MPI Case Study
Jacobi-Solver
(Point-to-point communication)

Dieter an Mey

e-mail: anmey@rz.rwth-aachen.de

Jacobi-Algorithmus

Finite Difference Method to solve the 2D Helmholtz equation

$$\Delta u + \kappa u = f$$

with homogeneous Dirichlet-boundary conditions is solved with a finite difference method.

The Laplace operator Δ is discretized with the central 5-point difference star .

The corresponding lineare equations with a banded coefficient matrix are solved iteratively with the (simple) Jacobi method.

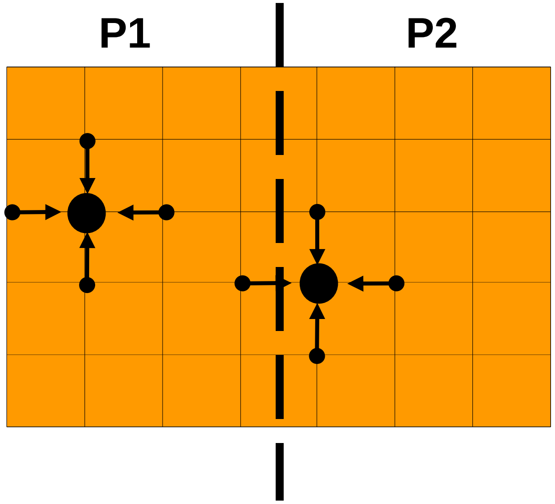
The Jacobi-method can easily be parallelized.

This example is from the OpenMP web page   www.openmp.org

Jacobi Solver – Serial Version

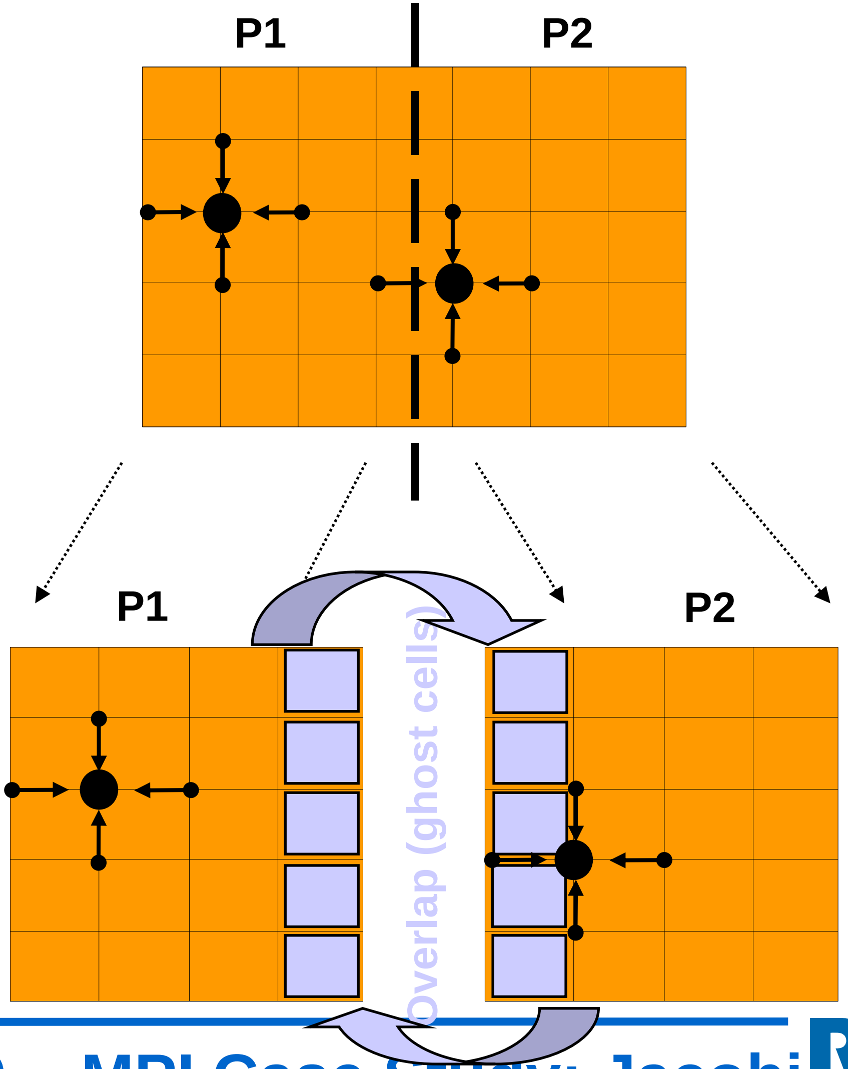
```
#define U(i,j) u[(i)*n+(j)]
#define UOLD(i,j) uold[(i)*n+(j)]
#define F(i,j) f[(i)*n+(j)]

/* ... */
  error = 10.0 * tol;
  k = 1;
  while (k <= maxit && error > tol) {
    error = 0.0;

    for (j=0; j<m; j++)
      for (i=0; i<n; i++)
        UOLD(j,i) = U(j,i);

    for (j=1; j<m-1; j++)
      for (i=1; i<n-1; i++){
        resid=(ax*(UOLD(j,i-1)+UOLD(j,i+1))+ay*(UOLD(j1,i)+UOLD(j+1,i))
             + b * UOLD(j,i) - F(j,i) ) / b;
        U(j,i) = UOLD(j,i) - omega * resid;
        error =error + resid*resid;
      }

    k++;
    error = sqrt(error) /(n*m);
  } /* while */
/* ... */
```

Jacobi-Method – Domain Decomposition



Shared
Memory
Parallelization

Jacobi-Method - Domain Decomposition



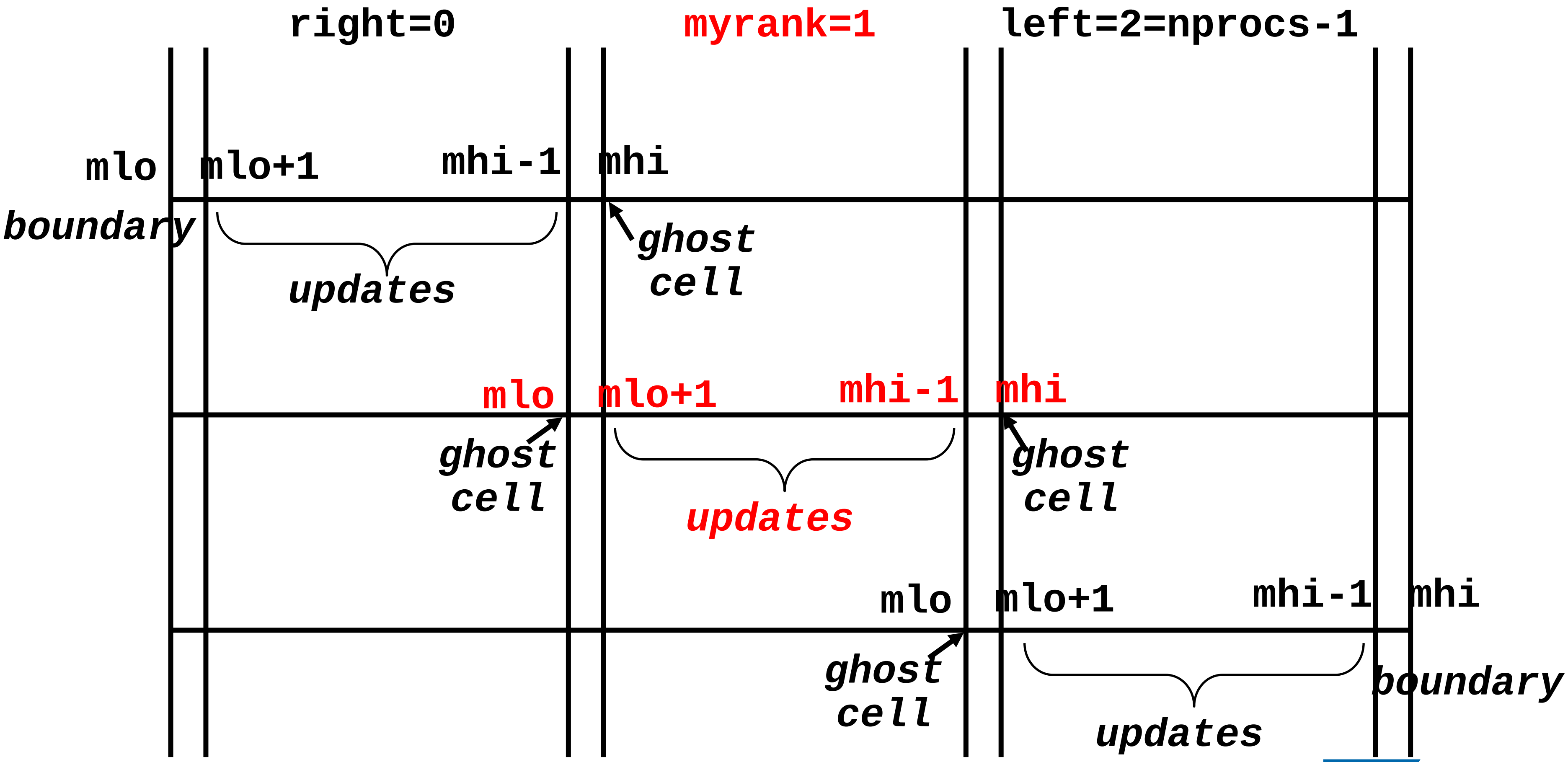
Shared Memory

Parallelization

Distributed Memory Parallelization

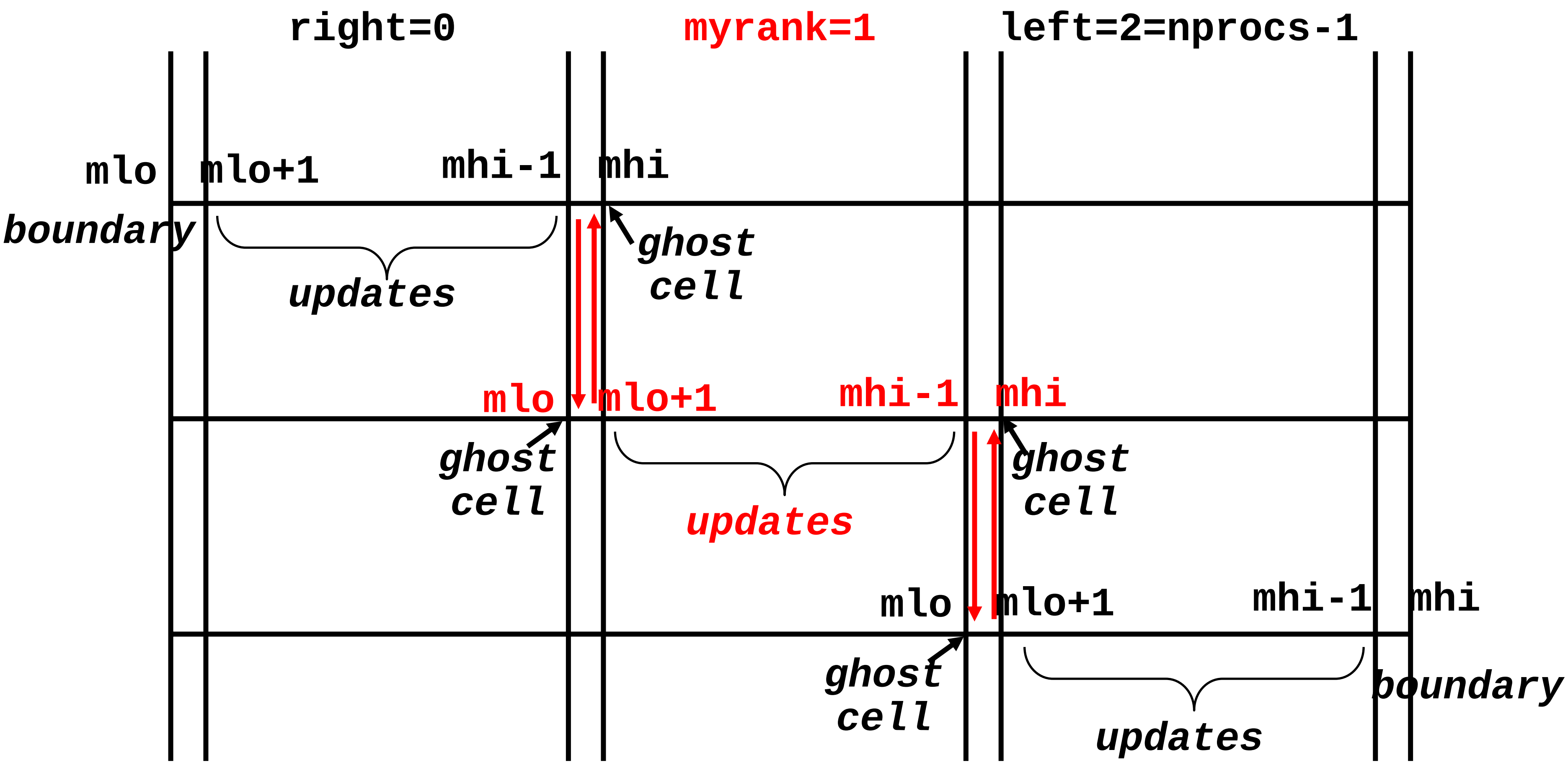
Jacobi-Method - Domain Decomposition

For example: nprocs = 3



Jacobi-Method - Domain Decomposition

For example: nprocs = 3

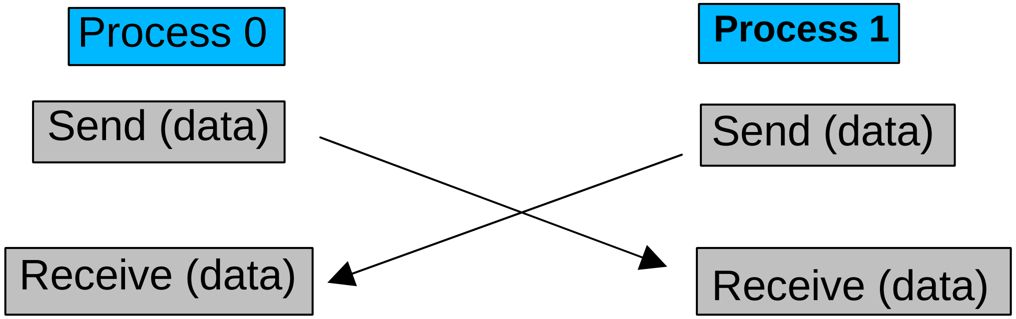


Jacobi – MPI Version 1: (B)locking send / recv ?

```
#define U(j,i) u[((j)-mlo)*n+(i)]
#define F(j,i) f[((j)-mlo)*n+(i)]
#define UOLD(j,i) uold[((j)-mlo)*n+(i)]
. . .
 while (k <= maxit && error > tol) {
    error = 0.0;
 /*  copy new solution into old - This is a very dangerous implementation. Why?  */
    if (myrank != nprocs-1){ /* send stripe mhi-1 to right neighbour */
      MPI_Send(&U(mhi-1,0), n, MPI_DOUBLE, myrank+1,tag, MPI_COMM_WORLD);
    }
    if (myrank != 0){ /* send stripe mlo+1 to left neighbour */
      MPI_Send(&U(mlo+1,0), n, MPI_DOUBLE, myrank-1,tag, MPI_COMM_WORLD);
    }

    if (myrank != 0){ /*  receive stripe mlo from left neighbour */
      MPI_Recv(&UOLD(mlo,0),n,MPI_DOUBLE,myrank-1,tag,MPI_COMM_WORLD,MPI_STATUS_IGNOR
    }
    if (myrank != nprocs-1 ){ /* receive stripe mhi from right neighbour */
      MPI_Recv(&UOLD(mhi,0),n,MPI_DOUBLE,myrank+1,tag,MPI_COMM_WORLD,MPI_STATUS_IGNOR
    }

    for (j=mlo+1; j<=mhi-1; j++) for (i=0; i<n; i++)  UOLD(j,i) = U(j,i);
    for (j=mlo+1; j<=mhi-1; j++) for (i=1; i<n-1; i++){
        resid = ...;
        U(j,i) = UOLD(j,i) - omega * resid;
        error = error + resid*resid;
     }
    error_local = error;
    MPI_Allreduce(&error_local, &error, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

    k++; error = sqrt(error) /(n*m);
  } /* while */
```

Deadlock?

How about this? Will it deadlock?



MPI_Sendrecv

MPI_Sendrecv(void *sendbuf,	What?
int sendcount,	How much?
MPI_Datatype sendtype,	Which type?
int dest,	destination?
int sendtag,	Message tag
void *recvbuf,	To where ?
int recvcount,	How much?
MPI_Datatype recvtype,	Which type?
int source,	From where?
int recvtag,	Message tag
MPI_Comm comm,	Communicator?
MPI_Status *status);	Receive Status?

Jacobi – MPI Version 2: Sendrecv

```
. . .
while (k <= maxit && error > tol) {
   error = 0.0;

   /* copy new solution into old */

   left = myrank-1;   if ( myrank == 0 )        left = MPI_PROC_NULL;
   right = myrank+1;  if ( myrank == nprocs-1 ) right = MPI_PROC_NULL;

     /* exchange stripe with right neighbour */
     MPI_Sendrecv(&U(mhi-1,0), n, MPI_DOUBLE, right,TAG_MOVE,
                 &UOLD(mhi,0), n , MPI_DOUBLE, right, TAG_MOVE,
                                             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
     /* exchange stripe with left neighbour */
     MPI_Sendrecv(&U(mlo+1,0), n, MPI_DOUBLE, left,TAG_MOVE,
                 &UOLD(mlo,0), n, MPI_DOUBLE, left,TAG_MOVE,
                           MPI_COMM_WORLD, MPI_STATUS_IGNORE);

   for (j=mlo+1; j<=mhi-1; j++) for (i=0; i<n; i++) UOLD(j,i) = U(j,i);
   for (j=mlo+1; j<=mhi-1; j++) for (i=1; i<n-1; i++){
       resid =(
               ax * (UOLD(j,i-1) + UOLD(j,i+1))
               + ay * (UOLD(j-1,i) + UOLD(j+1,i))
               + b * UOLD(j,i) - F(j,i)
               ) / b;
       U(j,i) = UOLD(j,i) - omega * resid;
       error = error + resid*resid;
    }
   error_local = error;
   MPI_Allreduce(&error_local, &error, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
   k++; error = sqrt(error) /(n*m);
 } /* while */
```

Jacobi – MPI Version 3: Asynchroneous Send/Recv

```
MPI_Request request[4];
MPI_Status  status[4];
int reqcnt,left, right;

. . .
/* copy new solution into old */
  left = myrank-1;  if ( myrank == 0 ) left = MPI_PROC_NULL;
  right = myrank+1; if ( myrank == nprocs-1 ) right = MPI_PROC_NULL;

    reqcnt = 0;
    /*  receive stripe mlo from left neighbour blocking */
    MPI_Irecv(&UOLD(mlo,0), n, MPI_DOUBLE, left,
              TAG_MOVE_RIGHT, MPI_COMM_WORLD, &request[reqcnt]);
    reqcnt++;
    /* receive stripe mhi from right neighbour blocking */
    MPI_Irecv(&UOLD(mhi,0), n , MPI_DOUBLE, right,
              TAG_MOVE_LEFT, MPI_COMM_WORLD, &request[reqcnt]);
    reqcnt++;
    /* send stripe mhi-1 to right neighbour async */
    MPI_Isend(&U(mhi-1,0), n, MPI_DOUBLE, right,
              TAG_MOVE_RIGHT, MPI_COMM_WORLD, &request[reqcnt]);
    reqcnt++;
    /* send stripe mlo+1 to left neighbour async */
    MPI_Isend(&U(mlo+1,0), n, MPI_DOUBLE, left,
              TAG_MOVE_LEFT, MPI_COMM_WORLD, &request[reqcnt]);

 for (j=mlo+1; j<=mhi-1; j++)
    for (i=0; i<n; i++)
      UOLD(j,i) = U(j,i);

 MPI_Waitall(reqcnt, request, status);
```

Overlap Communication and Computation

Jacobi Solver – OpenMP Version

```
#define U(i,j) u[(i)*n+(j)]
#define UOLD(i,j) uold[(i)*n+(j)]
#define F(i,j) f[(i)*n+(j)]
#include <omp.h>
/* ... */
  error = 10.0 * tol;
  k = 1;
  while (k <= maxit && error > tol) {
    error = 0.0;
#pragma omp parallel private(j,i,resid) \
                  shared(m,n,ax,ay,u,uold,f,b,omega)
{
#pragma omp for
    for (j=0; j<m; j++)
      for (i=0; i<n; i++)
        UOLD(j,i) = U(j,i);
#pragma omp for reduction(+:error)
    for (j=1; j<m-1; j++)
      for (i=1; i<n-1; i++){
        resid=(ax*(UOLD(j,i-1)+UOLD(j,i+1))+ay*(UOLD(j-1,i)+UOLD(j+1,i))
             + b * UOLD(j,i) - F(j,i) ) / b;
        U(j,i) = UOLD(j,i) - omega * resid;
        error =error + resid*resid;
      }
}   /* end of parallel region */
    k++;
    error = sqrt(error) /(n*m);
  } /* while */
/* ... */
```

Jacobi – MPI+OpenMP=Hybrid

```
. . .
 while (k <= maxit && error > tol) {
    error = 0.0;

    /* copy new solution into old */

    left = myrank-1;   if ( myrank == 0 )        left = MPI_PROC_NULL;
    right = myrank+1;  if ( myrank == nprocs-1 ) right = MPI_PROC_NULL;

      /* exchange stripe with right neighbour */
      MPI_Sendrecv(&U(mhi-1,0), n, MPI_DOUBLE, right,TAG_MOVE,
                  &UOLD(mhi,0), n , MPI_DOUBLE, right, TAG_MOVE,
                                            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      /* exchange stripe with left neighbour */
      MPI_Sendrecv(&U(mlo+1,0), n, MPI_DOUBLE, left,TAG_MOVE,
                  &UOLD(mlo,0), n, MPI_DOUBLE, left,TAG_MOVE,
                            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
#pragma omp parallel for
    for (j=mlo+1; j<=mhi-1; j++) for (i=0; i<n; i++) UOLD(j,i) = U(j,i);
#pragma omp parallel for private(resid) reduction(+:error)
    for (j=mlo+1; j<=mhi-1; j++) for (i=1; i<n-1; i++){
        resid =(
                ax * (UOLD(j,i-1) + UOLD(j,i+1))
                + ay * (UOLD(j-1,i) + UOLD(j+1,i))
                + b * UOLD(j,i) - F(j,i)
                ) / b;
        U(j,i) = UOLD(j,i) - omega * resid;
        error = error + resid*resid;
     }
    error_local = error;
    MPI_Allreduce(&error_local, &error, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    k++; error = sqrt(error) /(n*m);
  } /* while */
```