Parallel Port Example

April 24, 2002

PITTSBURGH
SUPERCOMPUTING
CENTER

Introduction

The objective of this lecture is to go over a simple problem that illustrates the use of the MPI library to parallelize a partial differential equation (PDE).

The Laplace problem is a simple PDE and is found at the core of many applications. More elaborate problems often have the same communication structure that we will discuss in this class. Thus, we will use this example to provide the fundamentals on how communication patterns appear on more complex PDE problems.

This lecture will demonstrate message passing techniques, among them, how to:

- **Distribute Work**
- **Distribute Data**
- **Communication:**
  Since each processor has its own memory, the data is not shared, and communication becomes important.
- **Synchronization**

Laplace Equation

The Laplace equation is:

$$\nabla^2 T = 0 \text{ ; or } \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

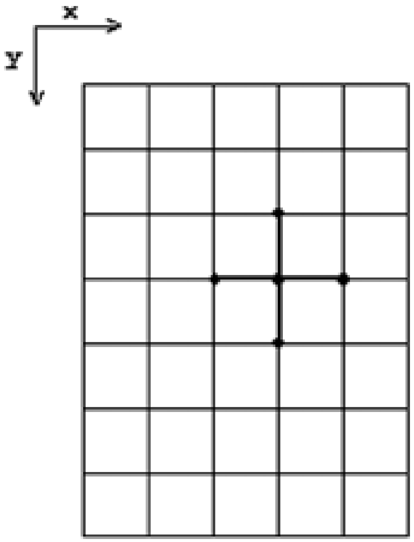
We want to know t(x,y) subject to the following initial boundary conditions:

$$T=0 \quad \left\{ \begin{array}{l} \text{intial values in the interior} \\ \text{at the top boundary} \\ \text{at the left boundary} \end{array} \right.$$

$$\text{T varies linearly from 0 to 100} \quad \left\{ \begin{array}{l} \text{along the right boundary} \\ \text{along the bottom boundary} \end{array} \right.$$

Laplace Equation

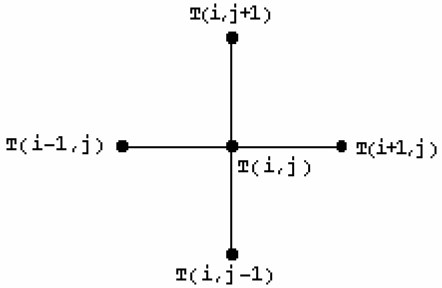
To find an approximate solution to the equation, define a square mesh or grid consisting of points X_i, Y_j.



The Point Jacobi Iteration

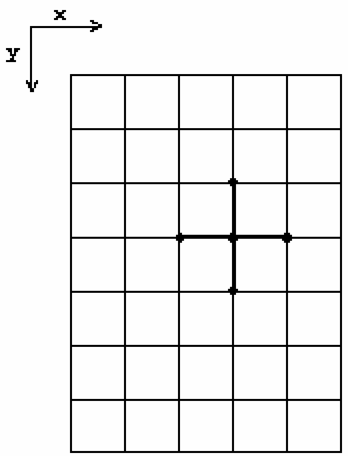
The method known as “point Jacobi iteration” calculates the value if T9i,j) as an average of the old values of T at the neighboring points:

```
T(I,J)=0.25*(Told(I-1,J)
            +Told(I+1,J)
            +Told(I,J-1)
            +Told(I,J+1))
```



The Point Jacobi Iteration

The iteration is repeated until the solution is reached.



If we want to solve T for [1000, 1000] points, the grid itself needs to be of dimension 1002 x 1002; since the algorithm to calculate T9i,j) requires values of T at I-1, I+1, j-1, and j+1.

Serial Code Implementation

In the following NR=numbers of rows, NC= number of columns. (excluding the boundary columns and rows)

The serial implementation of the Jacobi iteration is:

Fortran:

```
    DO J = 1,NC
       DO I = 1,NR
          T(I,J) = 0.25*(Told(I-1,J)
.                       +Told(I+1,J)
.                       +Told(I,J-1)
.                       +Told(I,J+1))
       ENDDO
    ENDDO
```

C:

```
    for (i=1; i <= NR; i++)
      for (j=1; j <= NC; j++)
       T[i][j] = 0.25*(Told[i+1][j]
                      +Told[i-1][j]
                      +Told[i][j+1]
                      +Told[i][j-1]);
```

Serial Version – C

```
/*
 *****************************************************************************
 * Laplace Eqn
 * T is initially 0.0
 * Boundaries are as follows
 *
 *             T = 0                                           Y
 *      |-------------------| 0       |-------------------|    |
 *      |                   |         |                   |    |
 *   T  |                   |         |-------------------|    |
 *   =  |                   |  T      |                   |    |
 *   0  |                   |         |-------------------|    |________X
 *      |                   |         |                   |
 *      |                   |         |-------------------|
 *      |                   |         |                   |
 *      |-------------------| 100     |-------------------|
 *      0                 100
 *
 * Use Central Differencing Method
 * Each process only has subgrid.
 * Each Processor works on a sub grid and then sends its
 * Boundaries to neighbours
 ****************************************************************   */

#define NPROC    1
#define NR       1000
#define NC       1000
#define NRL      (NR/NPROC)
#define NITER    1000

void initialize( float t[NRL+2][NC+2] );
void set_bcs   ( float t[NRL+2][NC+2] );
void print_trace( float t[NRL+2][NC+2], int mype, int npes, int iter );

#include

main(int argc, char **argv) {

  float t[NRL+2][NC+2], told[NRL+2][NC+2];
  int   i, j, iter, niter;

  initialize( t );

  set_bcs( t );

  for( i=0; i<=NRL+1; i++ )
    for( j=0; j<=NC+1; j++ )
      told[i][j] = t[i][j];

  fprintf(stderr, "How many iterations [100-1000]? ");
  scanf("%d", &niter);
  if( niter>NITER ) niter = NITER;
```

Serial Version – C

```
/*    Do Computation on Sub-grid for Niter iterations     */

  for( iter=1; iter<=niter; iter++ ) {

    for( i=1; i<=NRL; i++ )
      for( j=1; j<=NC; j++ )
       t[i][j] = 0.25 * ( told[i+1][j] + told[i-1][j] +
                          told[i][j+1] + told[i][j-1] );

/*   Copy for next iteration  */

    for( i=1; i<=NRL; i++ )
      for( j=1; j<=NC; j++ )
       told[i][j] = t[i][j];


/*   Print some test Values   */

    if( (iter%100) == 0 ) {
      print_trace( t, 0, 1, iter );
    }
  }  /* End of iteration *//

}    /* End of Program */

/*******************************************************************
 *                                                           *
 * Initialize all the values to 0. as a starting value       *
 *                                                           *
 *******************************************************************/


void initialize( float t[NRL+2][NC+2] ){

  int        i, j, iter;

  for( i=0; i<=NRL+1; i++ )        /* Initialize */
    for ( j=0; j<=NC+1; j++ )
      t[i][j] = 0.0;
}

/*******************************************************************
 *                                                           *
 * Set the values at the boundary.  Values at the boundary do not   *
 * Change through out the execution of the program           *
 *                                                           *
 *******************************************************************/

void set_bcs( float t[NRL+2][NC+2] ){

  int i, j;

  for( i=0; i<=NRL+1; i++ ) {      /* Set Left and Right bndry */
    t[i][0]       = 0.0;
    t[i][NC+1]    = (100.0/NRL) * i;
  }

  for( j=0; j<=NC+1; j++ ){        /* Set top and Bottom bndry */
    t[0][j]         = 0.0;
    t[NRL+1][j]     = (100.0/NC) * j;
  }
```

Serial Version – C

```
void set_bcs( float t[NRL+2][NC+2] ){

  int i, j;

  for( i=0; i<=NRL+1; i++ ) {        /* Set Left and Right bndry */
    t[i][0]        = 0.0;
    t[i][NC+1]     = (100.0/NRL) * i;
  }

  for( j=0; j<=NC+1; j++ ){          /* Set top and Bottom bndry */
    t[0][j]         = 0.0;
    t[NRL+1][j]     = (100.0/NC) * j;
  }

}
/*********************************************************************
 *                                                             *
 * Print the trace only in the last PE where most action is           *
 *                                                             *
 *********************************************************************/
void print_trace( float t[NRL+2][NC+2], int mype, int npes, int iter ){

  int joff, i;

  if( mype==npes-1 ){
    printf("\n---------- Iteration number: %d ------------\n", iter);

    joff = mype*NRL;
    for(i=NRL-10; i<=NRL; i++){
      printf("%15.8f", t[i][joff+i]);
    }
  }
  barrier();
  return;
}
```

Serial Version - Fortran

```
*************************************************************
* Laplace Eqn
* T is initially 0.0
* Boundaries are as indicated below
*
*          T=0.
*       __________                       ____|____|____|____     Y
*       |        | 0                 |     |    |    |    |   |
*       |        |                   |     |    |    |    |   |
* T=0.  | T=0.0  | T                  | 1  | 2  | 3  | 4  |   |
*       |        |                   |     |    |    |    |   |
*       |________| 100                |    |    |    |    |   |________X
*        0      100                   |____|____|____|____|
*                                          |    |    |
* Use Central Differencing Method
* Each process only has subgrid.
* Each Processor works on a sub grid and then sends its
* Boundaries to neighbours
*************************************************************

      program serial
      implicit none

      integer    NPROC,   NR,      NC,      NCL,           MXITER
      parameter (NPROC=1, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1)
      integer    i, j, iter, niter
```


PITTSBURGH SUPERCOMPUTING CENTER

Serial Version - Fortran

```
      real*8      t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1)
      integer     i, j, iter, niter

      call initialize( t )
      call set_bcs( t )

      do i=0, NR+1
         do j=0, NCL+1
            told(i,j) = t(i,j)
         enddo
      enddo

      print*, 'How many iterations [100-1000]?'
      read*,    niter

      if( niter.gt.MXITER ) niter = MXITER
*
* Do Computation on Sub-grid for Niter iterations
*
      Do 100 iter=1,niter

         Do j=1,NCL
            Do i=1,NR
               T(i,j) = 0.25 * ( Told(i+1,j)+Told(i-1,j)+
     $                           Told(i,j+1)+Told(i,j-1) )
            Enddo
         Enddo
*
* Copy
*
```

Serial Version - Fortran

```
         Do j=1,NCL
            Do i=1,NR
               Told(i,j) = T(i,j)
            Enddo
         Enddo
*
* Print some Values
*
         If( mod(iter,100).eq.0 ) then
            call print_trace(t, 0, 1, iter)
         endif
*
* Go to Next time step
*
 100  CONTINUE
*
* End of Program!
*
      END
*-----------------------------------------------------------------------*
      subroutine initialize( t )
      implicit none

      integer    NPROC,   NR,      NC,      NCL,           MXITER
      parameter (NPROC=1, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1)
      integer    i, j

      do i=0, NR+1
         do j=0, NCL+1
            t(i,j) = 0
         enddo
      enddo

      return
```

Serial Version - Fortran

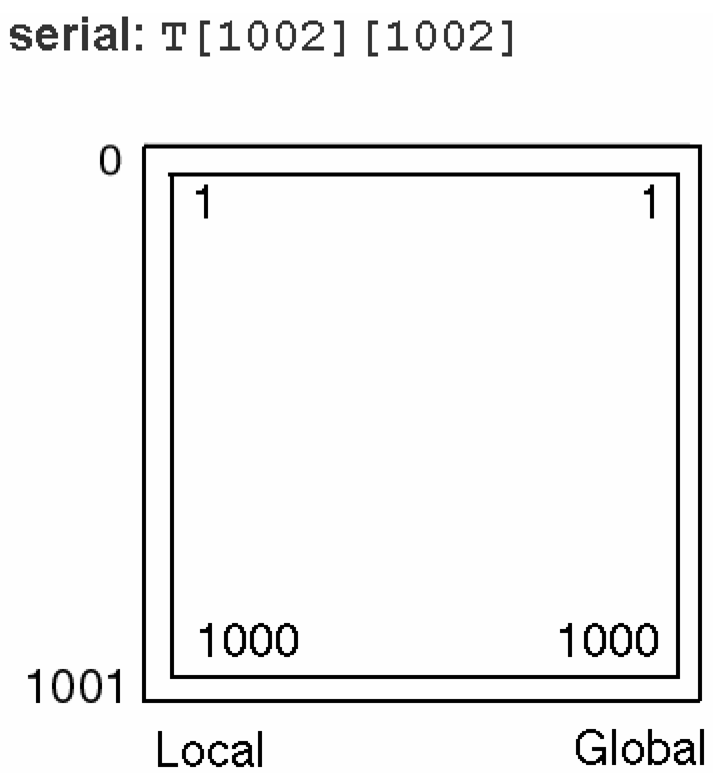
```
      end
*----------------------------------------------------------------------*
      subroutine set_bcs( t )
      implicit none

      integer    NPROC,   NR,       NC,       NCL,           MXITER
      parameter (NPROC=1, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1)
      integer    i, j
*
*Left and Right Boundaries
*
      do i=0,NR+1
         T(i,0    ) = 0.0
         T(i,NCL+1) = (100.0/NR) * i
      enddo
*
*Top and Bottom Boundaries
*
      do j=0,NCL+1
         T(0    ,j) = 0.0
         T(NR+1,j) = (100.0/NCL) * j
      enddo

      return

      end
*----------------------------------------------------------------------*
      subroutine print_trace(t, mype, npes, iter)
      implicit none

      integer    NPROC,   NR,       NC,       NCL,           MXITER
      parameter (NPROC=1, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1)
      integer    ioff, j, k, proc, mype, npes, iter
```

Serial Version - Fortran

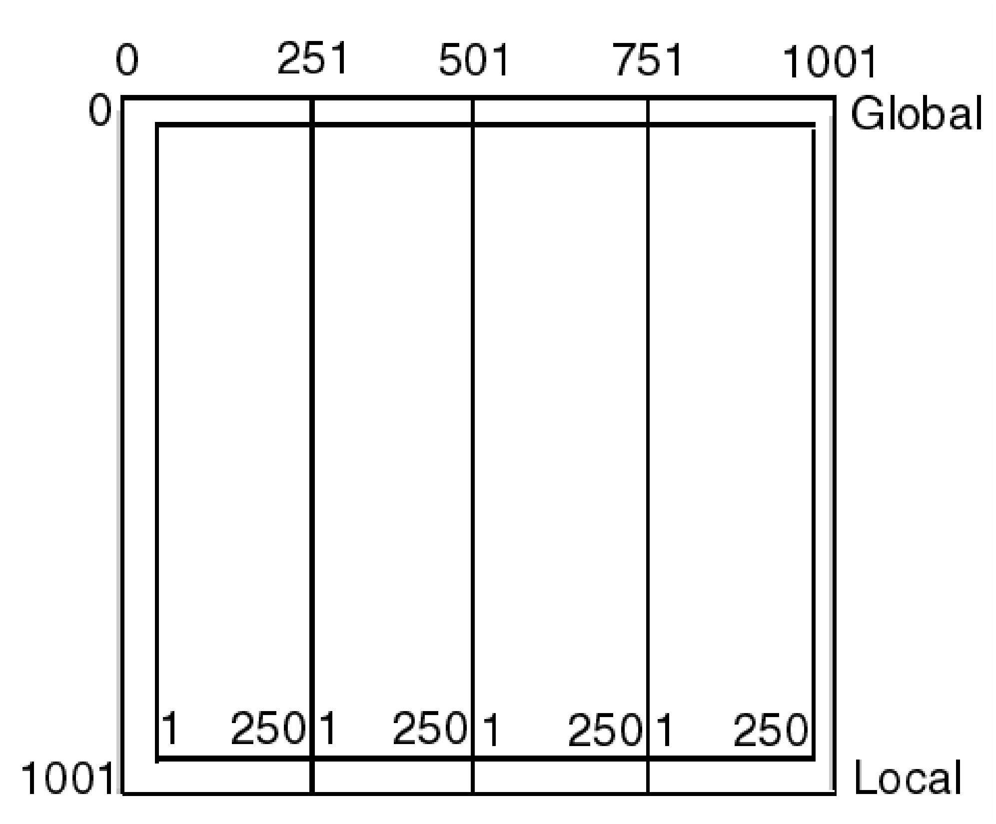
```
      if( mype.eq.npes-1 ) then
         write(6,1)iter
         ioff = mype*NCL
         write(6,3)(t(ioff+k,k), k=NCL-10 ,NCL)
      endif
      call barrier

1     format('---------- Iteration number: ', i10, '---------------')
3     format(5f15.8)
      return
      end
```

Parallel Version: Example Using 4 Processors

Recall that in the serial case the grid boundaries were:



Simplest Decomposition for Fortran Code

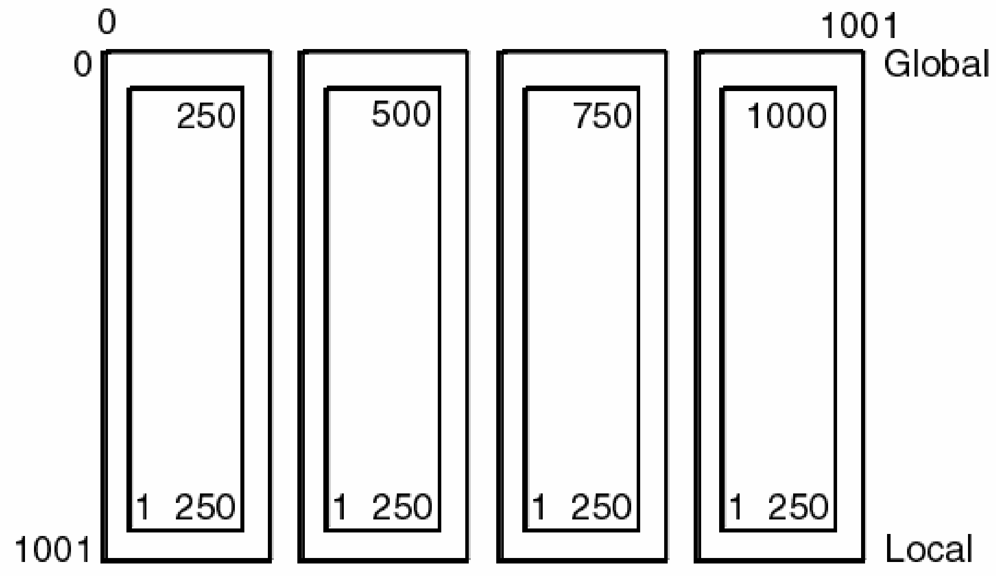


April 24, 2002

PITTSBURGH SUPERCOMPUTING CENTER

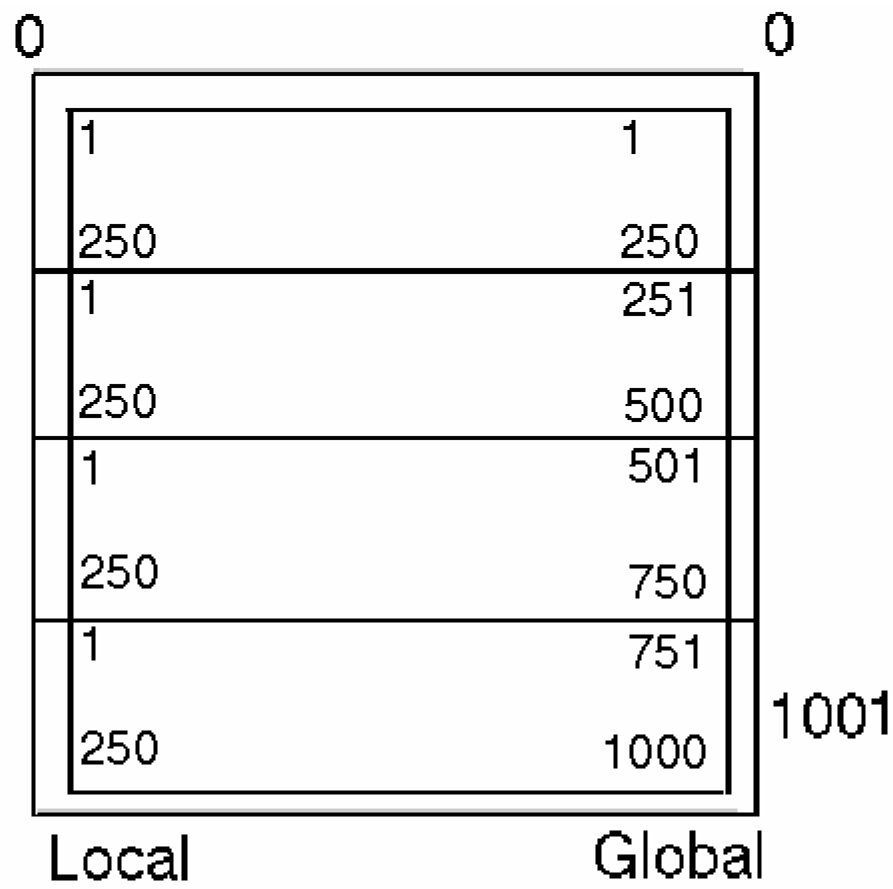
Simplest Decomposition for Fortran Code

A better distribution from the point of view of communication optimization is the following:



The program has a "local" view of data.
The programmer has to have a "global" view of data.

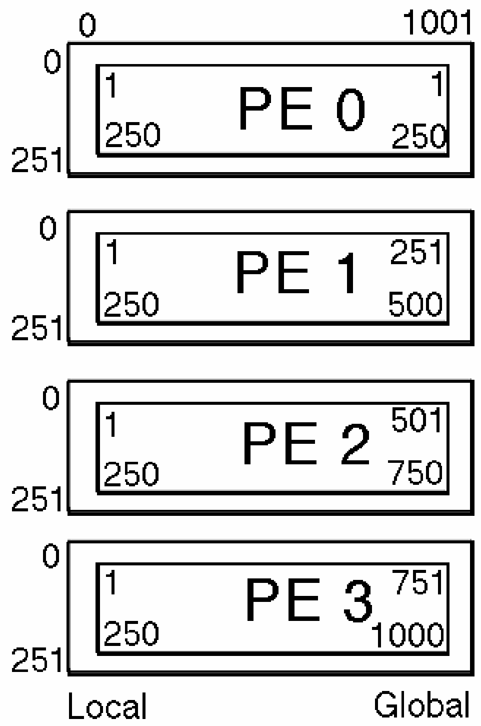
Simplest Decomposition for C Code



PITTSBURGH SUPERCOMPUTING CENTER

Simplest Decomposition for C Code

In the parallel case, we will break this up into 4 processors:
There is only one set of boundary values. But when we distribute the data, each processor needs to have an extra row for data distribution:



The program has a “local” view of data. The programmer has to have a “global” view of data.

Include Files

Fortran:

```
* (always declare all variables)
    implicit none
    INCLUDE 'mpif.h'

* Initialization and clean up (always check error codes):
    call MPI_Init(ierr)
    call MPI_Finalize(ierr)
```

C:

```
    #include "mpi.h"
/* Initialization and clean up (always check error codes): */

    stat = MPI_Init(&argc, &argv);
    stat = MPI_Finalize();
```

Note: Check for MPI_SUCCESS

```
    if (ierr. ne. MPI_SUCCESS) then
    do error processing
    endif
```

Initialization

Serial version:

In Fortran:

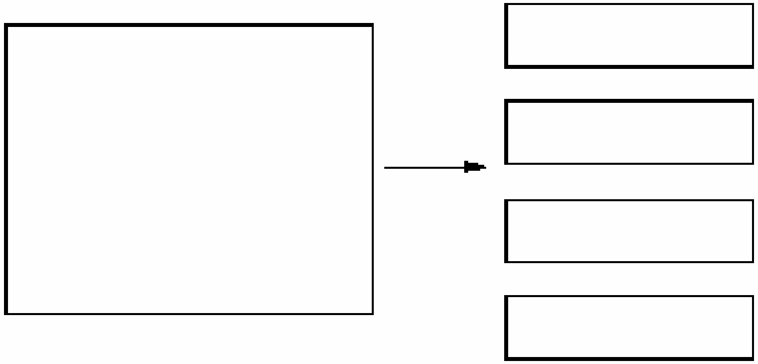
```fortran
do I = 0, NR + 1
   do J = 0, NC + 1
      T(I,J) = 0.0
   enddo
enddo
```

In C:

```c
for (i=0; i <= NR + 1; i++)
  for (j=0; j <= NC +1; j++)
    T[i][j] = 0.0;
```

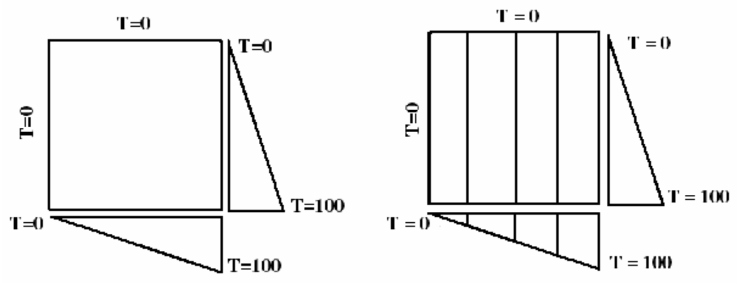
Parallel version:

Just for simplicity, we will distribute rows in C and columns in Fortran; this is easier because data is stored in rows C and in columns Fortran.



Parallel Version: Boundary Conditions

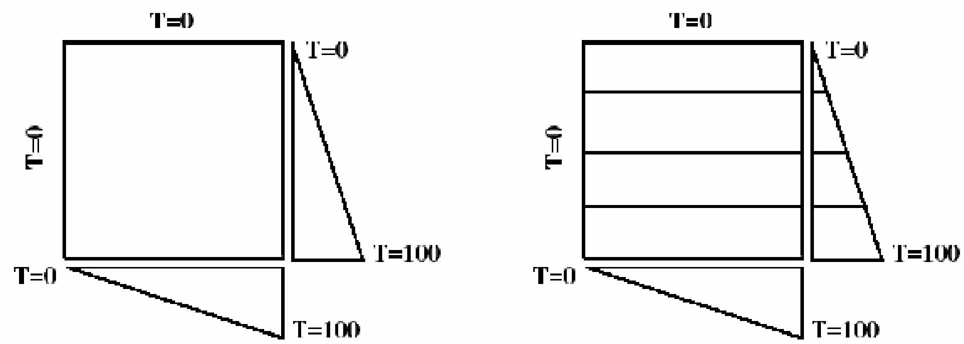
Fortran Version



We need to know MYPE number and how many PEs we are using.
Each processor will work on different data depending on MYPE.
Here are the boundary conditions in the serial code, where
NRL-local number of rows, NRL=NPROC

```
      subroutine set_bcs( t, mype, npes )
      integer    i, j, mype, npes
*
*Left and Right Boundaries
*
      if( mype.eq.0 ) then
         do i=0,NR+1
            T(i,0    ) = 0.0
         enddo
      endif

      if( mype.eq.npes-1 ) then
         do i=0,NR+1
            T(i,NCL+1) = >>>>>>>>>
         enddo
      endif
*
*Top and Bottom Boundaries
*
      tmin =  mype    * 100.0/npes
      tmax = (mype+1) * 100.0/npes
      do j=0,NCL+1
         T(0   ,j) = 0.0
         T(NR+1,j) = >>>>>>>>>>>>
      enddo

      return

      end
```

Parallel C Version: Boundary Conditions



We need to know MYPE number and how many PEs we are using. Each processor will work on different data depending on MYPE.
Here are the boundary conditions in the serial code, where
NRL=local number of rows, NRL=NR/NPROC

```
void set_bcs( float t[NRL+2][NC+2] ){

  int i, j;

  for( i=0; i<=NRL+1; i++ ) {      /* Set Left and Right bndry */
    t[i][0]       = 0.0;
    t[i][NC+1]    = >>>>>>>>;
  }

  for( j=0; j<=NC+1; j++ ){        /* Set top and Bottom bndry */
    t[0][j]        = 0.0;
    t[NRL+1][j]    = >>>>>>>>;
  }

}
```

Processor Information

Fortran:

Number of processors:

```
call MPI_Comm_size (MPI_COMM_WORLD, npes ierr)
```

Processor Number:

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

C:

Number of processors:

```
stat = MPI_Comm_size(MPI_COMM_WORLD, &npes);
```

Processor Number:

```
stat = MPI_Comm_rank(MPI_COMM_WORLD, &mype);
```

Maximum Number of Iterations

Only 1 PE has to do I/O (usually PE0).

Then PE0 (or root PE) will broadcast *niter* to all others. Use the collective operation MPI_Bcast.

Fortran:

```
MPI_Bcast(niter, 1, MPI_INTEGER, PE0, comm, ierr)
                 ^       ^        ^
              number    type    root
                of       of      PE
             elements   data
```

Here *number of elements* is how many values we are passing, in this case only one: *niter*.

C:

```
stat = MPI_Bcast(&niter, 1, MPI_INT, PE0, comm);
```

Main Loop

```
for (iter=1; iter <= NITER; iter++) {
```

Do averaging (each PE averages from 1 to 250)

Copy T into Told

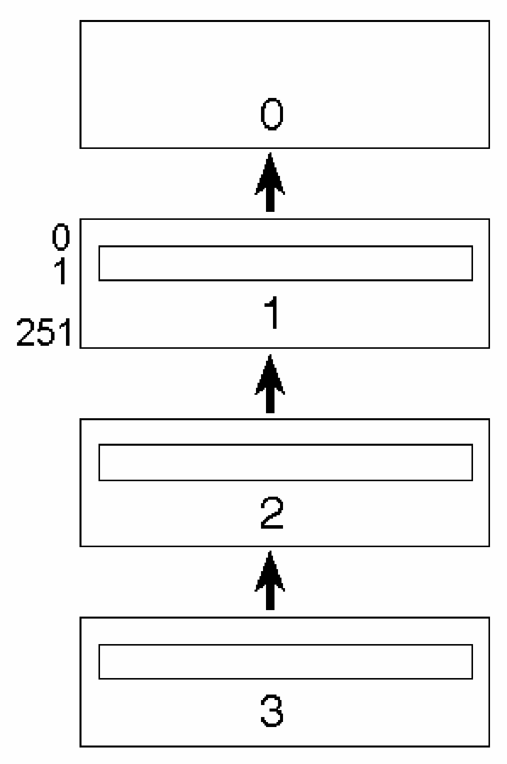
Send Values down	This is where we use MPI communication calls: need to exchange data between processors
Send values up	
Receive values from above	
Receive values from below	
(find the max change)	
Synchronize	
}	

Parallel Template: Send data up

Once the new T values have been calculated:

SEND

- All processors except processor 0 send their “first” row (in C) to their neighbor above (mype – 1).

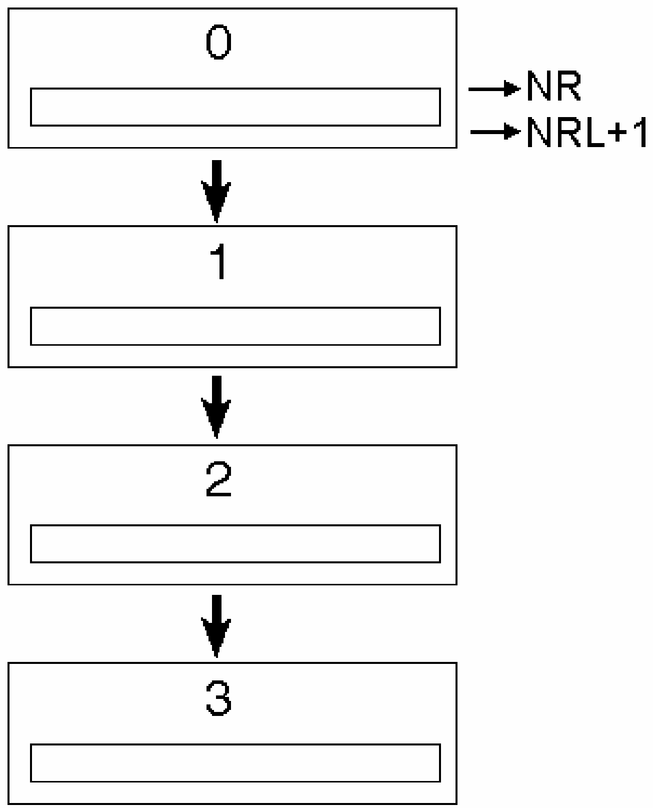


PITTSBURGH SUPERCOMPUTING CENTER

Parallel Template: Send data down

SEND

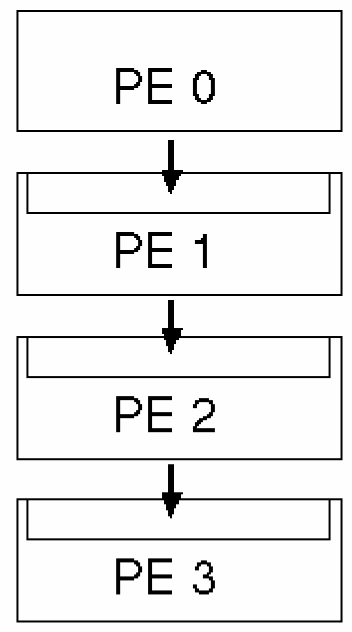
- All processors except the last one, send their “last” row to their neighbor below (mype + 1).



Parallel Template: Receive from above

Receive

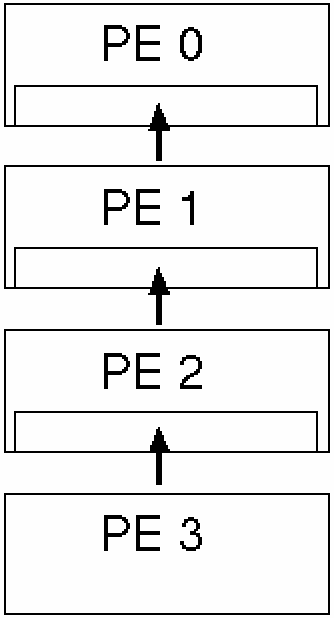
- All processors except PE0, receive from their neighbor above and unpack in row 0.



Parallel Template: Receive from below

Receive

- All processors except processor (NPES-1), receive from the neighbor below and unpack in the last row.



Example: PE1 receives 2 messages – there is no guarantee of the order in which they will be received.

PITTSBURGH SUPERCOMPUTING CENTER

Parallel Template (C)

```
/*
 ********************************************************************
 * Laplace Eqn
 * T is initially 0.0
 * Boundaries are as follows
 *
 *              T = 0                                            Y
 *      |-------------------| 0       |-------------------|   |
 *      |                   |         |                   |   |
 *  T   |                   |         |-------------------|   |
 *  =   |                   |  T      |                   |   |
 *  0   |                   |         |-------------------|   |________X
 *      |                   |         |                   |
 *      |                   |         |-------------------|
 *      |                   |         |                   |
 *      |-------------------| 100     |-------------------|
 *      0                 100
 *
 * Use Central Differencing Method
 * Each process only has subgrid.
 * Each Processor works on a sub grid and then sends its
 * Boundaries to neighbours
 ******************************************************************    */
#define NPES     4
#define NC       1000                    /* Number of Cols        */
#define NR       1000                    /* Number of Rows        */
#define NRL      (NR/NPES)               /* Number of Rows per PE */
#define NITER    1000                    /* Max num of Iterations */
#define DOWN     100                     /* Tag for messages down */
#define UP       101                     /* Tag for messages up   */
#define ROOT     0                       /* The root PE           */
#define MAX(x,y) ( ((x) > (y)) ? x : y )

#include <stdio.h>
#include <math.h>

<<<<<<<<<<<<<<< Include file

void initialize( float t[NRL+2][NC+2] );
void set_bcs    ( float t[NRL+2][NC+2], int mype, int npes );
void print_trace( float t[NRL+2][NC+2], int mype, int npes, int iter );
```

Parallel Template (C)

```
int main( int argc, char **argv ){

  int         npes;                          /* Number of PEs */
  int         mype;                          /* My PE number  */
  int         stat;                          /* Error Status  */
  int         niter;                         /* iter counter  */

<<<<<<<<<<<<<< Declaration for status of MPI_Recv

  float       t[NRL+2][NC+2], told[NRL+2][NC+2];
  float       dt;                            /* Delta t        */
  float       dtg;                           /* Delta t global*/

  int         i, j, iter;

<<<<<<<<<<<<<<<<< Initialize MPI

<<<<<<<<<<<<<<<<< Find npes
<<<<<<<<<<<<<<<<< Find mype

  if ( npes != NPES ){                       /* Currently hardcoded */
    MPI_Finalize();
    if( mype == 0 )
      fprintf(stderr, "The example is only for %d PEs\n", NPES);
    exit(1);
  }

  initialize(t);                       /* Give initial guess of 0. */

  set_bcs(t, mype, npes);              /* Set the Boundary values   */

  for( i=0; i<=NRL+1; i++ )            /* Copy the values into told */
    for( j=0; j<=NC+1; j++ )
      told[i][j] = t[i][j];
```

Parallel Template (C)

```
/*----------------------------------------------------------------*
 |       Do Computation on Sub-grid for Niter iterations           |
 *----------------------------------------------------------------*/

<<<<<<<<<<<<<<<<<<< Get the Maximum number of interations from user

<<<<<<<<<<<<<<<<<<< Broadcast the value of niter to all PEs

  for( iter=1; iter<=niter; iter++ ) {

    for( i=1; i<=NRL; i++ )
      for( j=1; j<=NC; j++ )
        t[i][j] = 0.25 * ( told[i+1][j] + told[i-1][j] +
                           told[i][j+1] + told[i][j-1] );
    dt = 0.;

    for( i=1; i<=NRL; i++ )        /* Copy for next iteration  */
      for( j=1; j<=NC; j++ ){
        dt          = MAX( fabs(t[i][j]-told[i][j]), dt);
        told[i][j] = t[i][j];
      }

<<<<<<<<<<<<<<<<<<< Exchange values

<<<<<<<<<<<<<<<<<<< Find max dt over the whole domain

/*   Print some test Values    */

    if( (iter%100) == 0 ) {
      print_trace( t, mype, npes, iter );
    }

<<<<<<<<<<<<<<<<<< Synchronize

  }  /* End of iteration */

<<<<<<<<<<<<<<<<<<< Clean up!

}    /* End of Program   */
```

Parallel Template (C)

```
/********************************************************************
 *                                                                  *
 * Initialize all the values to 0. as a starting value              *
 *                                                                  *
 ********************************************************************/

void initialize( float t[NRL+2][NC+2] ){

  int        i, j, iter;

  for( i=0; i<=NRL+1; i++ )        /* Initialize */
    for ( j=0; j<=NC+1; j++ )
      t[i][j] = 0.0;
}

/********************************************************************
 *                                                                  *
 * Set the values at the boundary.  Values at the boundary do not   *
 * Change through out the execution of the program                  *
 *                                                                  *
 ********************************************************************/

void set_bcs( float t[NRL+2][NC+2], int mype, int npes ){

  int i, j;

<<<<<<<<<<<<<< Set the Boundar values in all the PEs

}

/********************************************************************
 *                                                           *
 * Print the trace only in the last PE where most action is         *
 *                                                           *
 ********************************************************************/
void print_trace( float t[NRL+2][NC+2], int mype, int npes, int iter ){

  int joff, i;
```

Parallel Template (C)

```
  if( mype==npes-1 ){
    printf("\n---------- Iteration number: %d ------------\n", iter);

    joff = mype*NRL;
    for(i=NRL-10; i<=NRL; i++){
      printf("%15.8f", t[i][joff+i]);
    }
  }
  barrier();
  return;
}
```

Parallel Template (Fortran)

```
***********************************************************
* Laplace Eqn
* T is initially 0.0
* Boundaries are as indicated below
*
*          T=0.
*       _________                ____|____|____|____    Y
*       |       | 0             |    |    |    |    |   |
*       |       |               |    |    |    |    |   |
* T=0.  | T=0.0 | T              | 1  | 2  | 3  | 4  |   |
*       |       |               |    |    |    |    |   |
*       |_______| 100            |    |    |    |    |   |________X
*        0     100                |____|____|____|____|
*                                      |    |    |
* Use Central Differencing Method
* Each process only has subgrid.
* Each Processor works on a sub grid and then sends its
* Boundaries to neighbours
***********************************************************
      program serial
      implicit none

      integer    NPROC,   NR,      NC,      NCL,          MXITER
      parameter (NPROC=4, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      integer    LEFT,     RIGHT,     ROOT
      parameter (LEFT=100, RIGHT=101, ROOT=0)

<<<<<<<<<<<<<<<<<< include file

      real*8     t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1), dt, dtg
      integer    i, j, iter, niter, mype, npes, ierr

<<<<<<<<<<<<<<<<<< Declaration for status for MPI_Recv

<<<<<<<<<<<<<<<<<< Initialize MPI

<<<<<<<<<<<<<<<<<< Find npes
<<<<<<<<<<<<<<<<<< Find mype number
```

Parallel Template (Fortran)

```
      call initialize( t )
      call set_bcs( t, mype, npes )

      do i=0, NR+1
         do j=0, NCL+1
            told(i,j) = t(i,j)
         enddo
      enddo
<<<<<<<<<<<<<<<<<<< Get the Max number of iterations from user
<<<<<<<<<<<<<<<<<<< Broadcast it to all the PEs
*
* Do Computation on Sub-grid for Niter iterations
*
      Do 100 iter=1,niter

         Do j=1,NCL
            Do i=1,NR
               T(i,j) = 0.25 * ( Told(i+1,j)+Told(i-1,j)+
     $                           Told(i,j+1)+Told(i,j-1) )
            Enddo
         Enddo
*
* Copy
*
         dt = 0
         Do j=1,NCL
            Do i=1,NR
               dt        = max( abs(t(i,j) - told(i,j)), dt )
               Told(i,j) = T(i,j)
            Enddo
         Enddo

<<<<<<<<<<<<<<<<<<< Exchange boundary values
```

Parallel Template (Fortran)

```
<<<<<<<<<<<<<<<<<<<< Find max of dt in all the processors
*
* Print some Values
*
          If( mod(iter,100).eq.0 ) then
             call print_trace(t, mype, npes, iter)
          endif

          call MPI_Barrier( MPI_COMM_WORLD, ierr )
*
* Go to Next time step
*
 100  CONTINUE
<<<<<<<<<<<<<<<<<<<<<< Clean up!
*
* End of Program!
*
      END
*-----------------------------------------------------------------------*
      subroutine initialize( t )
      implicit none

      integer    NPROC,   NR,       NC,       NCL,           MXITER
      parameter (NPROC=4, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1)
      integer    i, j

      do i=0, NR+1
         do j=0, NCL+1
            t(i,j) = 0
         enddo
      enddo
```

Parallel Template (Fortran)

```
      return

      end
*-----------------------------------------------------------------------*
      subroutine set_bcs( t, mype, npes )
      implicit none

      integer    NPROC,   NR,      NC,      NCL,           MXITER
      parameter (NPROC=4, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1)
      integer    i, j, mype, npes
*
*Left and Right Boundaries
*
<<<<<<<<<<<<<<<<< Set the left and right boundary values
*
*Top and Bottom Boundaries
*
<<<<<<<<<<<<<<<< Set the top and bottom boundary values
      return

      end
*--------------------------------------------------------------------*
      subroutine print_trace(t, mype, npes, iter)
      implicit none

      integer    NPROC,   NR,      NC,      NCL,           MXITER
      parameter (NPROC=4, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1)
      integer    ioff, j, k, proc, mype, npes, iter
```

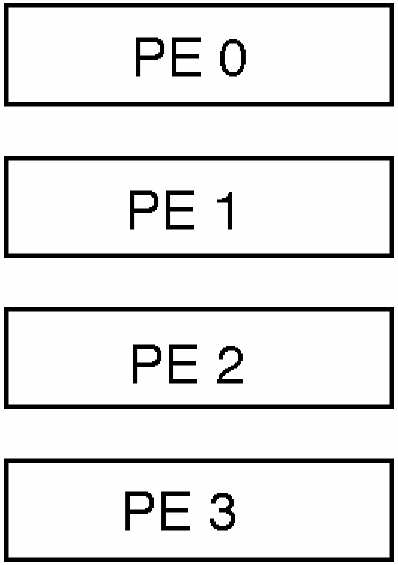
Parallel Template (Fortran)

```
*
*Left and Right Boundaries
*
<<<<<<<<<<<<<<<<< Set the left and right boundary values
*
*Top and Bottom Boundaries
*
<<<<<<<<<<<<<<<< Set the top and bottom boundary values
      return

      end
*----------------------------------------------------------------------*
      subroutine print_trace(t, mype, npes, iter)
      implicit none

      integer    NPROC,   NR,      NC,      NCL,           MXITER
      parameter (NPROC=4, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1)
      integer    ioff, j, k, proc, mype, npes, iter

      if( mype.eq.npes-1 ) then
         write(6,1)iter
         ioff = mype*NCL
         write(6,3)(t(ioff+k,k), k=NCL-10 ,NCL)
      endif
      call barrier

 1    format('---------- Iteration number: ', i10, '---------------')
 3    format(5f15.8)
      return
      end
```


PITTSBURGH
SUPERCOMPUTING
CENTER

Variations

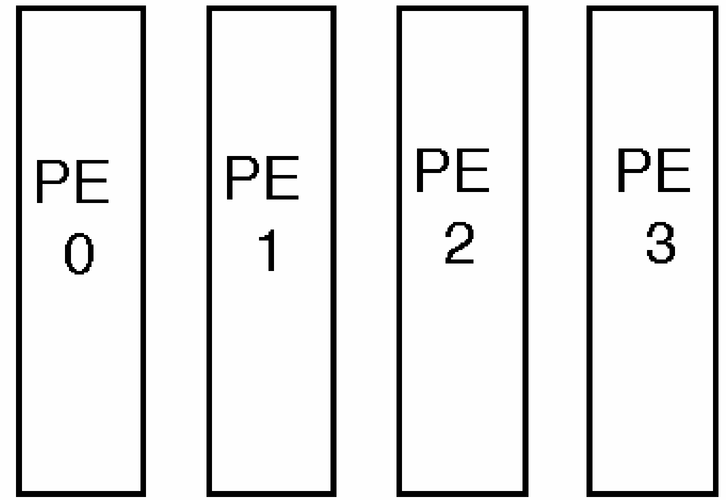
```
if ( mype != 0 ){
   up = mype - 1
   MPI_Send( t, NC, MPI_FLOAT, up, UP_TAG, comm, ierr
   ); }
```



Alternatively

```
up = mype - 1
if ( mype == 0 ) up = MPI_PROC_NULL;
MPI_Send( t, NC, MPI_FLOAT, up, UP_TAG, comm,ierr );
```

Variations




```
if( mype.ne.0 ) then
   left = mype - 1
   call MPI_Send( t, NC, MPI_REAL, left, L_TAG, comm, ierr)
endif
```

Alternatively

```
left = mype - 1
if( mype.eq.0 ) left = MPI_PROC_NULL
call MPI_Send( t, NC, MPI_REAL, left, L_TAG, comm, ierr)
endif
```

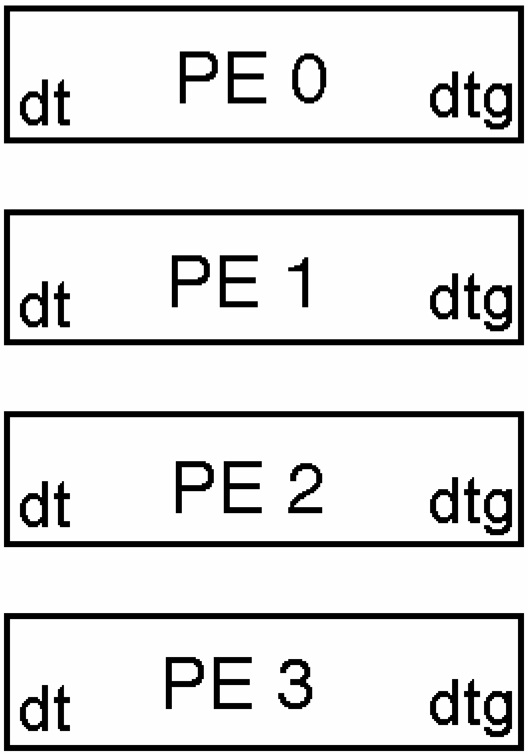
Note: You may also `MPI_Recv from MPI_PROC_NULL`

Variations

Send and receive at the same time:

```
MPI_Sendrecv( … )
```

Finding Maximum Change



Each PE can find it's own maximum change *dt*

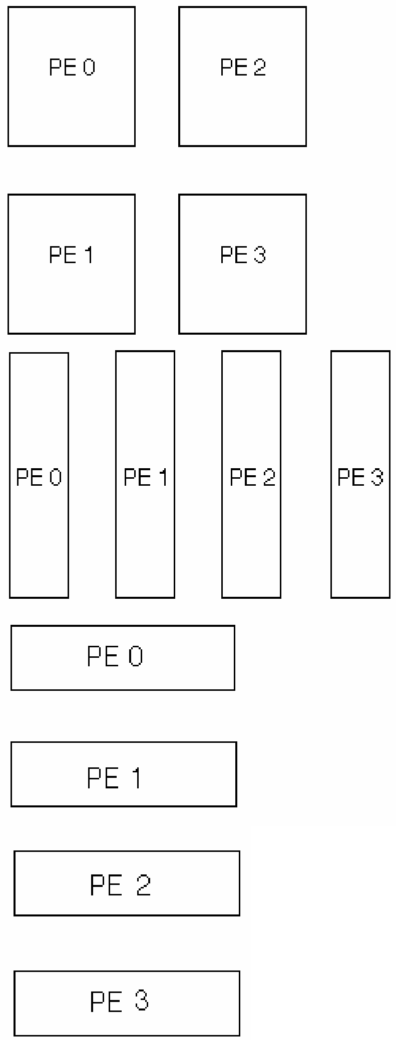
To find the global change *dtg* in C::

```
        MPI_Reduce(&dt, & dtg, 1, MPI_FLOAT,
MPI_MAX, PE0, comm);
```

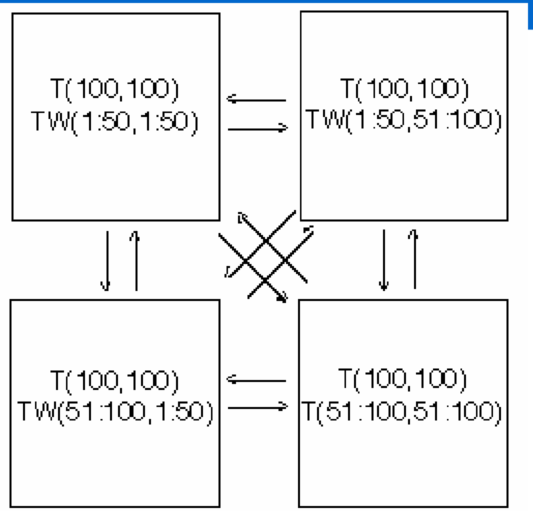
To find the global change *dtg* in Fortran:

```
        call
MPI_Reduce(dt,dtg,1,MPI_REAL,MPI_MAX, PE0,
comm, ierr)
```

Domain Decomposition

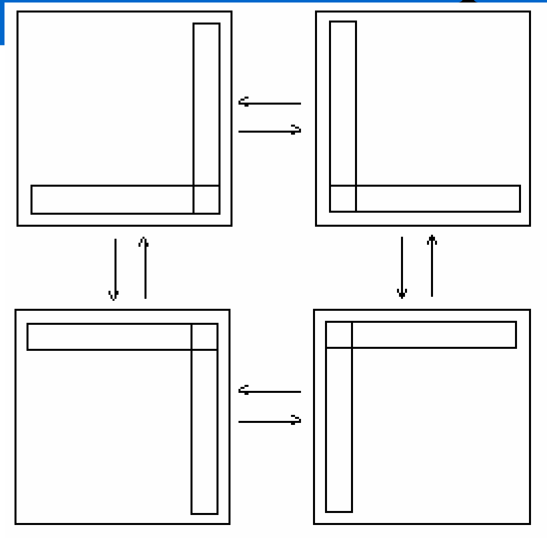


Data Distribution I
Domain Decomposition I



- All processors have entire T array.
- Each processor works on TW part of T.
- After every iteration, all processors broadcast their TW to all other processors.
- Increased memory.
- Increased operations.

Data Distribution I
Domain Decomposition II



- Each processor has sub-grid.
- Communicate boundary values only.
- Reduce memory.
- Reduce communications.
- Have to keep track of neighbors in two directions.

Exercise

1. Copy the following parallel templates into your /tmp directory in jaromir:

```
/tmp/training/laplace/laplace.t3e.c
/tmp/training/laplace/laplace.t3e.f
```

2. These are template files; your job is to go into the sections marked "<<<<<<" in the source code and add the necessary statements so that the code will run on 4 PEs.

   Useful Web reference for this exercise:
   To view a list of all MPI calls, with syntax and descriptions, access the Message Passing Interface Standard at:

   http://www-unix.mcs.anl.gov/mpi/www/

3. To compile the program, *after you have modified it,* rename the new programs laplace_mpi_c.c and laplace_mpi_f.f and execute:

```
cc -lmpi laplace_mpi_c
f90 -lmpi laplace_mpi_f
```

Exercise

4. To run:

```
echo 200 | mpprun -n4 ./laplace_mpi_c
echo 200 | mpprun -n 4 ./laplace_mpi_f
```

5. You can check your program against the solutions

   laplace_mpi_c.c and

   laplace_mpi_f.f

Source Codes

The following are the C and Fortran templates that you need to parallelize for the Exercise.

laplace.t3e.c

```
/*
 ******************************************************************
 * Laplace Eqn
 * T is initially 0.0
 * Boundaries are as follows
 *
 *           T = 0                                    Y
 *     |------------------| 0      |------------------|   |
 *     |                  |        |                  |   |
 *  T  |                  |        |------------------|   |
 *  =  |                  |  T     |                  |   |
 *  0  |                  |        |------------------|   |_______X
 *     |                  |        |                  |
 *     |                  |        |------------------|
 *     |                  |        |                  |
 *     |------------------| 100    |------------------|
 *     0                 100
 *
 * Use Central Differencing Method
 * Each process only has subgrid.
 * Each Processor works on a sub grid and then sends its
 * Boundaries to neighbours
 *****************************************************************   */
#define NPES     4
#define NC       1000                    /* Number of Cols        */
#define NR       1000                    /* Number of Rows        */
#define NRL      (NR/NPES)               /* Number of Rows per PE */
#define NITER    1000                    /* Max num of Iterations */
#define DOWN     100                     /* Tag for messages down */
#define UP       101                     /* Tag for messages up   */
#define ROOT     0                       /* The root PE           */
#define MAX(x,y) ( ((x) > (y)) ? x : y )
```

Source Codes

```
#include <stdio.h>
#include <math.h>

<<<<<<<<<<<<<<< Include file

void initialize( float t[NRL+2][NC+2] );
void set_bcs   ( float t[NRL+2][NC+2], int mype, int npes );
void print_trace( float t[NRL+2][NC+2], int mype, int npes, int iter );

int main( int argc, char **argv ){

  int        npes;                        /* Number of PEs */
  int        mype;                        /* My PE number  */
  int        stat;                        /* Error Status  */
  int        niter;                       /* iter counter  */

<<<<<<<<<<<<<<< Declaration for status of MPI_Recv

  float      t[NRL+2][NC+2], told[NRL+2][NC+2];
  float      dt;                          /* Delta t       */
  float      dtg;                         /* Delta t global*/

  int        i, j, iter;

<<<<<<<<<<<<<<<<<< Initialize MPI

<<<<<<<<<<<<<<<<<< Find npes
<<<<<<<<<<<<<<<<<< Find mype

  if ( npes != NPES ){                    /* Currently hardcoded */
    MPI_Finalize();
    if( mype == 0 )
      fprintf(stderr, "The example is only for %d PEs\n", NPES);
    exit(1);
  }
```

Source Codes

```
  initialize(t);                      /* Give initial guess of 0. */

  set_bcs(t, mype, npes);             /* Set the Boundary values   */

  for( i=0; i<=NRL+1; i++ )           /* Copy the values into told */
    for( j=0; j<=NC+1; j++ )
      told[i][j] = t[i][j];

/*-----------------------------------------------------------*
 |       Do Computation on Sub-grid for Niter iterations     |
 *-----------------------------------------------------------*/

<<<<<<<<<<<<<<<<<<< Get the Maximum number of interations from user

<<<<<<<<<<<<<<<<<<< Broadcast the value of niter to all PEs

  for( iter=1; iter<=niter; iter++ ) {

    for( i=1; i<=NRL; i++ )
      for( j=1; j<=NC; j++ )
        t[i][j] = 0.25 * ( told[i+1][j] + told[i-1][j] +
                           told[i][j+1] + told[i][j-1] );
    dt = 0.;

    for( i=1; i<=NRL; i++ )           /* Copy for next iteration  */
      for( j=1; j<=NC; j++ ){
        dt         = MAX( fabs(t[i][j]-told[i][j]), dt);
        told[i][j] = t[i][j];
      }

<<<<<<<<<<<<<<<<<<< Exchange values

<<<<<<<<<<<<<<<<<<< Find max dt over the whole domain

/*   Print some test Values   */
```

Source Codes

```
    if( (iter%100) == 0 ) {
      print_trace( t, mype, npes, iter );
    }

<<<<<<<<<<<<<<<<<< Synchronize

  }  /* End of iteration */

<<<<<<<<<<<<<<<<<<< Clean up!

}    /* End of Program   */

/*********************************************************************
 *                                                                   *
 * Initialize all the values to 0. as a starting value               *
 *                                                                   *
 *********************************************************************/

void initialize( float t[NRL+2][NC+2] ){

  int         i, j, iter;

  for( i=0; i<=NRL+1; i++ )        /* Initialize */
    for ( j=0; j<=NC+1; j++ )
      t[i][j] = 0.0;
}

/*********************************************************************
 *                                                                   *
 * Set the values at the boundary.  Values at the boundary do not    *
 * Change through out the execution of the program                   *
 *                                                                   *
 *********************************************************************/
```

Source Codes

```
void set_bcs( float t[NRL+2][NC+2], int mype, int npes ){

  int i, j;

<<<<<<<<<<<<<< Set the Boundar values in all the PEs

}

/********************************************************************
 *                                                                  *
 * Print the trace only in the last PE where most action is         *
 *                                                                  *
 ********************************************************************/
void print_trace( float t[NRL+2][NC+2], int mype, int npes, int iter ){

  int joff, i;

  if( mype==npes-1 ){
    printf("\n---------- Iteration number: %d ------------\n", iter);

    joff = mype*NRL;
    for(i=NRL-10; i<=NRL; i++){
      printf("%15.8f", t[i][joff+i]);
    }
  }
  barrier();
  return;
}
```

Source Codes

laplace.t3e.f

```
*************************************************************
* Laplace Eqn
* T is initially 0.0
* Boundaries are as indicated below
*
*          T=0.
*                                                                     Y
*        _________                        ____|____|____|____         |
*        |       |  0                    |    |    |    |    |        |
*        |       |                       |    |    |    |    |        |
* T=0.   | T=0.0 |  T                    | 1  | 2  | 3  | 4  |        |
*        |       |                       |    |    |    |    |        |
*        |_______|  100                  |    |    |    |    |        |________X
*        0     100                       |____|____|____|____|
*                                             |    |    |
* Use Central Differencing Method
* Each process only has subgrid.
* Each Processor works on a sub grid and then sends its
* Boundaries to neighbours
*************************************************************
      program serial
      implicit none

      integer    NPROC,   NR,      NC,      NCL,           MXITER
      parameter (NPROC=4, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      integer    LEFT,     RIGHT,     ROOT
      parameter (LEFT=100, RIGHT=101, ROOT=0)

<<<<<<<<<<<<<<<<<< include file

      real*8     t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1), dt, dtg
      integer    i, j, iter, niter, mype, npes, ierr

<<<<<<<<<<<<<<<<<< Declaration for status for MPI_Recv
```

Source Codes

```
<<<<<<<<<<<<<<<<<< Initialize MPI

<<<<<<<<<<<<<<<<<< Find npes
<<<<<<<<<<<<<<<<<< Find mype number

      call initialize( t )
      call set_bcs( t, mype, npes )

      do i=0, NR+1
         do j=0, NCL+1
            told(i,j) = t(i,j)
         enddo
      enddo
<<<<<<<<<<<<<<<<<<< Get the Max number of iterations from user
<<<<<<<<<<<<<<<<<<< Broadcast it to all the PEs
*
* Do Computation on Sub-grid for Niter iterations
*
      Do 100 iter=1,niter

         Do j=1,NCL
            Do i=1,NR
               T(i,j) = 0.25 * ( Told(i+1,j)+Told(i-1,j)+
     $                           Told(i,j+1)+Told(i,j-1) )
            Enddo
         Enddo
*
* Copy
*
         dt = 0
         Do j=1,NCL
            Do i=1,NR
               dt        = max( abs(t(i,j) - told(i,j)), dt )
               Told(i,j) = T(i,j)
            Enddo
         Enddo
```

Source Codes

```
<<<<<<<<<<<<<<<<<<< Exchange boundary values

<<<<<<<<<<<<<<<<<<< Find max of dt in all the processors
*
* Print some Values
*
         If( mod(iter,100).eq.0 ) then
            call print_trace(t, mype, npes, iter)
         endif

         call MPI_Barrier( MPI_COMM_WORLD, ierr )
*
* Go to Next time step
*
 100  CONTINUE
<<<<<<<<<<<<<<<<<<<<< Clean up!
*
* End of Program!
*
      END
*----------------------------------------------------------------------*
      subroutine initialize( t )
      implicit none

      integer    NPROC,    NR,      NC,      NCL,          MXITER
      parameter (NPROC=4, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1)
      integer    i, j

      do i=0, NR+1
         do j=0, NCL+1
            t(i,j) = 0
         enddo
      enddo

      return
```

Source Codes

```
      end
*-----------------------------------------------------------------------*
      subroutine set_bcs( t, mype, npes )
      implicit none

      integer    NPROC,   NR,      NC,      NCL,          MXITER
      parameter (NPROC=4, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1), told(0:NR+1,0:NCL+1)
      integer    i, j, mype, npes
*
*Left and Right Boundaries
*
<<<<<<<<<<<<<<<<< Set the left and right boundary values
*
*Top and Bottom Boundaries
*
<<<<<<<<<<<<<<<< Set the top and bottom boundary values
      return

      end
*---------------------------------------------------------------------*
      subroutine print_trace(t, mype, npes, iter)
      implicit none

      integer    NPROC,   NR,      NC,      NCL,          MXITER
      parameter (NPROC=4, NR=1000, NC=1000, NCL=NC/NPROC, MXITER=1000)

      real*8     t(0:NR+1,0:NCL+1)
      integer    ioff, j, k, proc, mype, npes, iter

      if( mype.eq.npes-1 ) then
         write(6,1)iter
         ioff = mype*NCL
         write(6,3)(t(ioff+k,k), k=NCL-10 ,NCL)
      endif
      call barrier
```

Source Codes

```
1     format('---------- Iteration number: ', i10, '---------------')
3     format(5f15.8)
      return
      end
```