Scientific Computing on Parallel Machines

Cálculo científico en ordenadores paralelos

Markus Uhlmann

CIEMAT

www.ciemat.es/sweb/comfos/personal/uhlmann

UCM – Abril 2008

Schedule

Part I	Introduction to parallel programming	lecture 1
Part II	Introduction to MPI	
	General introduction & MPI program structure	lecture 2
	Point-to-point communication	lecture 3,4
	Collective communication	lecture 5
	2D Poisson example	lecture 6,7
	Non-contiguous data & Mixed datatypes	lecture 7
	Virtual topologies & Communication subsets	lecture 8
	Use of linear algebra libraries	lecture 9
	Extensions	lecture 10

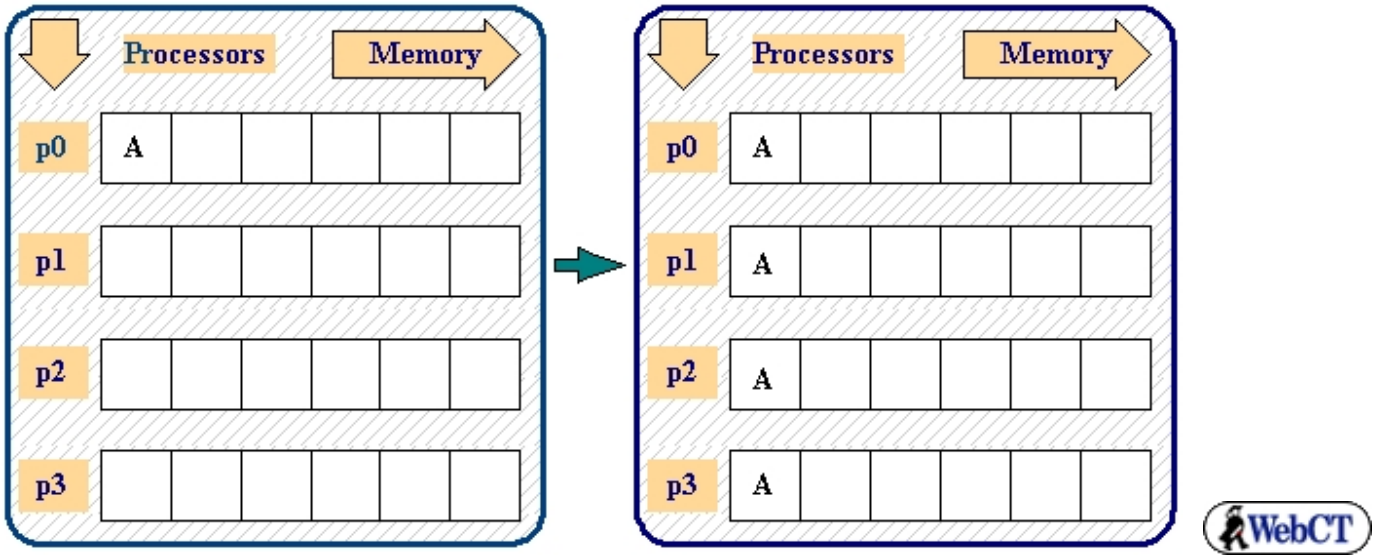
Collective Communication

Communication involving all members of communicator

- broadcasting of data
- gathering of data
- scattering of data
- reduction operations
- barrier synchronization

$\Rightarrow$ all processors in a communicator need to participate! (global)

Broadcast Operation



`MPI_BCAST(buffer,icount,dtype,iroot,icomm,ierr)`

- one-to-all communication operation (source)

Collective communication: Matching of Messages

Conditions for matching:

- same conditions apply as for point-to-point
- additionally:
  amount of data must exactly match (`icount`)

Collective communication involves all members

Wrong code:

```
if(myrank.eq.iroot)then
    call MPI_BCAST(p,1,MPI_REAL,iroot,MPI_COMM_WORLD,err)
endif
```

$\rightsquigarrow$ all members of communicator need to participate!

Collective communication deadlock

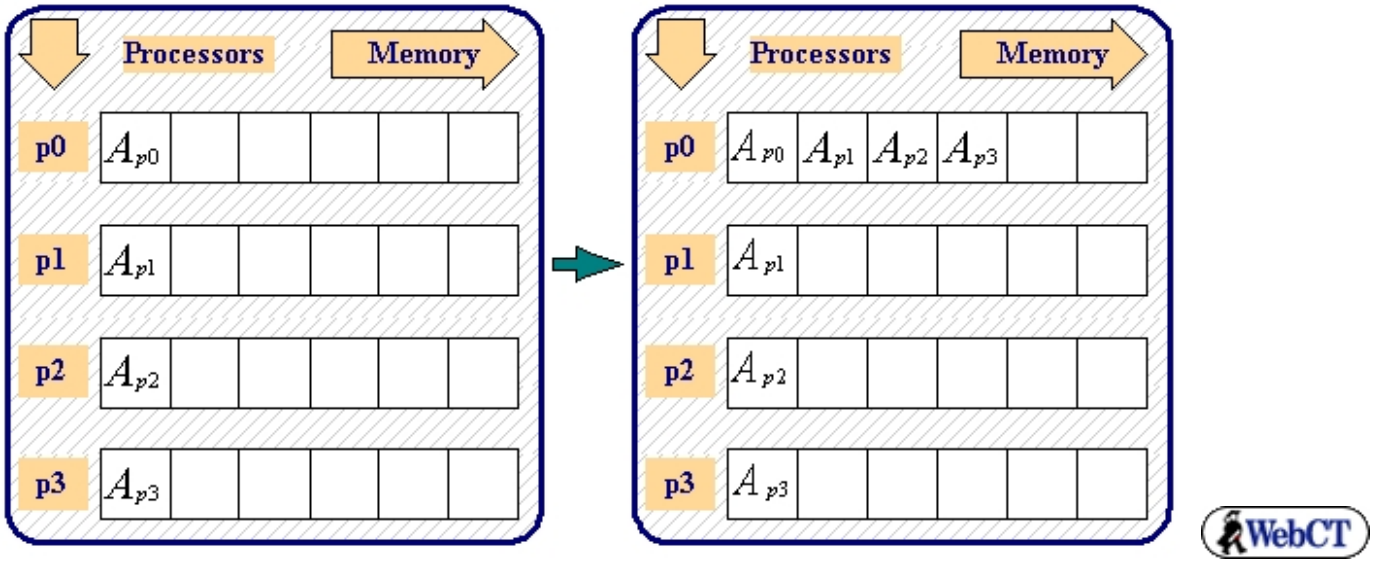
Wrong code II:

suppose that `size=2`

```
if(myrank.eq.0)then
    call MPI_BCAST(p,1,MPI_REAL,0,MPI_COMM_WORLD,err)
    call MPI_BCAST(p,1,MPI_REAL,1,MPI_COMM_WORLD,err)
elseif(myrank.eq.1)then
    call MPI_BCAST(p,1,MPI_REAL,1,MPI_COMM_WORLD,err)
    call MPI_BCAST(p,1,MPI_REAL,0,MPI_COMM_WORLD,err)
endif
```

$\rightsquigarrow$ all members need to post calls in the same order!

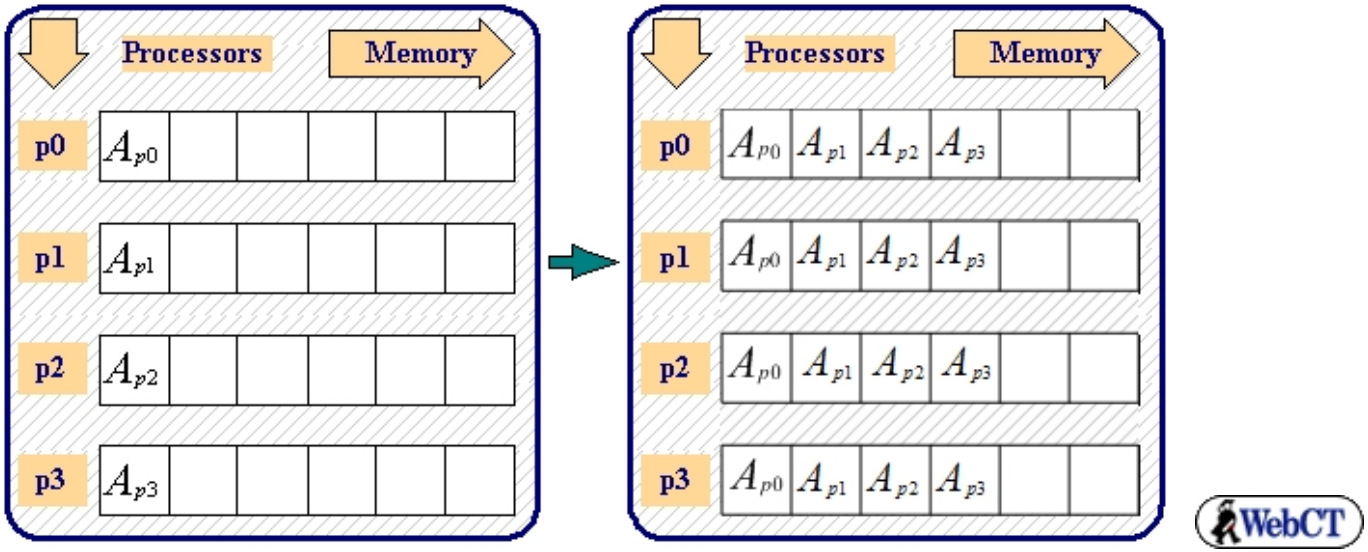
Gather Operation




```
MPI_GATHER(sendbuff,sendcount,sendtype,
           recvbuff,recvcount,recvtype,iroot,icomm,ierr)
```

- all-to-one communication operation (source)

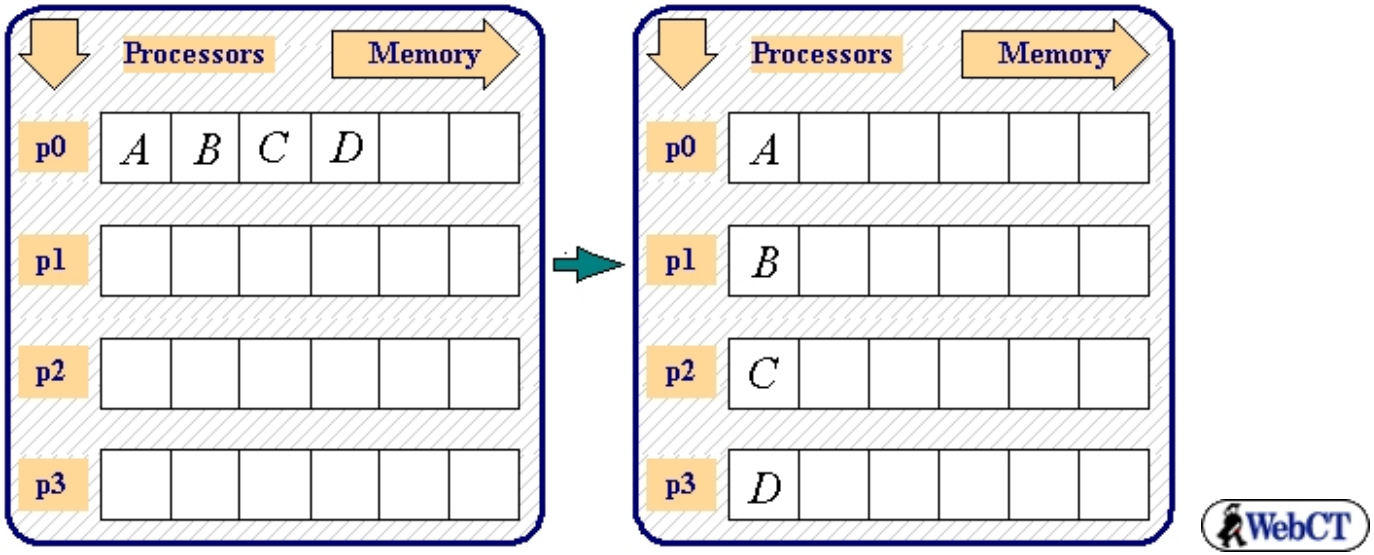
Allgather Operation




```
MPI_ALLGATHER(sendbuff,sendcount,sendtype,
              recvbuff,recvcount,recvtype,icomm,ierr)
```

- all-to-all communication operation

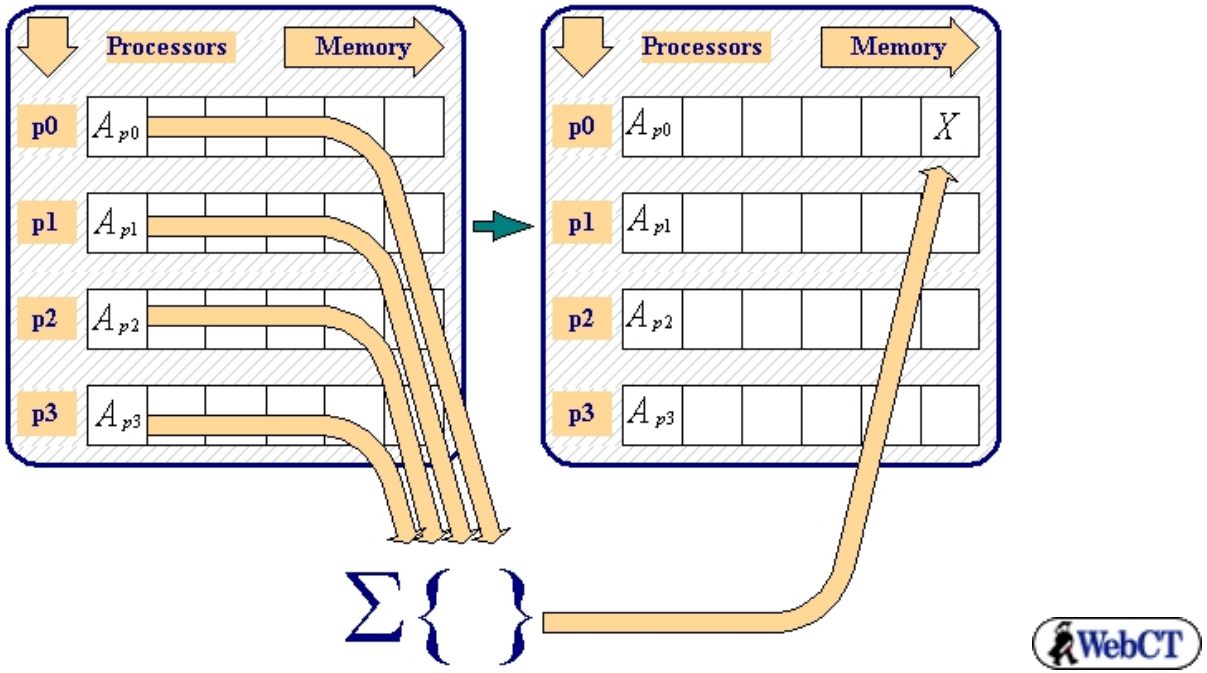
Scatter Operation




```
MPI_SCATTER(sendbuff,sendcount,sendtype,
            recvbuff,recvcount,recvtype,iroot,icomm,ierr)
```

- reverse operation of `MPI_GATHER` (source)

Data Reduction Operation




```
MPI_REDUCE(sendbuff,recvbuff,icount,dtype,
           operation,iroot,icomm,ierr)
```

- `operation` = `MPI_SUM,...` (source)

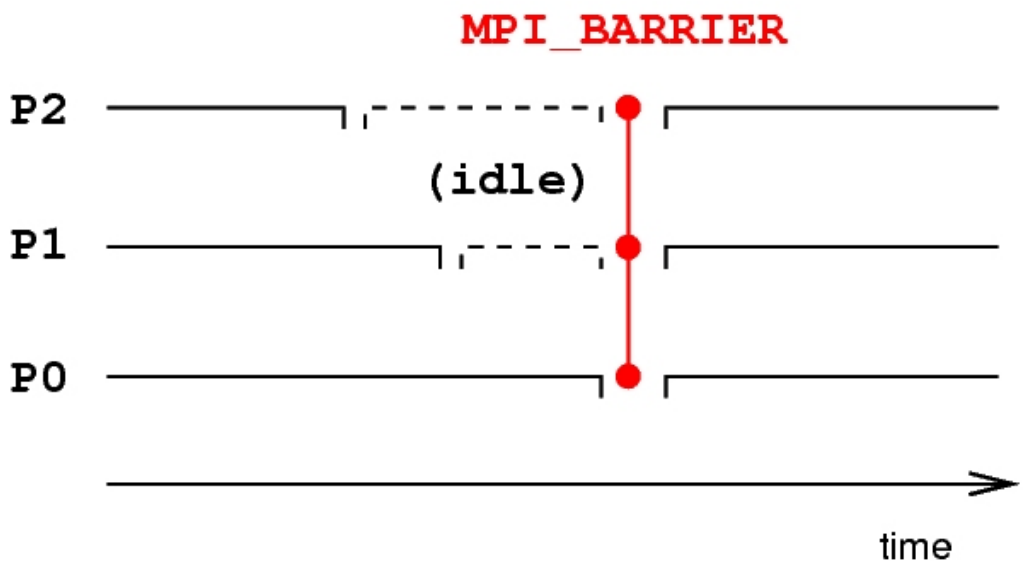
Data Reduction Operation – continued

Predefined reduction operations

- `MPI_MAX, MPI_MIN` — int; int, real
- `MPI_SUM` — int; int, real, complex
- `MPI_PROD` — int; int, real, complex
- `MPI_LAND, MPI_LOR, MPI_LXOR` — int; logical
- `MPI_BAND, MPI_BOR, MPI_BXOR` — int; int, byte
- `MPI_MINLOC, MPI_MAXLOC` — int; int, real

  $\rightsquigarrow$ returns max/min & location (rank) in special datatype

Barrier Synchronization



`MPI_BARRIER(icomm,ierr)`

- completion when all processes have synchronized

$\Rightarrow$ avoid if possible! (useful for debugging)

Collective Communication vs. Redundant Computation

Treatment of sequential sections of code

sequential work	(e.g. pre-processing of matrix entries)
parallel work	(e.g. solving linear system, multiple rhs)

Non-redundant scheme

`M:`	sequential work
`M:`	broadcast results
`ALL:`	parallel work

Redundant scheme

`ALL:`	sequential work
`ALL:`	parallel work

$\Rightarrow$ redundancy can be beneficial!

Example: Computation of π

- $\pi = \int_0^1 \frac{4}{1+x^2}\,dx \approx \sum_{i=1}^{N} \frac{4\Delta x}{1+x_i^2}$
- approximated by rectangular integration (sequential code)

Parallel algorithm

1. 'master': input and broadcast number of intervals
2. each processor computes partial sum
3. 'master' process collects the global result

   (parallel code)