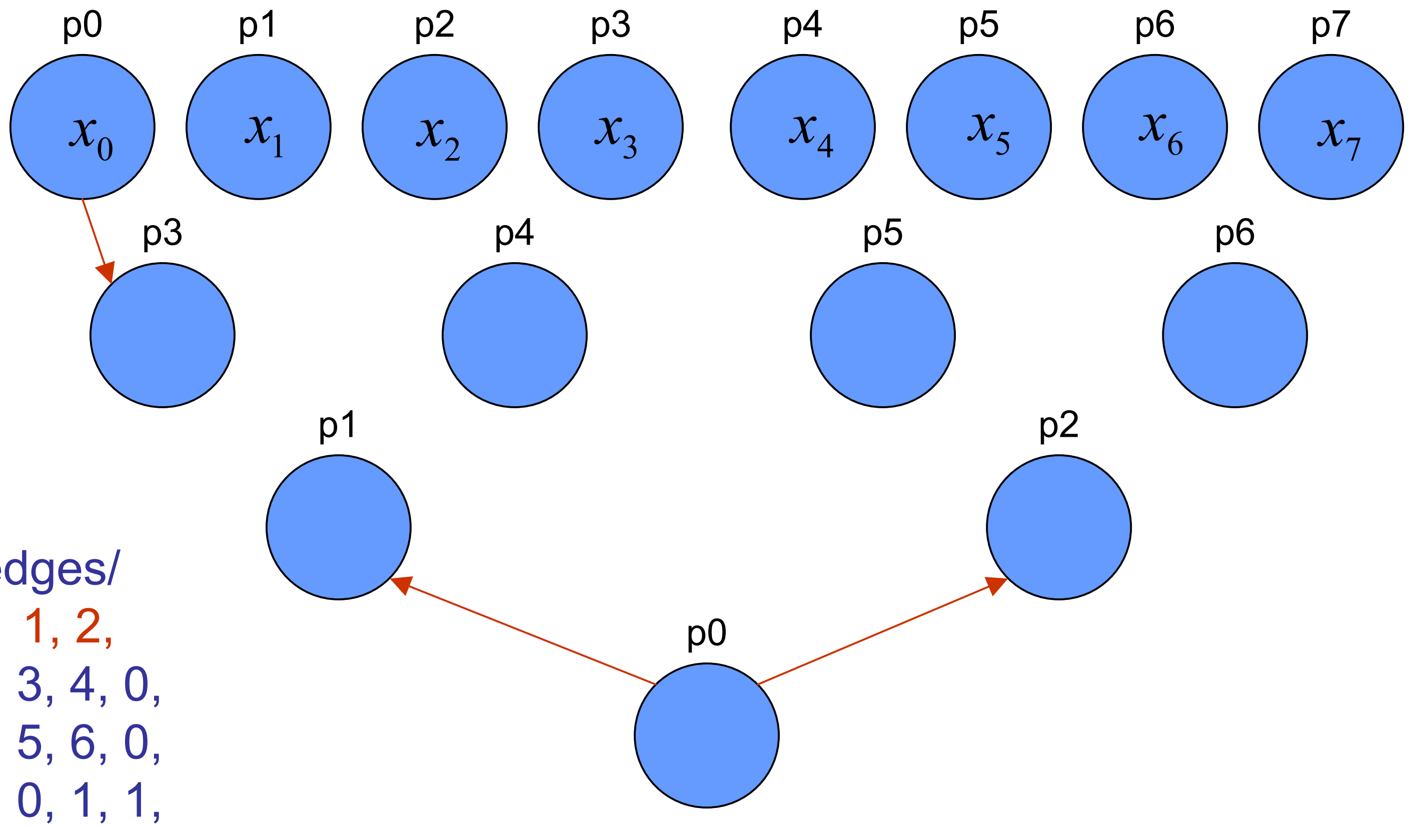
Define Arrays for MPI_Graph_create (Fortran)

Node	Neighbors	Index(1:8)	Edges
0	3	3	3,1,2
1	4	7	3,3,4,0
2	4	11	4,5,6,0
3	4	15	4,0,1,1
4	4	19	5,2,3,1
5	4	23	5,4,5,2
6	4	27	6,6,7,2
7	1	28	6

For Step n, node 0 need only to perform left ⊕ right to yield result

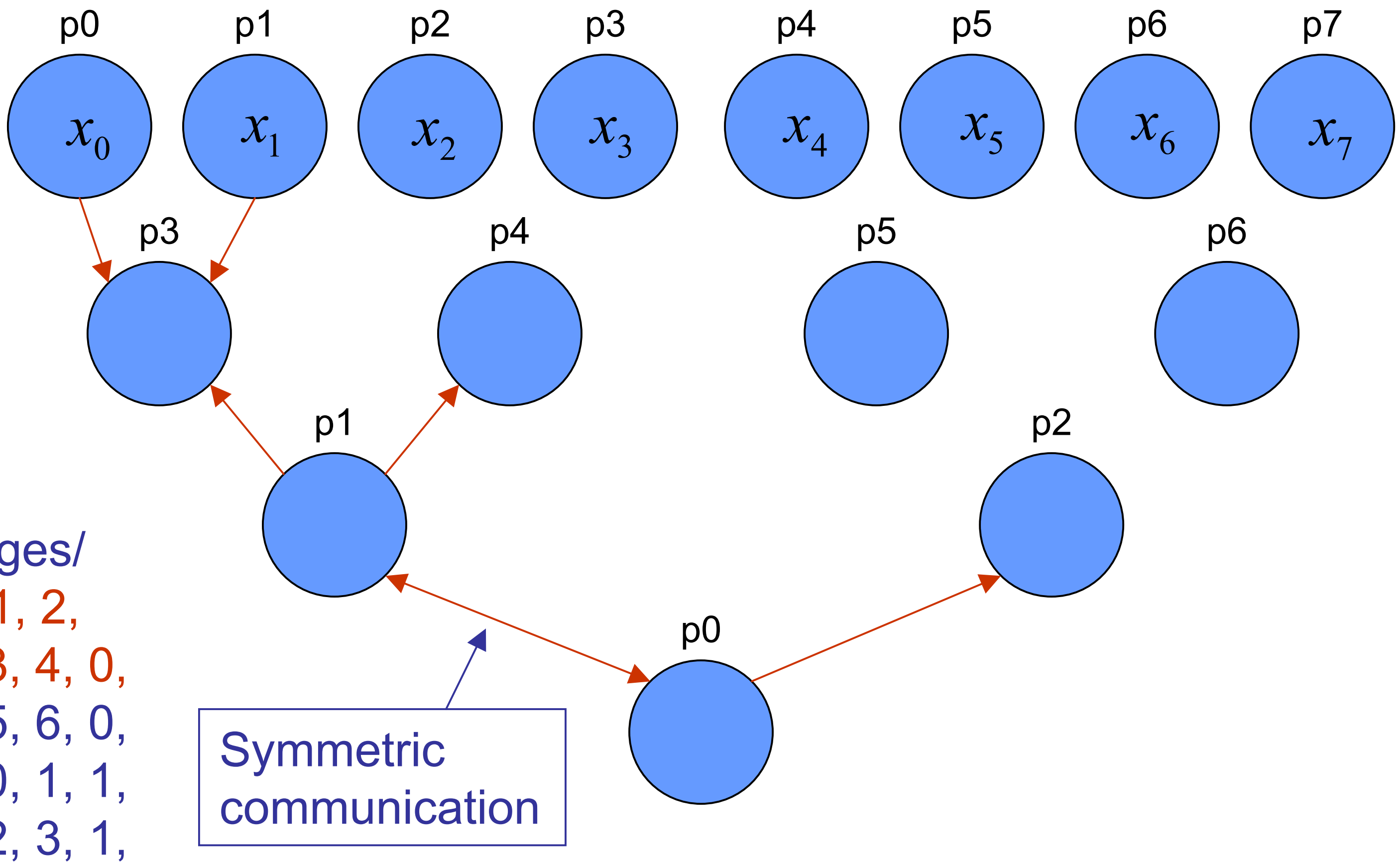
```
  data edges/
*      3,  1, 2,
1     3,  3, 4, 0,
2     4,  5, 6, 0,
3     4,  0, 1, 1,
4     5,  2, 3, 1,    ! Line continuation; remainder of line for node 4
5     5,  4, 5, 2,    ! 1st entry defines Step 0 send destination node
6     6,  6, 7, 2,    ! left, right and below for intermediate steps
7     6/              ! Node 7 only needed in Step 0
```

Edges Array is Symmetric – Graphical Verification



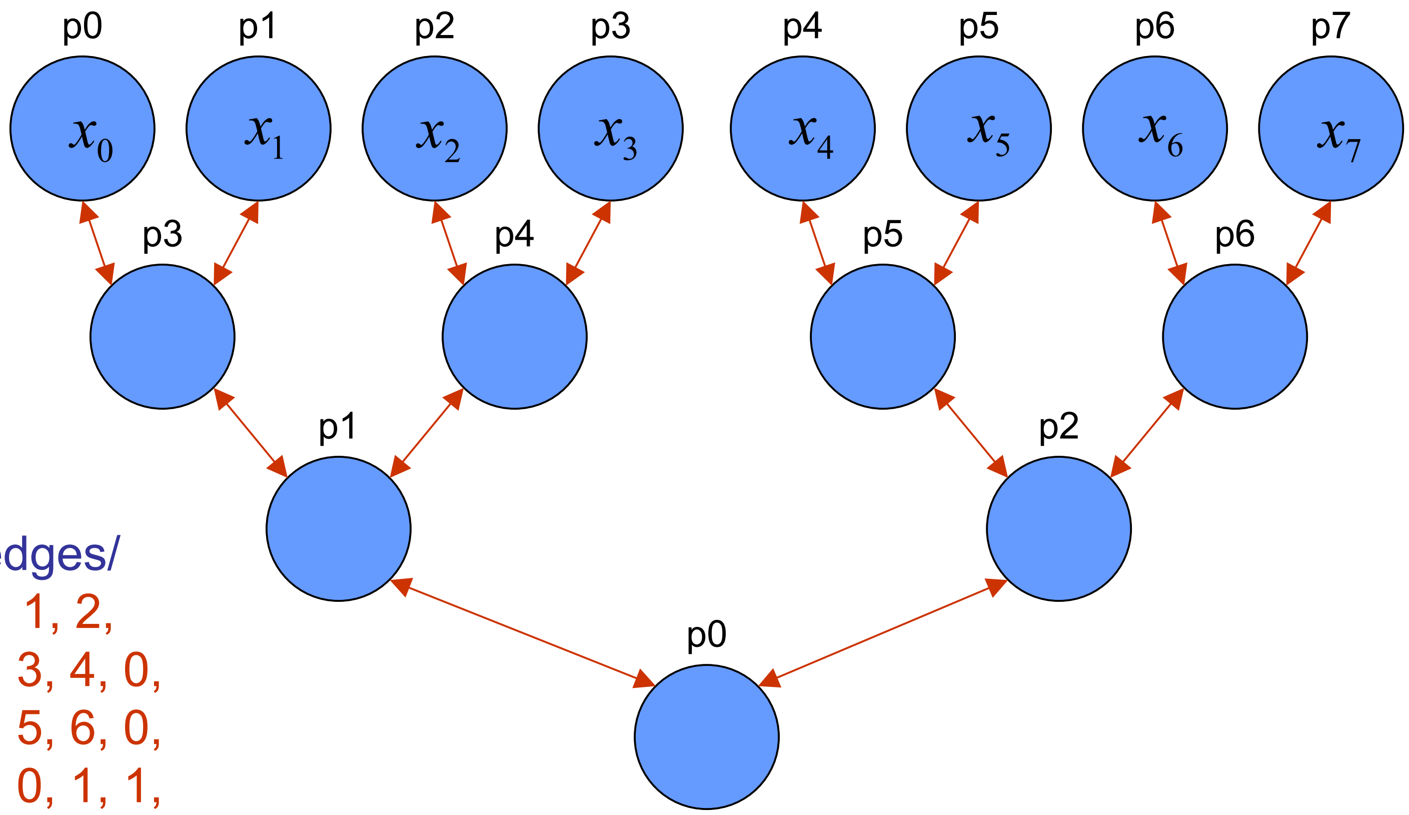
```
  data edges/
*      3,  1, 2,
1      3,  3, 4, 0,
2      4,  5, 6, 0,
3      4,  0, 1, 1,
4      5,  2, 3, 1,
5      5,  4, 5, 2,
6      6,  6, 7, 2,
7      6/
```

Edges Array is Symmetric – Graphical Verification



```
data edges/
*     3,  1, 2,
1     3,  3, 4, 0,
2     4,  5, 6, 0,
3     4,  0, 1, 1,
4     5,  2, 3, 1,
5     5,  4, 5, 2,
6     6,  6, 7, 2,
7     6/
```

Edges Array is Symmetric – Graphical Verification



```
data edges/
*       3,  1, 2,
1       3,  3, 4, 0,
2       4,  5, 6, 0,
3       4,  0, 1, 1,
4       5,  2, 3, 1,
5       5,  4, 5, 2,
6       6,  6, 7, 2,
7       6/
```

Graph Topology Example – Fortran Code

```
 Program graph_example
 implicit none
 integer n, n1, n2, p, i, ierr, comm, comm_graph
 integer xi, result, step_range(2,3), source, tag
 integer neighbors(0:7), index(0:8), edges(28), Nnodes
 integer left, right, below, left_value, right_value
 logical reorder
 data reorder/.false./
 data nnodes/8/, neighbors/3,4,4,4,4,4,4,1/
 data step_range/3,6, 1,2, 0,0/
 data edges/
*      3,  1, 2,        ! Step 0 destination, left, right
1     3,  3, 4, 0,     ! Step 0 destination, left, right, below
2     4,  5, 6, 0,     ! . . .
3     4,  0, 1, 1,     ! . . .
4     5,  2, 3, 1,     ! . . .
5     5,  4, 5, 2,     ! . . .
6     6,  6, 7, 2,     ! . . .
7     6/               ! . . .
```

Graph Topology Example – (cont'd)

```
include "mpif.h"          ! Brings in pre-defined MPI constants, ...
integer Iam, me, status(MPI_STATUS_SIZE)

call MPI_Init(ierr)                                   ! starts MPI
call MPI_Comm_rank(MPI_COMM_WORLD, Iam, ierr)  ! get current process id
call MPI_Comm_size(MPI_COMM_WORLD, p, ierr)       ! get # procs

n = int(alog10(float(p))/alog10(2.0))
index(0) = 0
do i=1,Nnodes
   index(i) = index(i-1) + neighbors(i-1)
enddo
```

Graph Topology Example – (cont’d)

```
      tag = 0
      comm = MPI_COMM_WORLD
C**create graph topology communicator using nnodes, index and edges
      call MPI_Graph_create(comm, Nnodes, index(1), edges, reorder,
     &                      graph_comm, ierr)
      call MPI_Comm_rank(graph_comm, me, ierr)

      xi = lam                    ! Step 0: load xi into leave nodes and send
      call MPI_Isend(xi, 1, MPI_INTEGER, edges(index(me)+1), tag,
     &               graph_comm, req1, ierr)
```

Graph Topology Example – (cont'd)

```
 do i=1,n-1                        ! All steps excluding 0 and n
    n1 = step_range(1,i)      ! begin from node
    n2 = step_range(2,i)      ! end at node
    if (me .ge. n1 .and. me .le. n2) then
       left    = edges(index(me)+2)
       right   = edges(index(me)+3)
       below = edges(index(me)+4)
       call MPI_Recv( left_value, 1, MPI_INTEGER,  left, tag,
&                          comm_graph, status, ierr)   ! Receive from left
       call MPI_Recv(right_value, 1, MPI_INTEGER, right, tag,
&                          graph_comm, status, ierr)   ! Receive from right
       result = left_value + right_value                ! Perform reduction operation
       call MPI_Isend(result, 1, MPI_INTEGER, below, tag,
&                          graph_comm, req2, ierr)      ! Send result to node below
    endif
 enddo
```

Graph Topology Example – (cont’d)

```
 if (me .eq. 0) then                                   ! Step n (last step)
    left   = edges(index(me)+2)
    right = edges(index(me)+3)
    call MPI_Recv( left_value, 1, MPI_INTEGER,  left, tag,
&                          graph_comm, status, ierr)   ! Receive from left
    call MPI_Recv(right_value, 1, MPI_INTEGER, right, tag,
&                         graph_comm, status, ierr)    ! Receive from right
   result = left_value + right_value                   ! Perform reduction operation
   write(*,*)'The global sum is', result               ! Print result
 endif

 call MPI_Finalize(ierr)                                ! # of sends/receives = 14

 stop
 end
```

http://webct.ncsa.uiuc.edu:8900/

Cartesian Topology Example

We demonstrate the application of Cartesian Topology through the solution of a Laplace Equation using finite difference method …

Laplace Equation

Laplace Equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \qquad x, y \in [0,1] \qquad (1)$$

Boundary Conditions:

$$u(x,0) = \sin(\pi x) \qquad 0 \le x \le 1$$

$$u(x,1) = \sin(\pi x)e^{-x} \qquad 0 \le x \le 1 \qquad (2)$$

$$u(0, y) = u(1, y) = 0 \qquad 0 \le y \le 1$$

Analytical solution:

$$u(x, y) = \sin(\pi x)e^{-xy} \qquad x, y \in [0,1] \qquad (3)$$

Laplace Equation Discretized

Discretize $\nabla^2 u = 0$ by centered-difference yields:

$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} \qquad i = 1,2,\ldots,m; \;\; j = 1,2,\ldots,m \qquad (4)$$

where *n* and *n+1* denote current and next time step, respectively, while

$$u_{i,j}^{n} = u^{n}(x_i, y_j) \qquad i = 0,1,2,\ldots,m+1; \;\; j = 0,1,2,\ldots,m+1 \qquad (5)$$

$$= u^{n}(i\Delta x, j\Delta y)$$

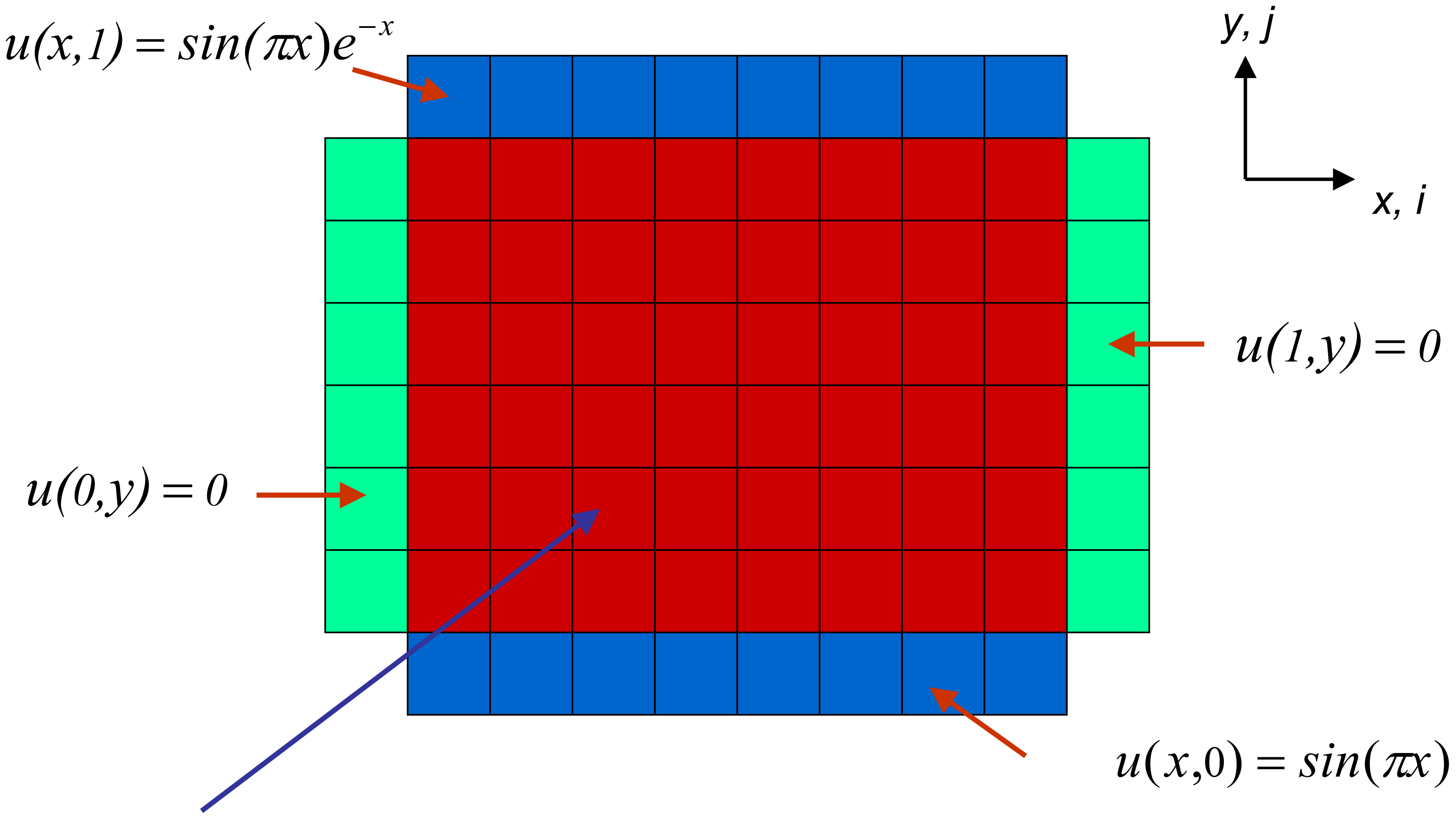
For simplicity, we take

$$\Delta x = \Delta y = \frac{1}{m+1}$$

http://webct.ncsa.uiuc.edu:8900/

Computational Domain

$u(x,1) = \sin(\pi x)e^{-x}$



$u(1,y) = 0$

$u(0,y) = 0$

$u(x,0) = \sin(\pi x)$

$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} \qquad i = 1,2,\ldots,m; \quad j = 1,2,\ldots,m$$

Five-point Finite-Difference Stencil

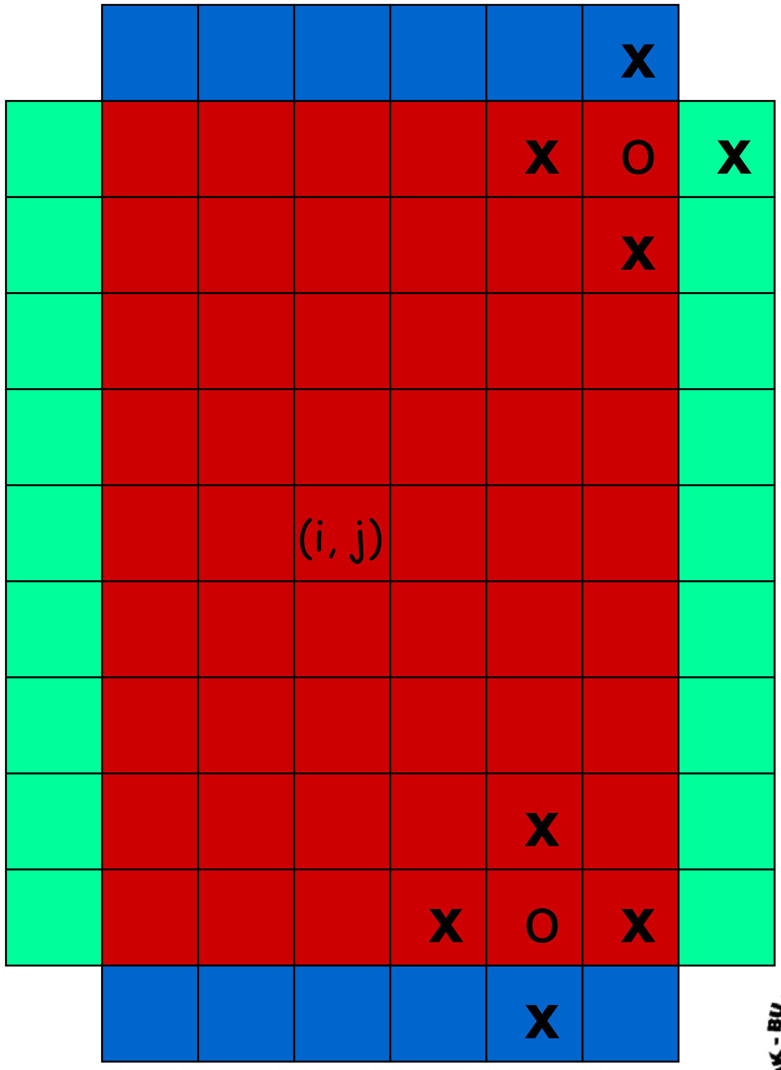
$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4}$$

🟥 Interior (or solution) cells

Where solution of the Laplace equation is sought.

🟦 🟩 Exterior (or boundary) cells

Blue cells denote cells where non-homogeneous boundary conditions are imposed while homogeneous boundary conditions are shown as green cells.



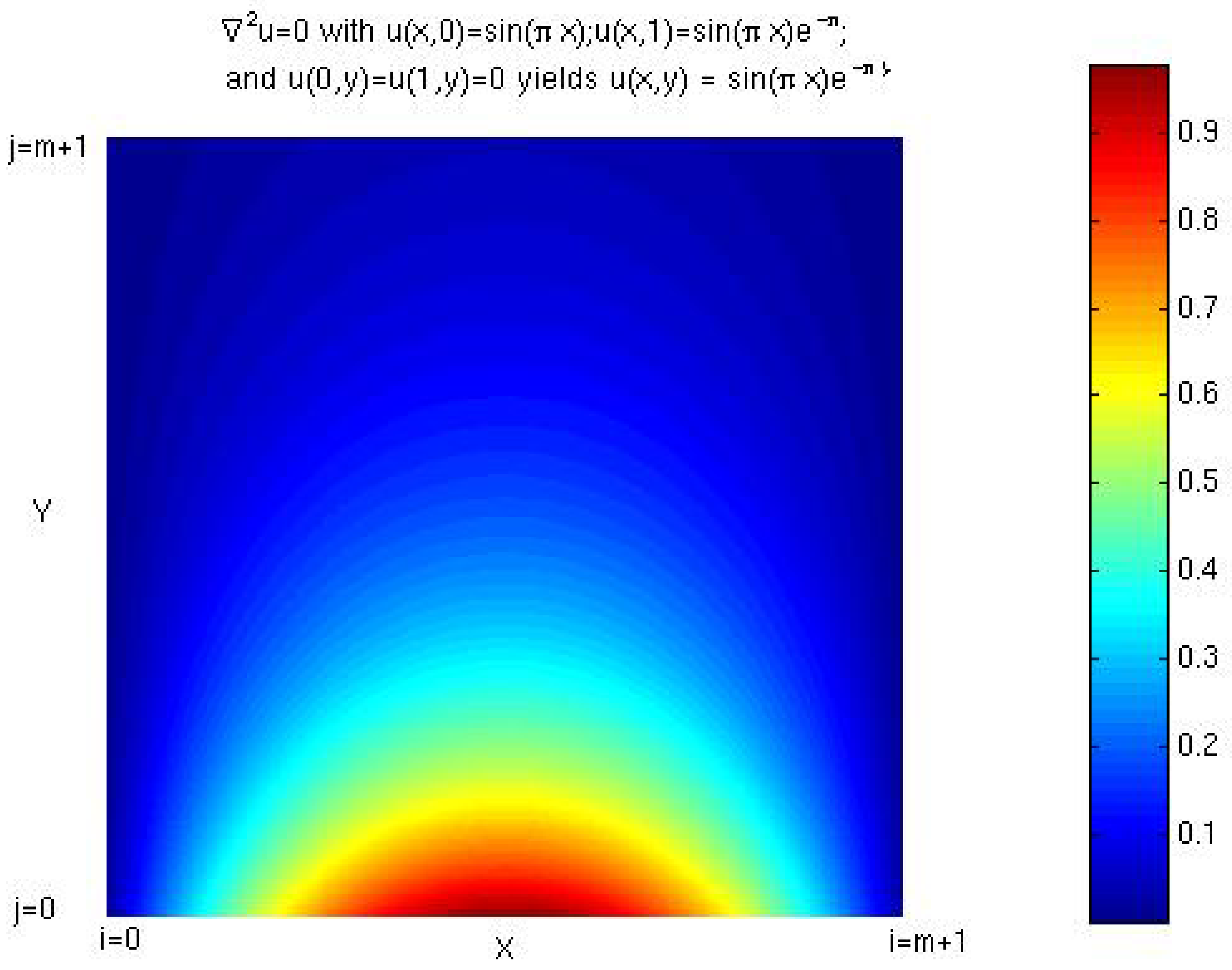
http://webct.ncsa.uiuc.edu:8900/

Jacobi Scheme

1. Make initial guess for *u* at all interior points *(i,j)*.
2. Use 5-pt stencil to compute $u_{i,j}^{n+1}$ at all interior points *(i,j)*.
3. Stop if prescribed convergence threshold is reached, otherwise continue on to the next step.
4. Update: $u_{i,j}^{n} = u_{i,j}^{n+1}$ for all *i* and *j .*
5. Go to step 2.

This is a simple iterative scheme that lends itself as an intuitive instructional procedure. Slowness in convergence renders it impractical for real applications.

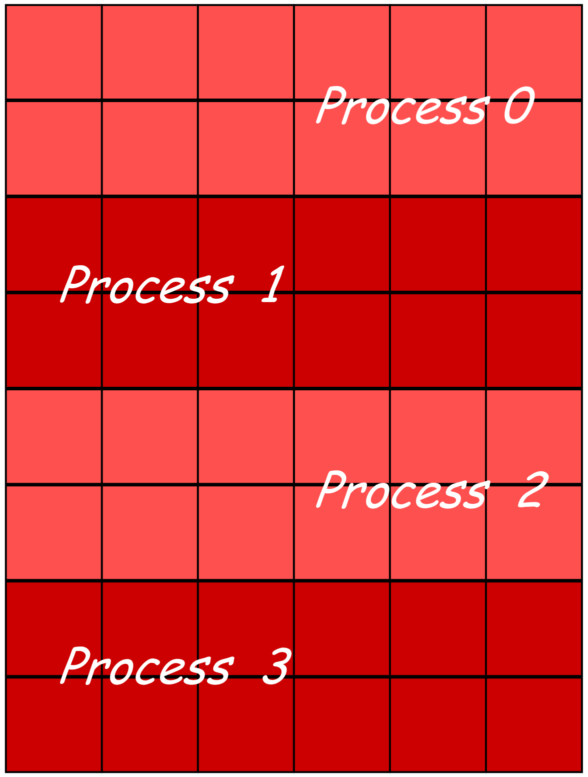
Solution Contour Plot



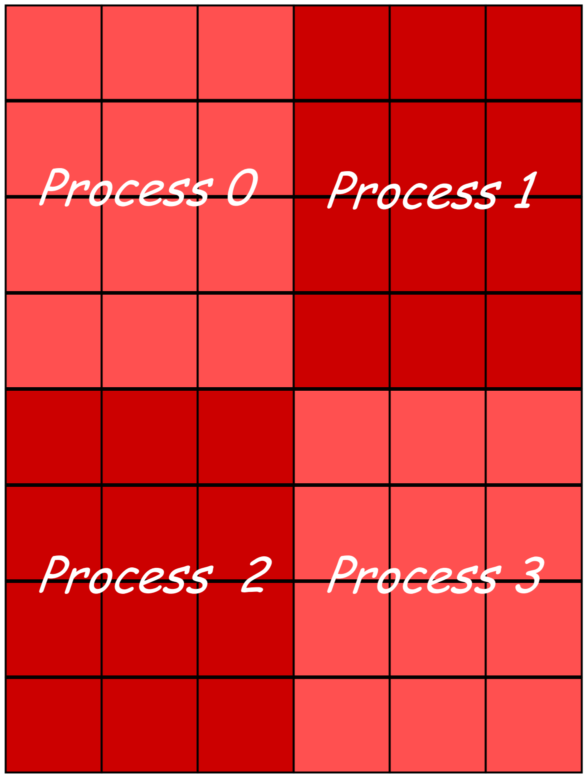
http://webct.ncsa.uiuc.edu:8900/

Domain Decompositions

1D Domain Decomposition



2D Domain Decomposition



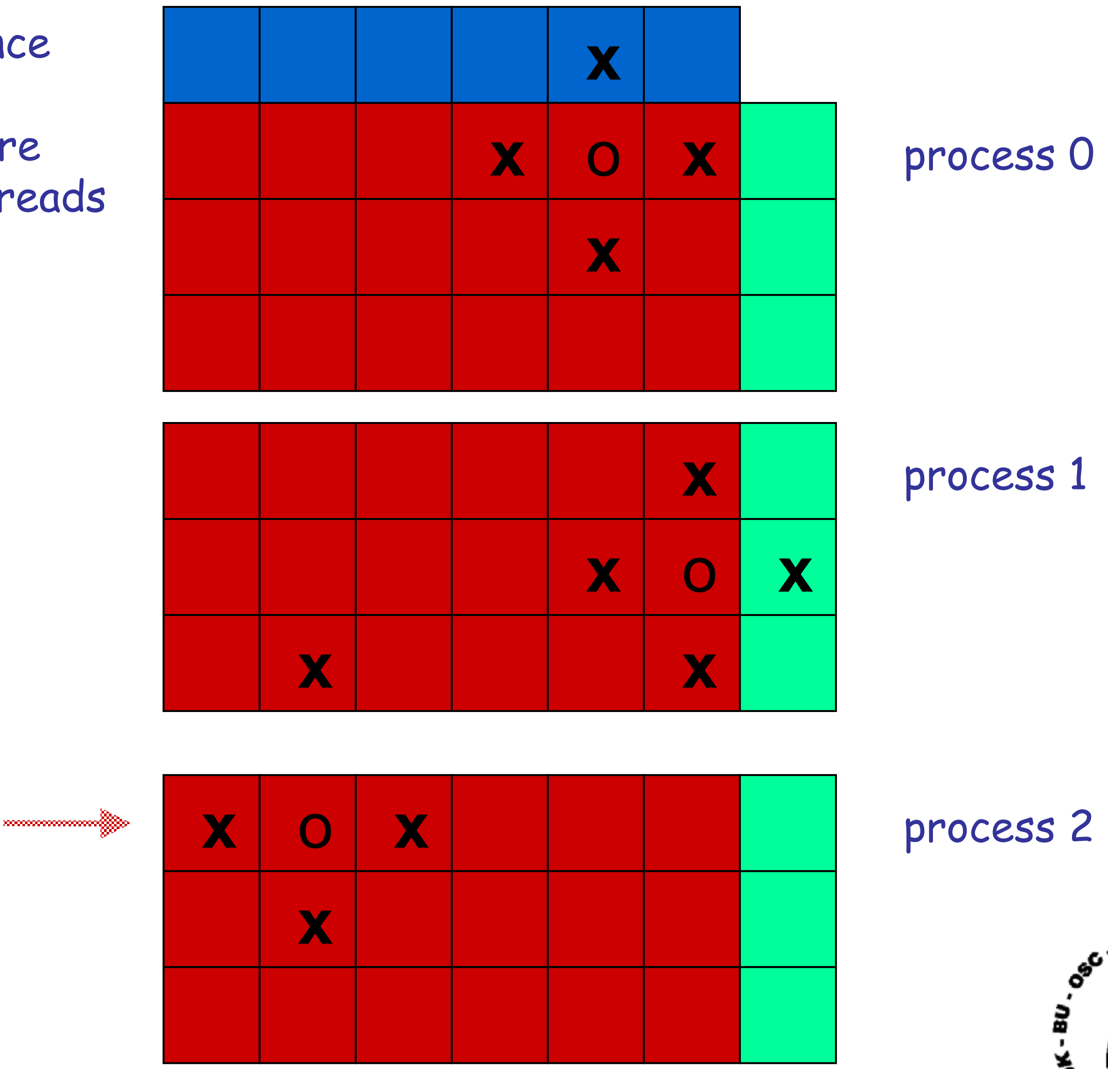
UK - BU - OSC - Alliance - NCSA - HPCERC - PACS Training
Asynchronous MPI Training

http://webct.ncsa.uiuc.edu:8900/

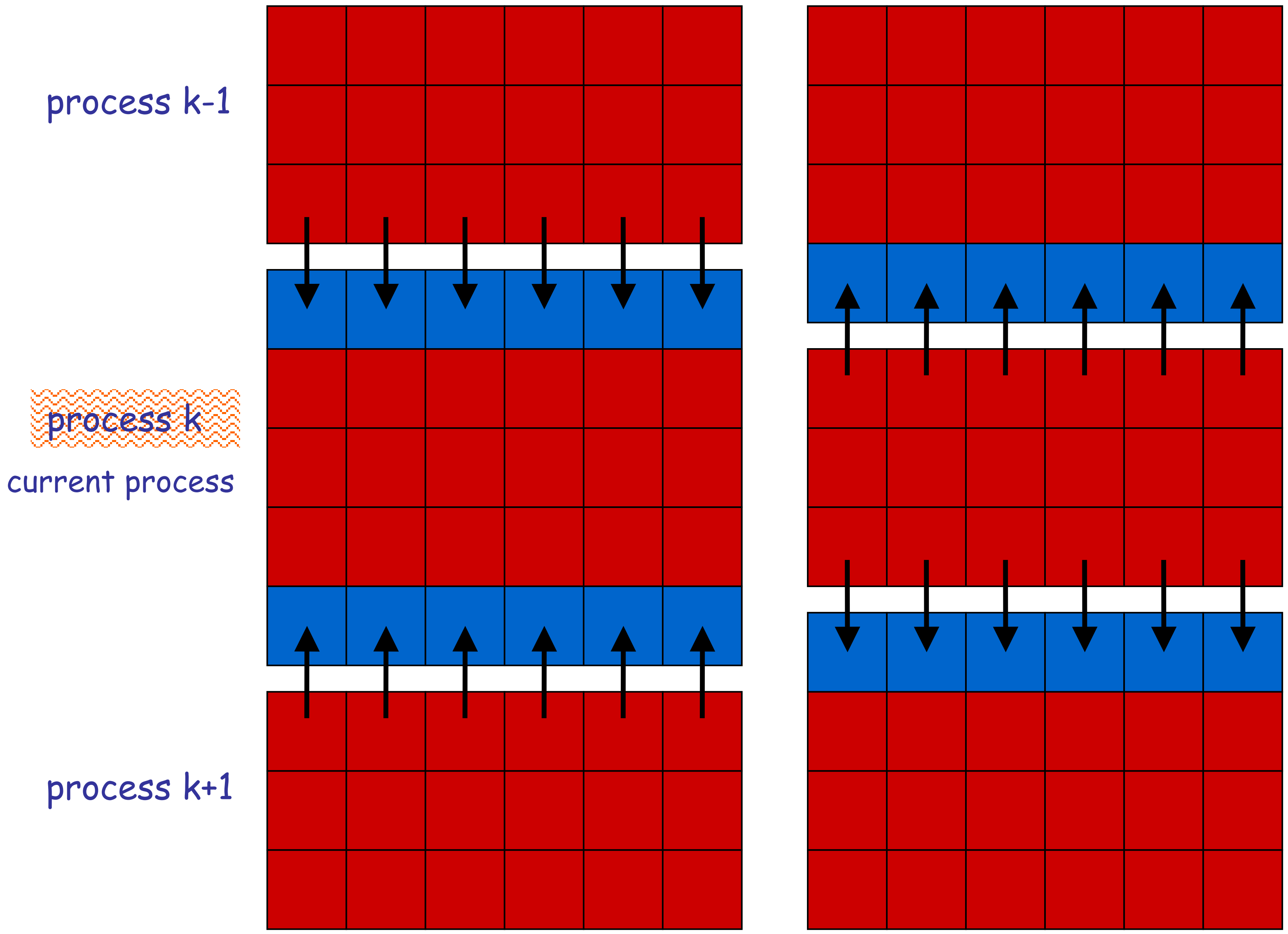
One-Dimensional Domain Decomposition

Five-point finite-difference stencil applied at thread domain border cells require cells from neighboring threads and/or boundary cells.

Message passing required



Message Passing to Fill Boundary Cells



http://webct.ncsa.uiuc.edu:8900/

For Individual Processes . . .

Recast 5-pt finite-difference stencil for individual processes

🟥 $$v^{n+1,k}_{\xi,\eta} = \frac{v^{n,k}_{\xi+1,\eta} + v^{n,k}_{\xi-1,\eta} + v^{n,k}_{\xi,\eta+1} + v^{n,k}_{\xi,\eta-1}}{4}$$

$$\xi = 1,2,\ldots,m; \quad \eta = 1,2,\ldots,m'$$
$$m' = m/p; \quad k = 0,1,2,\ldots,p-1$$

Boundary Conditions

🟦 $v^{n,k}_{\xi,m'+1} = v^{n,k+1}_{\xi,1}; \quad \xi = 0,\ldots,m+1; \quad k = 0$

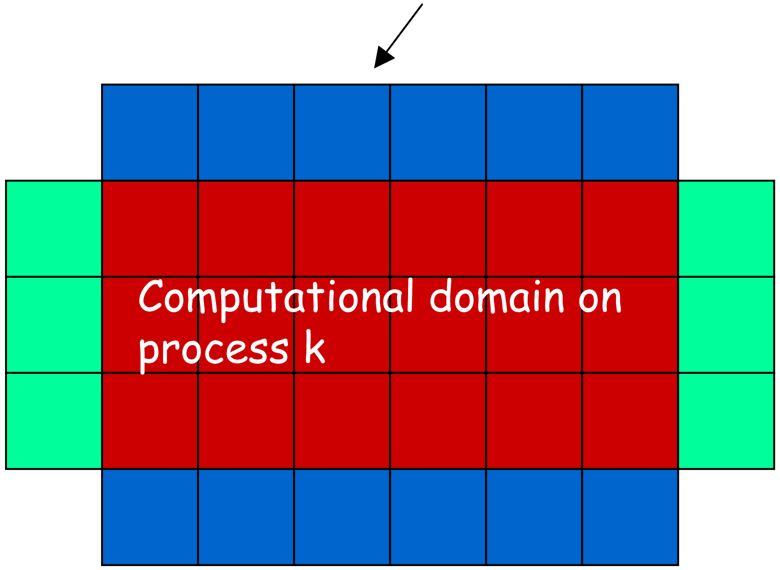
🟦 $v^{n,k}_{\xi,0} = v^{n,k-1}_{\xi,m'}; \quad \xi = 0,\ldots,m+1; \; 0 < k < p-1$

🟦 $v^{n,k}_{\xi,m'+1} = v^{n,k+1}_{\xi,1}; \quad \xi = 0,\ldots,m+1; \; 0 < k < p-1$

🟦 $v^{n,k}_{\xi,0} = v^{n,k-1}_{\xi,m'}; \quad \xi = 0,\ldots,m+1; \; k = p-1$

🟩 $v^{n,k}_{0,\eta} = v^{n,k}_{1,\eta} = 0; \quad \eta = 1,\ldots m'; \; 0 \le k \le p-1$

Cell values obtained from neighboring processes through message passing



- For simplicity, *m* is divisible by *p*
- B.C. time-dependent
- B.C. obtained by message-passing

Relationship Between u and v

Physical boundary conditions

- $v^{n,k}_{\xi,0} = u(x_i,0) = sin(\pi x_i);$ $\quad \xi = i = 0,\ldots,m+1; \quad k=0$
- $v^{n,k}_{\xi,m'+1} = u(x_i,1) = sin(\pi x_i)e^{-\pi};$ $\quad \xi = i = 0,\ldots,m+1; \quad k = p-1$
- $v^{n,k}_{0,\eta} = u(0,y_{\eta+k*m'}) = 0;$ $\quad \eta = 1,\ldots,m'; \; 0 \le k \le p-1$
- $v^{n,k}_{m+1,\eta} = u(1,y_{\eta+k*m'}) = 0;$ $\quad \eta = 1,\ldots,m'; \; 0 \le k \le p-1$

Relationship between global solution *u* and thread-local solution *v*

$$u^{n}_{\xi,\eta+k*m'} = v^{n,k}_{\xi,\eta}$$

$\xi = 1,2,\ \ldots, m; \quad \eta = 1,2,\ \ldots, m'$

$m' = m/p; \quad k = 0,1,2,\ \ldots, p-1$

MPI Functions Used For Jacobi Solver

- *MPI_Sendrecv* (= *MPI_Send* + *MPI_Recv*) – to set boundary conditions for individual threads
- *MPI_Cart_Create* – to create Cartesian topology
- *MPI_Cart_Coords* – to find equivalent Cartesian coordinates of given rank
- *MPI_Cart_Rank* – to find equivalent rank of Cartesian coordinates
- *MPI_Cart_shift* – to find current thread's adjoining neighbor threads
- *MPI_Allreduce* – to search for global error to determine whether convergence has been reached.

Jacobi Solver for 2D Laplace Equation

Fortran:

```
      CALL MPI_Comm_rank(MPI_COMM_WORLD, me, ierr)   ! current rank
        :
        :
        :
      start_time = MPI_Wtime()              ! starts wallclock, measured in seconds
! create 2D cartesian topology for matrix
      CALL MPI_Cart_create(MPI_COMM_WORLD, ndim, dims,
     &        periods, reorder, comm_2d, ierr)
      CALL MPI_Comm_rank(comm_2d, k, ierr)          ! me .ne. k if reorder=.true.
      CALL MPI_Cart_coords(comm_2d, k, ndim, coord, ierr)
      CALL bc2D(m, mp, n, np, v, coord, dims)      ! Initialize boundary condition
      CALL MPI_Cart_shift(comm_2d, 0, 1, below, above, ierr)
      CALL MPI_Cart_shift(comm_2d, 1, 1,  left, right, ierr)
      CALL MPI_Op_create(onenorm, commute, myop)
```

Jacobi Solver for 2D Laplace Equation (cont'd)

```
   iter = 0                                            ! Initialize iteration counter
   DO WHILE (gdel .gt. TOL)                            ! iterate until error < TOL
       iter = iter + 1                                 ! increment iteration counter
       CALL update_jacobi_2D(mp, np, v, vnew, del)         ! Update solution
       IF(MOD(iter,INCREMENT) .eq. 0) THEN     ! Check gdel periodically
!  Compute global error
        CALL MPI_Allreduce( del, gdel, 1, MPI_DOUBLE_PRECISION,
     &          myop, comm_2d, ierr )                     ! Or use MPI_MAX
        IF(k .eq.  0) WRITE(*,'(i7,d13.5)')iter,gdel      ! Print on rank 0
       ENDIF
      CALL update_bc_2D( mp, np, v, below, above, left, right, comm_2d)
   ENDDO
   end_time = MPI_Wtime()              ! Stop timer
```

Jacobi Solver for 2D Laplace Equation

C :

```
    start_time = MPI_Wtime();
    MPI_Comm_rank(MPI_COMM_WORLD, &me);
/* create 2D cartesian topology for matrix */
    MPI_Cart_create(MPI_COMM_WORLD, ndim, dims,
          periods, reorder, &comm_2d);
    MPI_Comm_rank(comm_2d, &k);          /* me != k if reorder=1 */
    MPI_Cart_coords(comm_2d, k, ndim, coord);
    bc2D( m, mp, n, np, v, coord, dims);  /* boundary conditions */

    MPI_Cart_shift(comm_2d, 0, 1, &below, &above);
    MPI_Cart_shift(comm_2d, 1, 1,  &left, &right);
    MPI_Op_create(onenorm, commute, &myop);
```

Problem Set

1. Write a program to perform the equivalent of MPI_MAX
2. Using graph topology, rewrite the parallel reduction example program using your own approach.