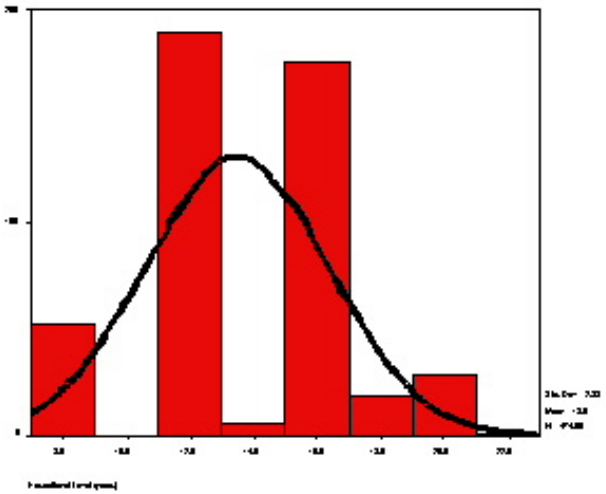
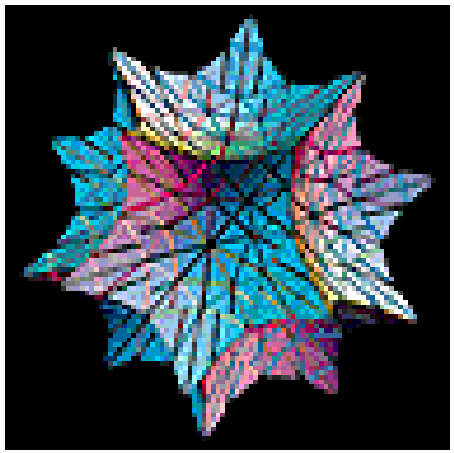


Introduction to MPI
February 18**, 2004**

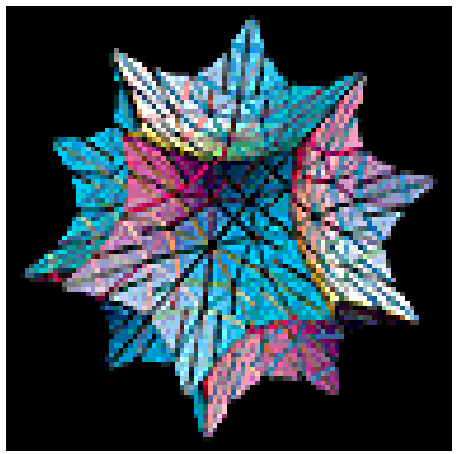


Presented by the
ITC Research Computing Support Group

Kathy Gerber, Ed Hall, Katherine Holcomb, Tim F. Jost Tolson

- Introduction toMPI– February 18
- Beyond Powerpoint: Alternative Software for Scientific/Technical Presentations
  - Wednesday, March 17
- Minitab for Windows, Version 14: What's New
  - Wednesday, March 31
- Computing with the IMSL Scientific Libraries
  - Wednesday, April 14

ITC Research Computing Support
Introduction to MPI



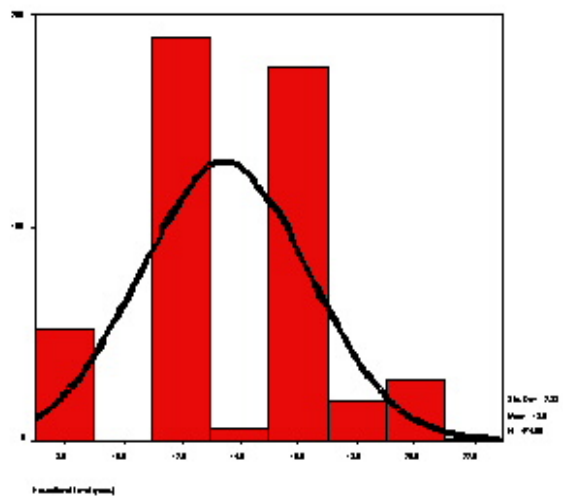
By: Katherine Holcomb

Research Computing Support Center

Phone: 243-8800 Fax: 243-8765

E-Mail: Res-Consult@Virginia.EDU

http://www.itc.Virginia.edu/researchers



MPI=Message-Passing Interface

- Standard
  - Defined by an open specification
- Portable
  - Implemented by most vendors of MPP and clusters
- Simple
  - Basic functionality is easy to learn

MPI is a Library

- Standard bindings exist for C and Fortran
- C++ : An interface exists, but is not as well standardized or documented as the C interface. Many C++ programmers use the C bindings.
- Fortran 90 users can use a module
  - use mpi
  - or can include the Fortran 77 include file

Messages

In MPI, a message consists of data+envelope

The envelope is information that uniquely identifies the source, the destination, and the identification of the message. It consists of

Sender's rank:
the process ID

Receiver's rank:
the process ID

Tag:
an arbitrary identifier

Communicator:
an ID for a group of processes that can exchange data

MPI supplies a predefined communicator, MPI_COMM_WORLD, consisting of all processes running at the start of execution

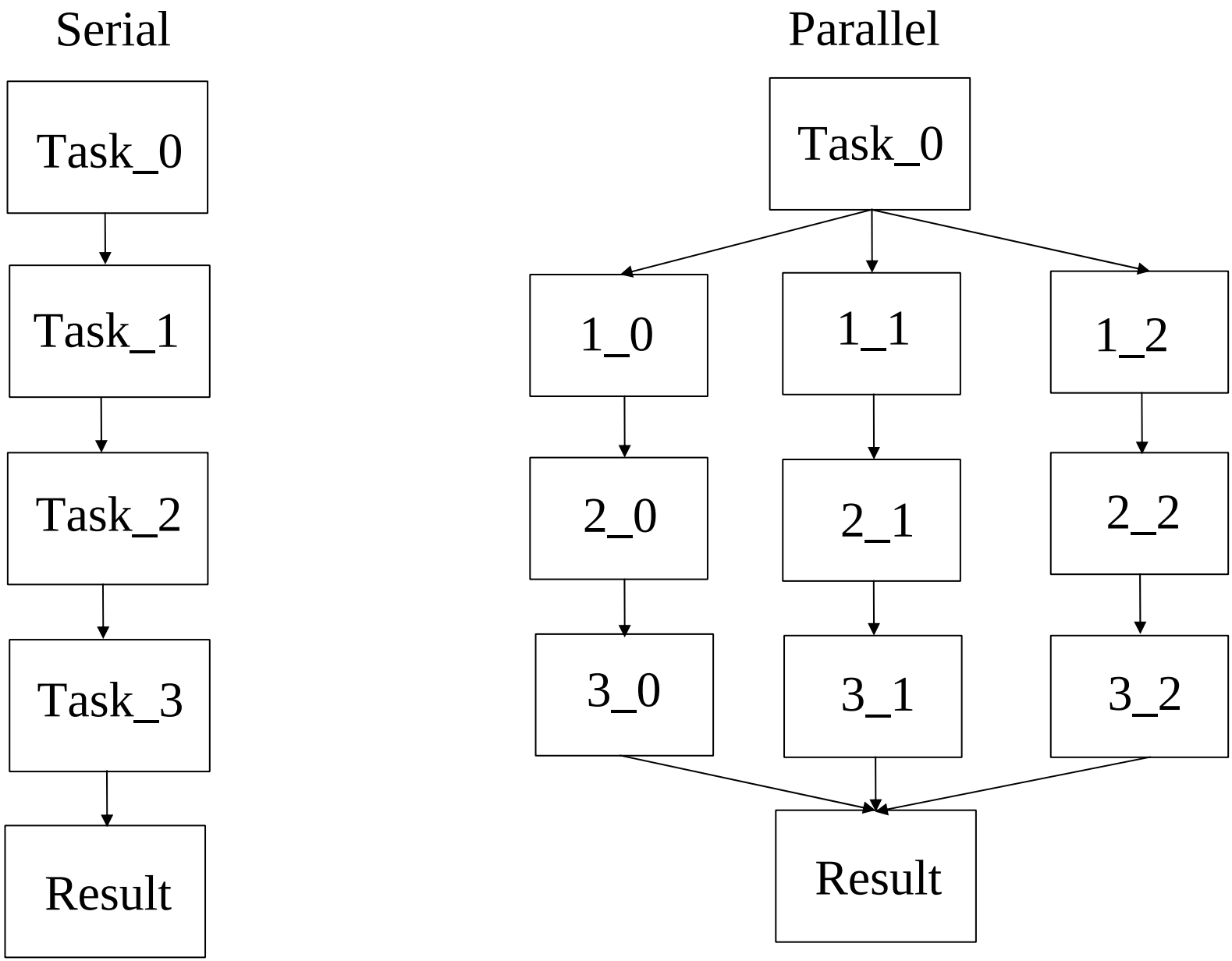
MPI Datatype	C
MPI_INT	signed int
MPI_SHORT	signed short
MPI_LONG	signed long
MPI_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short
MPI_UNSIGNED	
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI Datatype	Fortran
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER
MPI_BYTE	
MPI_PACKED	

Collective Communications

- Scatter/Broadcast: send information from one processor to all. Broadcast sends same data to all; scatter can send different data to each process.
- Gather: receive information from all processes at one process.
- Barrier Synchronization
- Global Reduction Operations
  - Sum, product, max, min, others: gathers data and performs global operation

Collective Communications



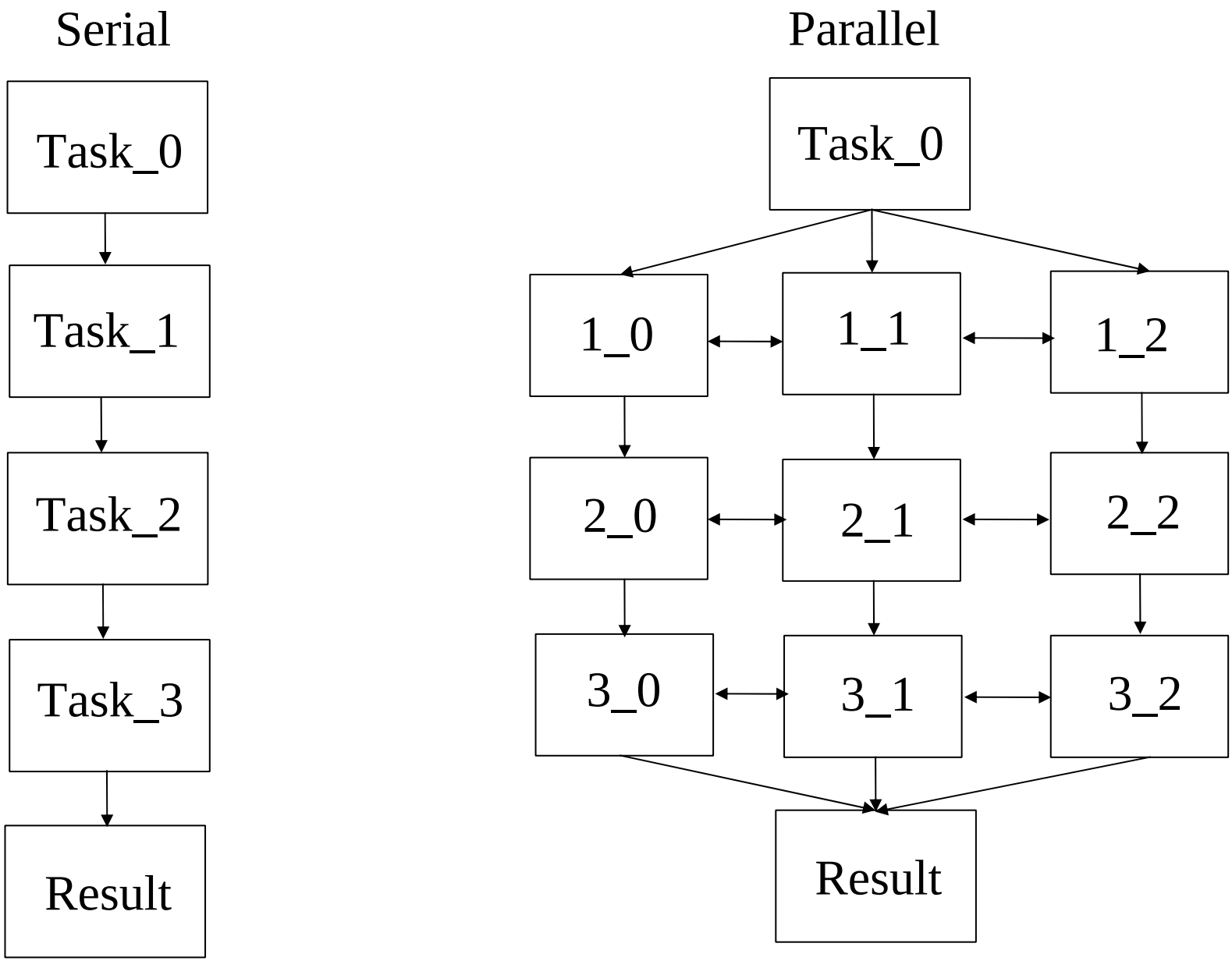
Global Reduction Operations

- MPI_MAX maximum
- MPI_MIN minimum
- MPI_SUM sum
- MPI_PROD product
- MPI_I_AND logical and
- MPI_BAND bitwise and
- MPI_I_OR logical or
- MPI_BOR bitwise or
- MPI_I_XOR logical xor
- MPI_BXOR bitwise xor

Point-to-Point Communications

- Send and Receive
  - Process sends data from itself to one other process or receives data from one other process
  - Accomplished via writing and reading buffers
  - Blocking or Nonblocking
- Within a processor, messages are *ordered*.
  - First message sent arrives first, second next, etc.
- Among different processes, messages are **not** ordered.
  - Parallel codes are, in general, nondeterministic.

A Common Parallel Pattern



MPI Send-Receive

Blocking send and receive

Send: returns when message has been buffered or sent so that memory allocated for the message can be reused – does not mean delivered!

Receive: returns when data has been received into memory referenced for data

C:

```
int my_rank, my_neighbor, tag=50;
MPI_Status status

int mdata[100], idata[100]
...
MPI_Comm_rank(MPI_COMM_WORLD, &myrank)
...
MPI_Send(mdata,100,MPI_INT,my_neighbor,tag, MPI_COMM_WORLD);
MPI_Recv(idata,100,MPI_INT,my_neighbor,tag,MPI_COMM_WORLD,&status);
```

Blocking Send-Recv and Safety

Deadlock:

```
call MPI_Comm_rank(comm,rank,ierr)
if (rank .eq. 0) then
        call MPI_Recv(recvbuf,count,MPI_REAL,1,tag,comm,status,ierr)
        call MPI_Send(sendbuf,count,MPI_REAL,1,tag,comm,ierr)
else if (rank  .eq. 1) then
        call MPI_Recv(sendbuf,count,MPI_REAL,0,tag,comm,status,ierr)
        call MPI_Send(sendbuf,count,MPI_REAL,0,tag,comm,ierr)
endif
```

Safe:

```
call MPI_Comm_rank(comm,rank,ierr)
if (rank .eq. 0) then
        call MPI_Send(sendbuf,count,MPI_REAL,1,tag,comm,ierr)
        call MPI_Recv(recvbuf,count,MPI_REAL,1,tag,comm,status,ierr)
else if (rank  .eq. 1) then
        call MPI_Recv(sendbuf,count,MPI_REAL,0,tag,comm,status,ierr)
        call MPI_Send(sendbuf,count,MPI_REAL,0,tag,comm,ierr)
endif
```

Another Unsafe Pattern

Unsafe: Sending to two neighbors simultaneously

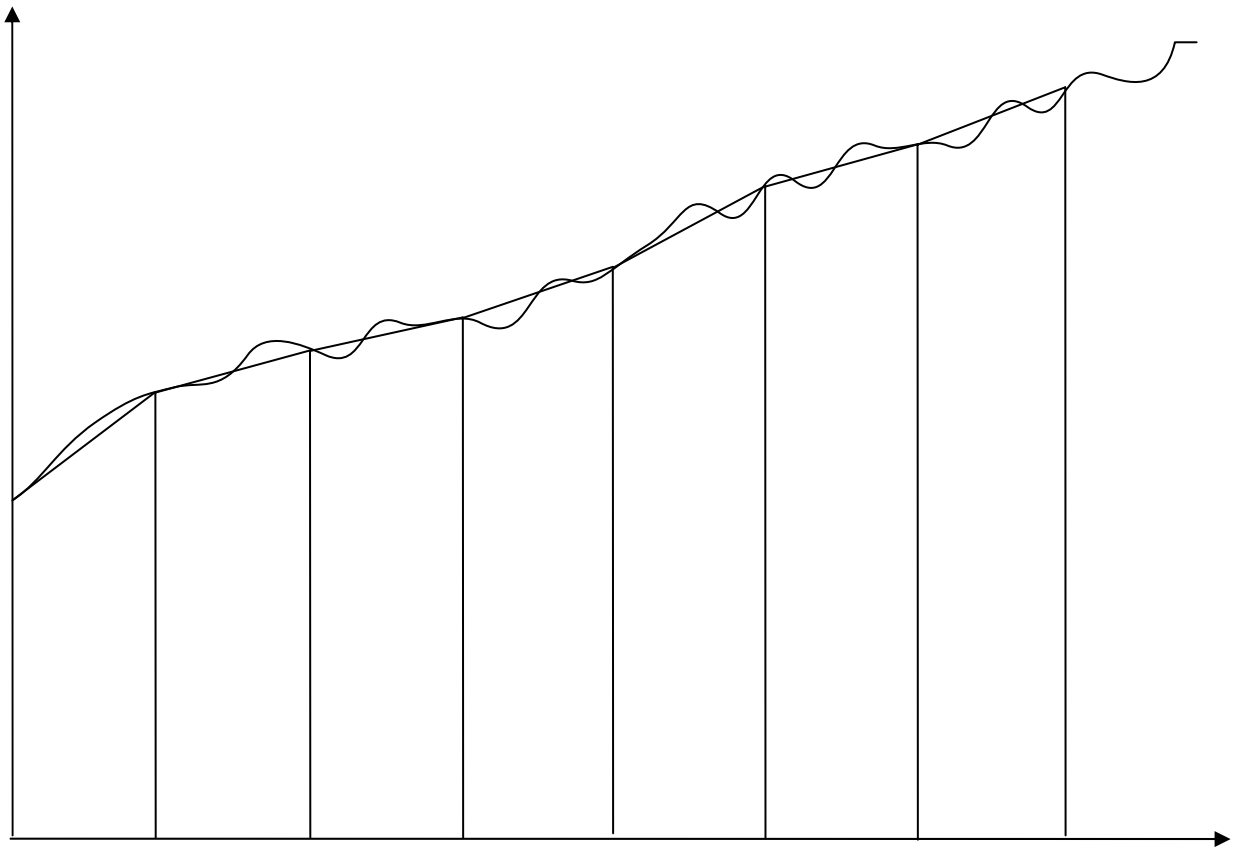
```
call MPI_Comm_rank(comm,rank,ierr)
call MPI_Send(rightbuf,count,MPI_REAL,rank+1,tag,comm,ierr)
call MPI_Send(leftbuf,count,MPI_REAL,rank-1,tag,comm,ierr)

call MPI_Recv(leftbc,count,MPI_REAL,rank-1,tag,comm,status,ierr)
call MPI_ Recv(rightbc,count,MPI_REAL,rank+1,tag,comm,ierr)
```

Safe:

```
call MPI_Comm_rank(comm,rank,ierr)
if (mod(rank,2) .ne. 0) then
   call MPI_Send(rightbuf,count,MPI_REAL,rank+1,tag,comm,ierr)
   call MPI_Recv(leftbc,count,MPI_REAL,rank1,tag,comm,status,ierr)
else if (rank  .eq. 1) then
   call MPI_Recv(rightbuf,count,MPI_REAL,rank+1,tag,comm,status,ierr)
   call MPI_Send(leftbc,count,MPI_REAL,rank-1,tag,comm,ierr)
endif
```

(Real code must account for the special cases rank=0 and rank=npes-1 also.)

EXAMPLE 1: Parallelizing the Trapezoid Rule



```
program trapezoid
implicit none

! Calculate a definite integral using trapezoid rule

real    :: a, b
integer :: n

real    :: h, integral

real    :: f,x
integer :: i

read(*,*) a, b, n
h=(b-a)/n

integral = (f(a) + f(b))/2.0
x=a
do i=1, n-1
     x = x+h
     integral = integral + f(x)
enddo

integral = h*integral
print *, integral
stop
end
```

```
program partrap
implicit none

real   :: integral, total
real   :: a, b, h
integer:: n

real   :: local_a, local_b
integer:: local_n

real   :: trap, f

include 'mpif.h'

integer:: my_rank, p
integer:: source, dest
integer, parameter :: tag=50
integer:: ierr, status(MPI_STATUS_SIZE)

call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
call MPI_Comm_size(MPI_COMM_WORLD, p, ierr)

if (my_rank .eq. 0) then
     read(*,*) a, b, n
endif
```

```
call MPI_Bcast(a,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)
call MPI_Bcast(b,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)
call MPI_Bcast(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

h= (b-a)/n
local_n = n/p

local_a = a + my_rank*local_n*h
local_b = local_a + local_n*h
integral = trap(local_a, local_b, local_n, h)

call MPI_Reduce(integral, total, 1, MPI_REAL, MPI_SUM, &
                                    0, MPI_COMM_WORLD,ierr)

if (my_rank .eq. 0) then
     print *, total
endif

call MPI_Finalize(ierr)

stop
end
```

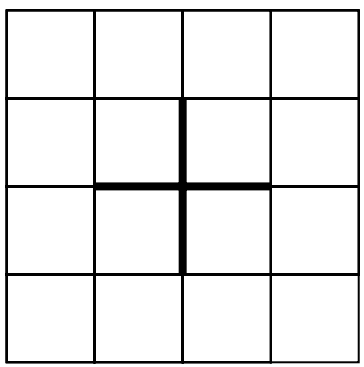
```
real function trap(local_a, local_b, local_n, h)

implicit none

real   :: local_a, local_b, h
integer:: local_n

real   :: f,x
integer:: i
real   :: parint

parint = (f(local_a) + f(local_b))/2.0

trap   = parint*h

return
end


real function f(x)
implicit none
real  :: x

f=sin(x) + cos(x)
return
end
```

EXAMPLE 2: Jacobi Iteration

Laplace Equation: $\nabla^2 T = 0$

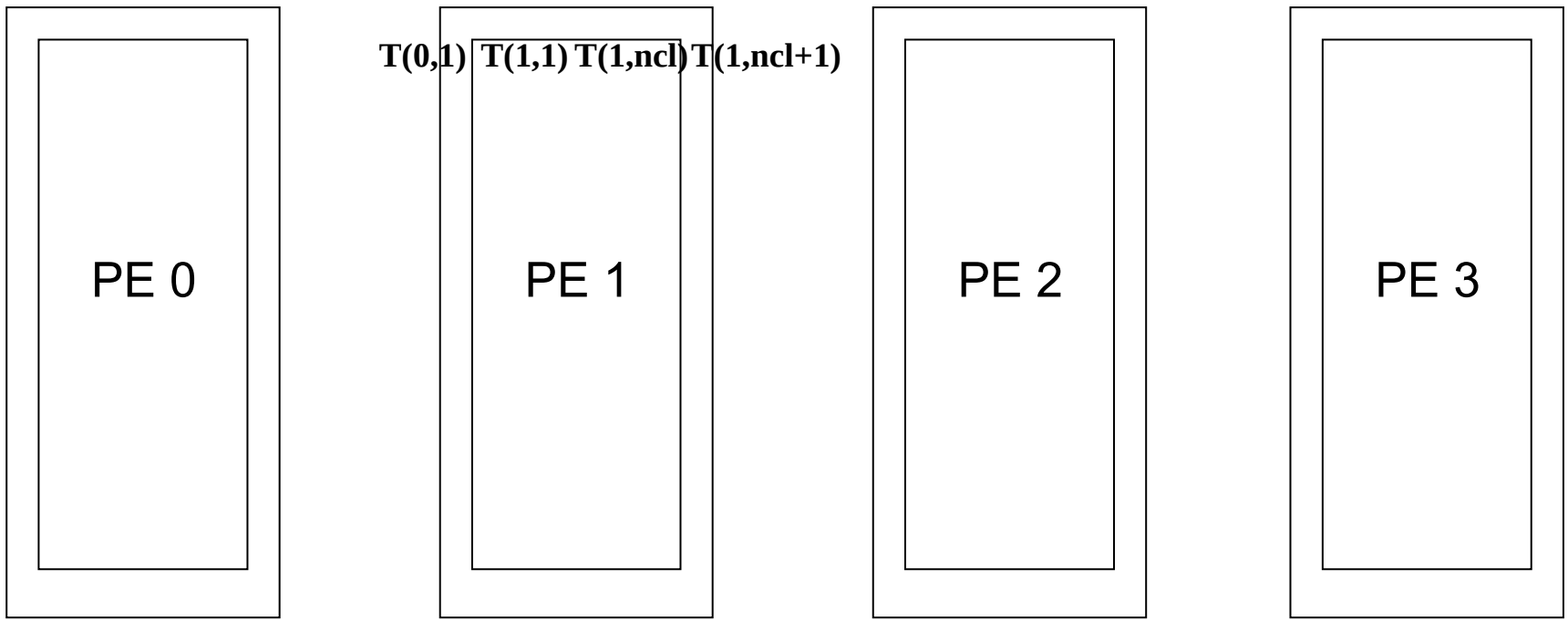
$T_n = 0.25*(T_{n-1}(i-1,j) + T_{n-1}(i+1,j) + T_{n-1}(i,j-1) + T_{n-1}(i,j+1))$

This leads to a five-point stencil



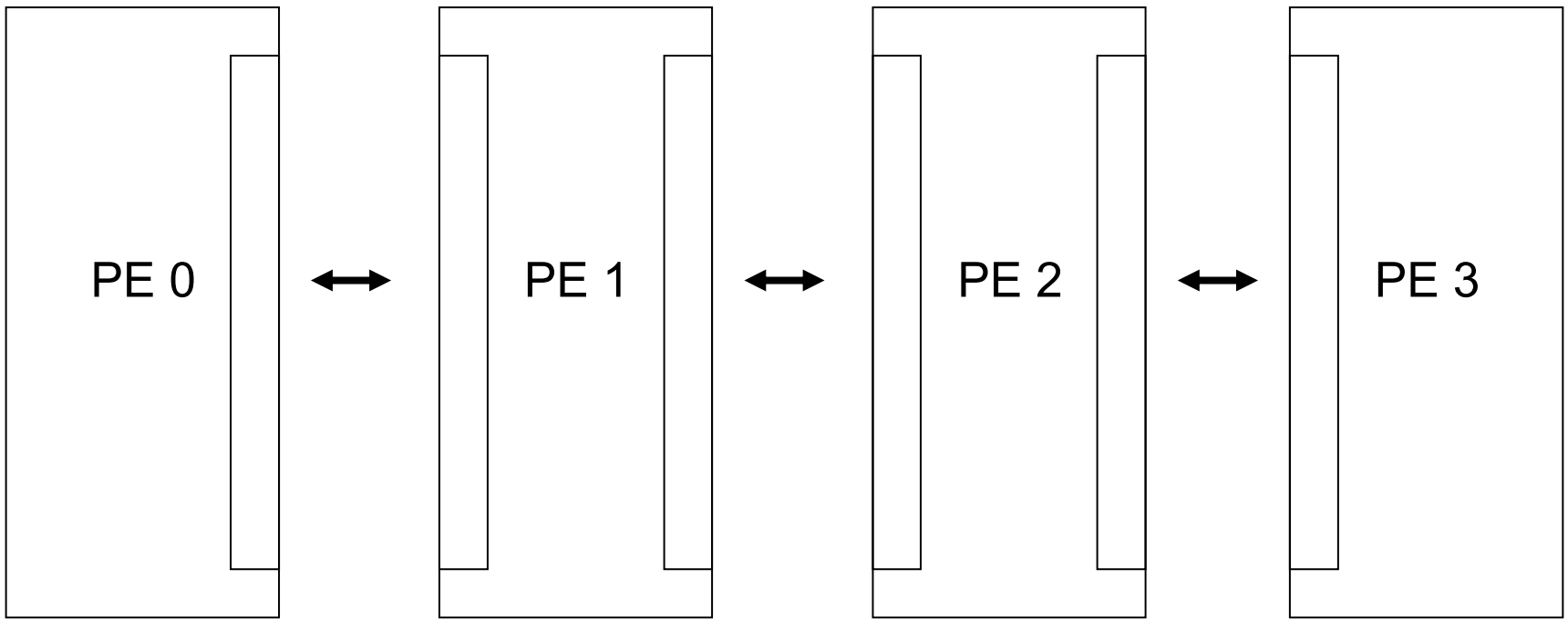
Sample Parallelization Strategy

Break the grid into groups of columns for Fortran, rows for C.



The outer rectangles accommodate boundary and “ghost” zones.

Exchange Edge Data at Each Iteration



One Solution

```
if (mod(mype,2).eq.1) then                    !  mype is odd
  call MPI_SEND(T(1,1)  ,nr,MPI_DOUBLE_PRECISION,mype-1,tag,    &
                MPI_COMM_WORLD,ierr)
 if ( mype .lt. npes-1) then
  call MPI_SEND(T(1,ncl),nr,MPI_DOUBLE_PRECISION,mype+1,tag,    &
                MPI_COMM_WORLD,ierr)
 endif
  call MPI_RECV(T(1,0)  ,nr,MPI_DOUBLE_PRECISION,mype-1,tag,    &
                MPI_COMM_WORLD,status,ierr)
 if ( mype .lt. npes-1) then
  call MPI_RECV(T(1,ncl+1),nr,MPI_DOUBLE_PRECISION,mype+1,tag,  &
                MPI_COMM_WORLD,status,ierr)
 endif
```

```
    else                                    !  mype is even
        if ( mype .gt. 0) then
          call MPI_RECV(T(1,0)  ,nr,MPI_DOUBLE_PRECISION,mype-1,tag,    &
                                  MPI_COMM_WORLD,status,ierr)
        endif
        if ( mype .lt. npes-1) then
          call MPI_RECV(T(1,ncl),nr,MPI_DOUBLE_PRECISION,mype+1,tag,    &
                                  MPI_COMM_WORLD,status,ierr)
        endif
        if ( mype .gt. 0) then
          call MPI_SEND(T(1,1)  ,nr,MPI_DOUBLE_PRECISION,mype-1,tag,    &
                                  MPI_COMM_WORLD,status,ierr)
        endif
        if ( mype .lt. npes-1) then
          call MPI_SEND(T(1,ncl),nr,MPI_DOUBLE_PRECISION,mype+1,tag,    &
                                  MPI_COMM_WORLD,status,ierr)
        endif
      endif
```

Another Solution

- This pattern of sends and receives is sufficiently common that there is a subroutine
  - MPI_SENDRECV(sendbuf,count,mpi_type,dest,
    - tag,recbuf,count,mpi_type,source,tag,comm,status, err)
- Sends and receives can go to the NULL process MPI_PROC_NULL
  - This can simplify code

Nonblocking Sends and Receives

- Beyond the scope of this brief talk
- Allow overlap of computation and communication
- Begin with the character **I (**ISEND**,** IRECV**)**
- Completed with IWAIT

EXERCISE

- On Aspen, copy /lv1/rescomp/mpi_workshop.tar to your home directory.
- Untar it
  - tar xf mpi_workshop.tar
- Cd to trapezoid. Compile and run trap.f90 (serial version of Jacobi iteration code)
  - module add ifc
  - ifc –o trap –cm –w trap.f90
  - ./trap > trap.out

Exercise [cont.]

- Compile partrap.f90
  - module add mpich-eth-intel
  - mpif90 –o partrap –cm –w partrap.f90
- Use partrap.sh to submit and run the job. Use npes=1 and npes=2.
- Compare results.

HOMEWORK

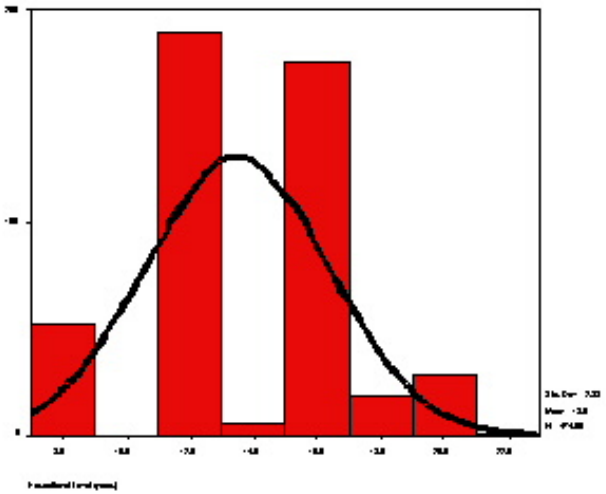
- Examine, compile, and run serjacobi.f90. This is a Jacobi iteration code.
- Your assignment: take jacobi.f90 or jacobi.c and parallelize it. These codes have indications added for adding MPI calls and what they should do.
- Two possible answers for jacobi.f90 are in the subdirectory *solution*. No peeking until you've given it a try yourself!

MPI References

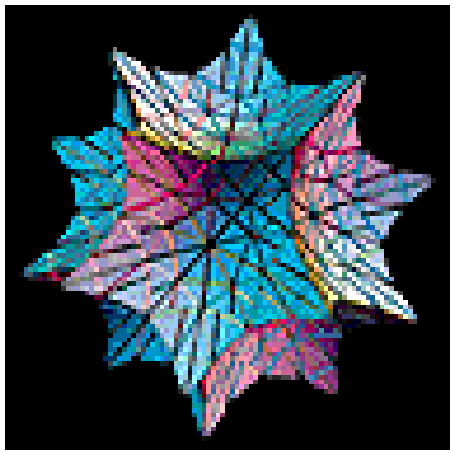
- **Parallel Programming with MPI** by Peter Pacheco.
- **MPI: The Complete Reference, Second Edition** by Snir *et al*. PostScript or PDF versions of the First Edition of this manual are available around the Internet, e.g.
  - http://www.phyast.pitt.edu/beowulf/Tutorial.html
  - The first edition contains some bugs in example code, but is fine as a reference to the subroutine parameter lists.

Web Resources

- MPI Homepage:
  - http://www-unix.mcs.anl.gov/mpi
- Online Tutorial through NCSA:
  - http://pacont.ncsa.uiuc.edu:8900/public/MPI/
- Links to PostScript or PDF versions of User's Guides (C and F77/F90)
  - http://www.phyast.pitt.edu/beowulf/Tutorial.html
- HTML version of the first edition of the **Complete Reference** :
  - http://www.netlib.org/utk/papers/mpi-book/mpi-book.html



Upcoming Talks



Aspen Hands-on Tutorial is online at www.itc.virginia.edu/research/linux-cluster/hands-on

- Talks are online at www.itc.virginia.edu/research/talks

- Beyond Powerpoint: Alternative Software for Scientific/Technical Presentation. Wednesday, March 17

- Minitab for Windows, Version 14: What's New. Wednesday, March 31

- Computing with the IMSL Scientific Libraries. Wednesday, April 14