

your logo here

CRAY T3E

Parallelization of grid-based applications

Peter Michielse, Ronald van Pelt

your logo here

Agenda

- Numerical solution of the Laplace equation (serial)
- Numerical solution of the Laplace equation (parallel)
- Intermezzo on Cray T3E hardware
- Parallel implementation

your logo here

The Laplace equation

The Laplace equation in two dimensions:

$$-\Delta u = f \quad \text{on} \quad \Omega = (0,1) \times (0,1)$$

$$u = g \quad \text{on} \quad d\Omega \quad (\text{Dirichlet})$$

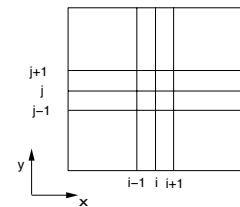
Approximate solution required:

- * Define a mesh or grid consisting of points (x_i, y_j)
- * Compute the (approximate) solution in these grid points

your logo here

Discretization

Discretization on the computational domain by means of finite differences:

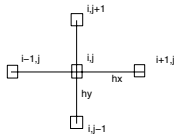


Several approaches to compute $u(i,j)$ based on its neighbouring points. This is called a *stencil*.

Note that the distances in x- and y-direction do not need to be equal (non-equidistant).

Stencil

The stencil we will use is the following (5-point):



The solution method we will use is iterative.
Its general form is:

$$u(i,j) = F(u(i,j), u(i-1,j), u(i+1,j), u(i,j-1), u(i,j+1), f, hx, hy)$$

In words: the new value of $u(i,j)$ is computed from the old and new values from itself and its neighbours

For simplicity, assume: $hx = hy = h$

Discretization

$$\Delta = \delta^2 u / \delta x^2 + \delta^2 u / \delta y^2$$

$$\delta u / \delta x = \frac{u(i+1/2) - u(i-1/2)}{h}$$

$$\delta^2 u / \delta x^2 \longrightarrow \frac{u(i+1,j) - 2u(i,j) + u(i-1,j)}{h^2}$$

$$\delta^2 u / \delta y^2 \longrightarrow \frac{u(i,j+1) - 2u(i,j) + u(i,j-1)}{h^2}$$

Then:

$$-u(i+1,j) + 2u(i,j) - u(i-1,j) - u(i,j+1) + 2u(i,j) - u(i,j-1) = h^2 f(i,j)$$

$$u(i,j) = 0.25 * h^2 f(i,j) + 0.25 * (u(i-1,j) + u(i+1,j) + u(i,j-1) + u(i,j+1))$$

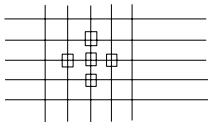
Example iterative methods

Point Jacobi:

$$u_{\text{new}}(i,j) = 0.25 * (h^2 f(i,j) + u_{\text{old}}(i-1,j) + u_{\text{old}}(i+1,j) + u_{\text{old}}(i,j-1) + u_{\text{old}}(i,j+1))$$

Point Gauss-Seidel:

$$u_{\text{new}}(i,j) = 0.25 * (h^2 f(i,j) + u_{\text{new}}(i-1,j) + u_{\text{old}}(i+1,j) + u_{\text{new}}(i,j-1) + u_{\text{old}}(i,j+1))$$



Example iterative methods

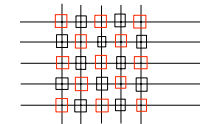
Red-Black Gauss-Seidel:

$$u_{\text{new}}(i,j) = 0.25 * (h^2 f(i,j) + u_{\text{new}}(i-1,j) + u_{\text{old}}(i+1,j) + u_{\text{new}}(i,j-1) + u_{\text{old}}(i,j+1))$$

Two sweeps:

red sweep: all red points

black sweep: all black points



Treatment of boundaries

Dirichlet boundary conditions:
 $u = g$ on $d\Omega$

Suppose, the number of internal grid points is $N \times N$
Then, including the boundaries, the total number of points is $(N+2) \times (N+2)$

Then, the distance between two neighbouring points is $h = 1/(N+1)$

Internal points range: $i = 1, \dots, N$
 $j = 1, \dots, N$

Before start of iteration process, initialize the boundary point solution to g , and use it in the stencil calculations

Basic serial algorithm

The basic serial algorithm looks as follows:

Input

Initialize grid
Initialize boundary values

while ((no convergence) .and. (no_max_iter))

 Sweep over red points
 Sweep over black points
 Determine convergence
 uold = unew

end while

Output

Detailed serial algorithm

"Initialize grid"

Remember:

$$\begin{aligned} -\Delta u &= f \quad \text{on } \Omega = (0,1) \times (0,1) \\ u &= g \quad \text{on } d\Omega \end{aligned}$$

```
do 40 j = nystart-1,nyend+1
do 40 i = nxstart-1,nxend+1
  x(i,j) = dble(i)/dble(nx+1)
  y(i,j) = dble(j)/dble(ny+1)
  f(i,j) = funcf(x(i,j),y(i,j),itype)
  uexact(i,j) = uexactf(x(i,j),y(i,j),itype)
  uold(i,j) = 0.0d0
40 continue
```

itype denotes the kind of testproblem

Detailed serial algorithm

"Initialize boundary values"

```
c
c south boundary
c
  if (south .eq. -1) then
    j = 0
    do 51 i = 0,nx+1
      xx = x(i,j)
      yy = y(i,j)
      u(i,j) = exp(-(xx-yy)*(xx-yy))
51 continue
  endif
c
c north boundary
c
  if (north .eq. -1) then
    j = ny+1
    do 61 i = 0,nx+1
      xx = x(i,j)
      yy = y(i,j)
      u(i,j) = exp(-(xx-yy)*(xx-yy))
61 continue
  endif
c
c west boundary .....
```

Detailed serial algorithm

"Sweep over red points"

```
do 20 j = nystart,nyend
  do 10 i = istart,nxend,2

    unew(i,j) = 0.25d0*h*h*f(i,j) +
    & 0.25d0*
    & (uold(i-1,j)+uold(i+1,j)+uold(i,j-1)+uold(i,j+1))

  10 continue

  if (istart .eq. nxstart) then
    istart = nxstart + 1
  else
    istart = nxstart
  endif

  20 continue
```

Detailed serial algorithm

"Sweep over black points"

```
do 60 j = nystart,nyend
  do 50 i = istart,nxend,2

    unew(i,j) = 0.25d0*h*h*f(i,j) +
    & 0.25d0*
    & (unew(i-1,j)+unew(i+1,j)+unew(i,j-1)+unew(i,j+1))

  50 continue

  if (istart .eq. nxstart) then
    istart = nxstart + 1
  else
    istart = nxstart
  endif

  60 continue
```

Detailed serial algorithm

"Determine convergence"

```
rmax = -100000.0

do 20 j = nystart-1,nyend+1
  do 20 i = nxstart-1,nxend+1
    if (abs(uold(i,j) - unew(i,j)) .gt. rmax) then
      rmax = abs(uold(i,j) - unew(i,j))
      ico = i
      jco = j
    endif
  20 continue

  if (rmax .lt. eps) conv = .true.
```

How to parallelize ?

Most work in the red (and also black) sweep:

```
do 10 i = istart,nxend,2
  unew(i,j) = 0.25d0*h*h*f(i,j) +
  & 0.25d0*
  & (uold(i-1,j)+uold(i+1,j)+uold(i,j-1)+uold(i,j+1))
10 continue
```

On a shared-memory parallel machine, the compiler should automatically parallelize this loop:

Fine-grained parallelism

Appropriate for the T3E ?

Coarse-grained parallelism

So, let's go for coarser-grained parallelism:

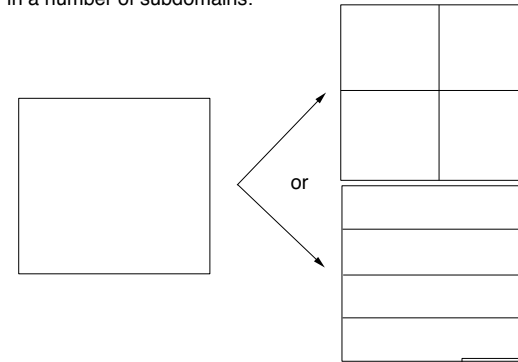
- less synchronization points
- less communication (when implemented on a distributed memory machine)

Typically, domain decomposition is used to accomplish coarse-grained parallelism

Note that domain decomposition parallelism can be applied on both shared memory and distributed memory machines

Domain decomposition

Domain decomposition splits the computational domain in a number of subdomains:



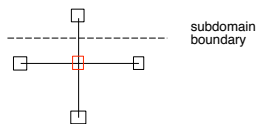
Domain decomposition

Each subdomain is devoted to a processor

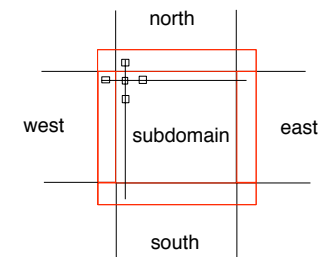
The processor executes the numerical algorithm for the points in its subdomain:

```
do 10 i = istart,nxend,2
  unew(i,j) = 0.25d0*h*h*f(i,j) +
&           0.25d0*
&           (uold(i-1,j))+uold(i+1,j)+uold(i,j-1)+uold(i,j+1))
10 continue
```

Immediately, it becomes clear where the problems are:



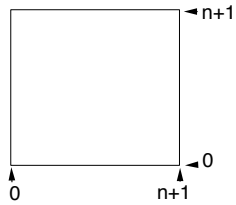
Domain decomposition



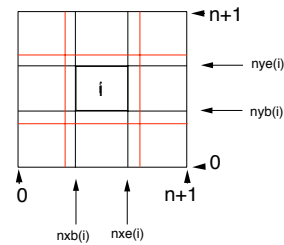
So, each subdomain computes the solution in its internal area. For the points at the internal boundary, it needs input from neighbouring subdomains

Grid points numbering

Global domain:



Global numbering is kept in subdomains:



So, arrays in subdomains run from $nxb(i)-1$ to $nxe(i)+1$, and from $nyb(i)-1$ to $nyb(i)+1$

Implementation aspects

For the remainder, we make a few assumptions on the parallel implementation:

- Distributed memory model
- Each processor only knows the subdomain it is responsible for
- T3E in mind: SPMD (= Single Program Multiple Data)
- MPI
- One of the MPI instances takes care of the administrative issues (and also of its own subdomain)
- General subdomain configurations (nx blocks in x -direction, ny blocks in y -direction)
- In MPI, the processor id's range from 0 to $nprocs-1$
- We assume the processor with $id = 0$ to do the global administration issues

Basic parallel algorithm

MPI initialization

Input, and communication of input
Initialize (sub)grid, determine neighbours
Initialize boundary values

while ((no convergence) .and. (no_max_iter))

Sweep over red points
Communicate red values at internal boundaries
Sweep over black points
Communicate black values at internal boundaries

Determine convergence (in subdomain)
Communicate to single PE ($my_id = 0$)
Determine global convergence (one PE)
Communicate convergence result

end while

Output

Note that each processor runs the same program !

Detailed parallel algorithm

"MPI initialization"

```
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)
```

numprocs comes from:
`mpirun -np numprocs executable`

myid is the id of the process/processor

Detailed parallel algorithm

"Input, and communication of input"

```

if (myid .eq. 0) then

  ibuffer(1) = isubs
  ibuffer(2) = kbx      blocks in x-direction
  ibuffer(3) = kby      blocks in y-direction
  ibuffer(4) = nmax     max. iterations

  itag = 1

  do id = 1,numprocs-1
    call MPI_SEND(ibuffer,4,MPI_INTEGER,id,itag,
    & MPI_COMM_WORLD,ierr)
  enddo

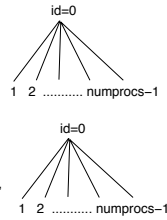
  rbuffer(1) = omega    relaxation parameter, assume 1.0
  rbuffer(2) = eps      relative accuracy

  itag = 2

  do id = 1,numprocs-1
    call MPI_SEND(rbuffer,2,MPI_DOUBLE_PRECISION,id,itag,
    & MPI_COMM_WORLD,ierr)
  enddo

else
  ..... next page .....

```



Detailed parallel algorithm

"Input, and communication of input"

```

.... previous page ....

else

c
c Slaves receive data and compute related parameters
c

  call MPI_RECV(ibuffer,4,MPI_INTEGER,0,1,
  & MPI_COMM_WORLD,istat,ierr)

  isubs = ibuffer(1)
  kbx   = ibuffer(2)
  kby   = ibuffer(3)
  nmax  = ibuffer(4)

  kblock = kbx*kby

  call MPI_RECV(rbuffer,2,MPI_DOUBLE_PRECISION,0,2,
  & MPI_COMM_WORLD,istat,ierr)

  omega = rbuffer(1)
  eps   = rbuffer(2)

endif

```

Detailed parallel algorithm

"Initialize (sub)grid, determine neighbours"

```

do 201 ky = 1,kby
do 201 kx = 0,kbx-1

  ibls((ky-1)*kbx+kx) = nint(dble(  kx*nx)/dble(kbx))+1
  ible((ky-1)*kbx+kx) = nint(dble((kx+1)*nx)/dble(kbx))
  jbls((ky-1)*kbx+kx) = nint(dble((ky-1)*ny)/dble(kby))+1
  jble((ky-1)*kbx+kx) = nint(dble(  ky*ny)/dble(kby))

201 continue
c
c Now, slave myid is responsible for the global points:
c
c   ibls(myid) to ible(myid) in X-direction
c   jbls(myid) to jble(myid) in Y-direction

```

Note:

- kbx and kby were input values, and have been communicated to each id
- Each processor (id) runs this code 1
- Each processor knows on which grid points every other processor works on

Detailed parallel algorithm

"Initialize (sub)grid, determine neighbours"

```

c
c Next, determine neighbours of myid
c

  myid_div = myid/kbx
  myid_mod = mod(myid,kbx)

  south = myid - kbx
  if (myid_div .eq. 0) south = -1

  north = myid + kbx
  if (myid_div .eq. kby-1) north = -1

  west = myid - 1
  if (myid_mod .eq. 0) west = -1

  east = myid + 1
  if (myid_mod .eq. kbx-1) east = -1

  neighbours(1) = north
  neighbours(2) = south
  neighbours(3) = west
  neighbours(4) = east

```

| | | |
|---|----|----|
| 9 | 10 | 11 |
| 6 | 7 | 8 |
| 3 | 4 | 5 |
| 0 | 1 | 2 |

So, each processor (id) knows the id's of its (max. 4) neighbours

Detailed parallel algorithm

Basically, the administration is ready now:

- Each processor knows on which part of the global domain it works
- Each processor knows the lower and upper bounds of its arrays, and can allocate them
- Each processor knows its own id, and knows the id's of its neighbours
- Furthermore, each processor knows which of its neighbours is north, south, east or west
- If there is no neighbour on either of the 4 sides (subdomain has a global boundary), the neighbour is set to -1

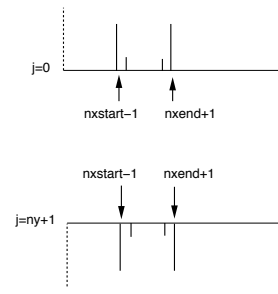
Detailed parallel algorithm

"Initialize boundary values"

```

c
c south boundary
c
  if (south.eq. -1) then
    j = 0
    do 51 i = nxstart-1, nxend+1
      xx = x(i,j)
      yy = y(i,j)
      u(i,j) = exp(-(xx-yy)*(xx-yy))
    51 continue
  endif
c
c north boundary
c
  if (north.eq. -1) then
    j = ny+1
    do 61 i = nxstart-1, nxend+1
      xx = x(i,j)
      yy = y(i,j)
      u(i,j) = exp(-(xx-yy)*(xx-yy))
    61 continue
  endif
c
c west boundary .....

```



So, basically no changes compared to serial algorithm

Detailed parallel algorithm

"Sweep over red points"

```

do 20 j = nystart, nyend
  do 10 i = istart, nxend, 2
    unew(i,j) = 0.25d0*h*h*f(i,j) +
    & 0.25d0*
    & (uold(i-1,j)+uold(i+1,j)+uold(i,j-1)+uold(i,j+1))
  10 continue
  if (istart.eq. nxstart) then
    istart = nxstart + 1
  else
    istart = nxstart
  endif
  20 continue

```

So, no changes compared to serial algorithm, provided that:

- input for internal boundary points is up-to-date (must have come from neighbouring subdomains)

Detailed parallel algorithm

"Sweep over black points"

```

do 60 j = nystart, nyend
  do 50 i = istart, nxend, 2
    unew(i,j) = 0.25d0*h*h*f(i,j) +
    & 0.25d0*
    & (unew(i-1,j)+unew(i+1,j)+unew(i,j-1)+unew(i,j+1))
  50 continue
  if (istart.eq. nxstart) then
    istart = nxstart + 1
  else
    istart = nxstart
  endif
  60 continue

```

The same holds here:
No changes compared to serial algorithm, provided that input for internal boundary points is available

Detailed parallel algorithm

"Communicate red values at internal boundaries"

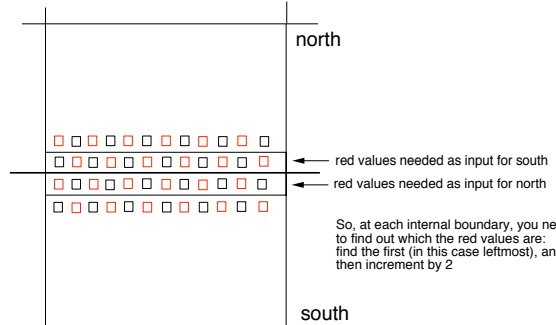
Several issues to take care of:

- What to communicate ?
which values
- How to communicate ?
which order or strategy

Detailed parallel algorithm

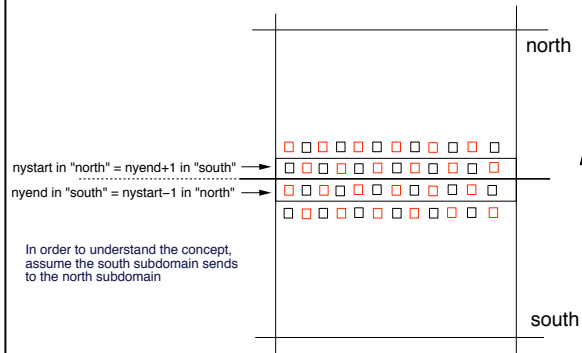
"Communicate red values at internal boundaries"

Which values to communicate ?



Detailed parallel algorithm

"Communicate red values at internal boundaries"



Detailed parallel algorithm

"Communicate red values at internal boundaries"

```

if (nnorth .ge. 0) then          determine "if" there is a north neighbour
  if (color .eq. 1) then        color=1 (communicate red), color=0 (communicate black)
c Red
c
c   if (mod(nyend,2) .eq. 1) then  red points: (even,even) and (odd,odd)
c   if (mod(nxstart,2) .eq. 1) then black points: (even,odd) and (odd,even)
c     istart = nxstart
c   else
c     istart = nxstart + 1
c   endif
c   if (mod(nxstart,2) .eq. 1) then
c     istart = nxstart + 1
c   else
c     istart = nxstart
c   endif
c   else
c Black .....
c   .....
c   endif
c   endif

```

Note: this code is executed in the south subdomain

Detailed parallel algorithm

"Communicate red values at internal boundaries"

```

nnorth = neighbours(1)      id of MPI-instance that takes care of north
                             neighbour
do i = istart,nxend,2
  rbufnorth(i) = unew(i,nyend) fill a buffer array to use in MPI_SEND
enddo

length = nxend+1-(nxstart-1)+1 determine number of values to send
itag = 4*(niter-1)+103

call MPI_SEND(rbufnorth,length,MPI_DOUBLE_PRECISION,nnorth,itag,
&
  MPI_COMM_WORLD,ierr)

```

itag characterizes this specific message. It is important to have each different message to have a different tag value;

nnorth is the destination of the message

Note: this code is executed in the south subdomain

Detailed parallel algorithm

"Communicate red values at internal boundaries"

```

if (nsouth .ge. 0) then      determine "if" there is a south neighbour
  if (color .eq. 1) then    color=1 (communicate red), color=0 (communicate black)
c
c Red
c
  if (mod(nystart-1,2) .eq. 1) then red points: (even,even) and (odd,odd)
  if (mod(nxstart,2) .eq. 1) then black points: (even,odd) and (odd,even)
    istart = nxstart
  else
    istart = nxstart + 1
  endif
  else
  if (mod(nxstart,2) .eq. 1) then
    istart = nxstart + 1
  else
    istart = nxstart
  endif
  endif
  else
c
c Black
c
  .....
endif
endif

```

Note: this code is executed in the north subdomain

Detailed parallel algorithm

"Communicate red values at internal boundaries"

```

nsouth = neighbours(2)      id of MPI-instance that takes care of south
                             neighbour
length = nxend+1-(nxstart-1)+1 determine number of values to receive
itag = 4*(niter-1)+103

call MPI_RECV(rbufsouth,length,MPI_DOUBLE_PRECISION,nsouth,itag,
&
  MPI_COMM_WORLD,istat,ierr)

do i = istart,nxend,2
  unew(i,nystart-1) = rbufsouth(i) store the received values in the appropriate
  array
enddo

```

Note that the value of itag should correspond with the value in the corresponding MPI_send message

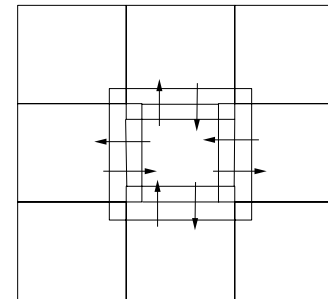
nsouth is the source of the message

Note: this code is executed in the north subdomain

Detailed parallel algorithm

"Communicate red values at internal boundaries"

Which communication strategy ?

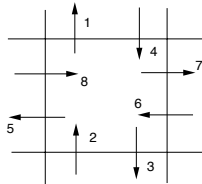


So, exchange of 8 messages between a subdomain and its neighbours

Detailed parallel algorithm

A safe strategy is the following:

- Each subdomain sends its north boundary to its north neighbour (if any)
Then, each subdomain receives from its south neighbour (if any)
- Next, each subdomain sends its south boundary to its south neighbour (if any)
Then, each subdomain receives from its north neighbour (if any)
- Next, each subdomain sends its west boundary to its west neighbour (if any)
Then, each subdomain receives from its east neighbour (if any)
- Next, each subdomain sends its east boundary to its east neighbour (if any)
Then, each subdomain receives from its west neighbour (if any)



Detailed parallel algorithm

"Sweep over black points"

Basically equivalent to the "sweep over red points"

"Communicate black values at internal boundaries"

Basically equivalent to the "communication of red values at internal boundaries"

Detailed parallel algorithm

"Determine convergence (in subdomain)"

```
rmax = -100000.0
```

```
do 20 j = nystart-1,nyend+1
do 20 i = nxstart-1,nxend+1
  if (abs(uold(i,j) - unew(i,j)) .gt. rmax) then
    rmax = abs(uold(i,j) - unew(i,j))
    ico = i
    jco = j
  endif
20 continue
```

So, no changes compared to serial version, except that no global decision on convergence can be taken yet

Detailed parallel algorithm

"Communicate to single PE (my_id = 0)"

```
c
c Send convergence results to master
c
  if (myid.ne. 0) then
    rbuffer(1) = rmax
    itag = 100000 + 2*(n-1) + 1
    call MPI_SEND(rbuffer,1,MPI_DOUBLE_PRECISION,0,itag,
      & MPI_COMM_WORLD,ierr)
  c
  c Wait for message of master to continue or not
  c
  .....
```

So, each processor (except 0) sends its maximum difference to processor 0, and then starts waiting for a message from processor 0 on what to do further

Detailed parallel algorithm

"Determine global convergence"

```
if (myid .ne. 0) then
  "subdomains send to processor 0"   see previous slide
else
  c
  c Master receives local convergence results and decides whether
  c to continue or not
  rmax_ar(0) = rmax
  itag = 100000 + 2*(n-1) + 1
  do id = 1, numprocs-1
    call MPI_RECV(rbuffer, 1, MPI_DOUBLE_PRECISION, id, itag,
      & MPI_COMM_WORLD, iostat, ierr)   source could be -1
    &
    rmax_ar(id) = rbuffer(1)
    if (rmax_ar(id) .gt. rmax) rmax = rmax_ar(id)
  enddo
  signal = 0
  if (rmax .lt. eps) then
    conv = .true.
    signal = 1
  endif
endif
endif
```

Detailed parallel algorithm

"Communicate convergence result"

```
if (myid .eq. 0) then
  c
  c Master send slaves message to continue or not
  c
  itag = 100000 + 2*(n-1) + 2
  do id = 1, numprocs-1
    call MPI_SEND(signal, 1, MPI_INTEGER, id, itag,
      & MPI_COMM_WORLD, ierr)
  enddo
endif
```

So, each subdomain receives one integer ("signal"):

- * signal = 1: global convergence, stop
- * signal = 0: no global convergence yet, continue

Detailed parallel algorithm

"Communicate convergence result"

```
c
c Wait for message of master to continue or not
c
signal = -1
itag = 100000 + 2*(n-1) + 2
call MPI_RECV(signal, 1, MPI_INTEGER, 0, itag,
  & MPI_COMM_WORLD, iostat, ierr)
```

Message received from processor 0
Depending on signal (0 or 1), continue or not

Exercise 1

Write an MPI program for $nprocs$ processors which splits a square computational domain into a number of subdomains. The number of subdomains in the x -direction is kbx , in the y -direction kby . Assume that $nprocs = kbx * kby$.

kbx and kby are input to process 0. Assume the subdomain numbers range from 0 to $nprocs-1$.

Desired output: for each subdomain, print its north, south, east and west neighbour. If a subdomain does not have a neighbour in a certain direction, print -1 .

Exercise 2

Copy the directory with the example red–black solver to your own directory. In subroutine "rbcomm", the communication between the subdomains takes place, as described earlier.

As you can see, this is the safe strategy.

Change the strategy to, e.g., a less safe strategy by having each subdomain sending to all its neighbours, and then receiving from all its neighbours.

Determine whether this influences the elapsed execution time.