Master in High Performance Computing

PARALLEL PROGRAMMING II

Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

The MPI_BARRIER

Blocks until all processes have reached this routine

```
INCLUDE 'mpif.h'

MPI_BARRIER(COMM, IERROR)

 INTEGER    COMM, IERROR
```

Ivan Girotto
igirotto@ictp.it

MPI DATA TYPES

MPI datatype handle	C datatype
MPI_INT	int
MPI_SHORT	short
MPI_LONG	long
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_CHAR	char
MPI_BYTE	unsigned char

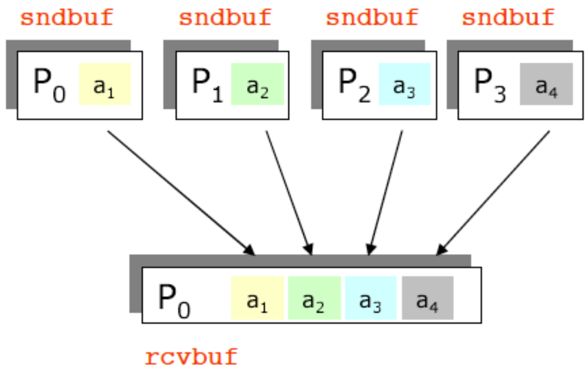
Calling MPI_REDUCE

MPI_REDUCE(in, out, count, type, op, receiver, comm, err)

in:	data to be sent (from all)
out:	storage for reduced data (on receiver)
count:	number of data items to be reduced
type:	type (=size) of data items
op:	reduction operation, e.g. **MPI_SUM**
receiver:	rank of sending processor of data
communicator:	group identifier, **MPI_COMM_WORLD**
err:	error status or **MPI_SUCCESS**

Master in High Performance Computing

MPI_Gather



One-to-all communication: different data collected by the root process, from all others processes in the communicator.

It is the opposite of Scatter

MPI_GATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)

[IN sendbuf] starting address of send buffer (choice)

[IN sendcount] number of elements in send buffer (integer)

[IN sendtype] data type of send buffer elements (handle)

[OUT recvbuf] address of receive buffer (choice, significant only at root)

[IN recvcount] number of elements for any single receive (integer, significant only at root)

[IN recvtype] data type of recv buffer elements (significant only at root) (handle)

[IN root] rank of receiving process (integer)

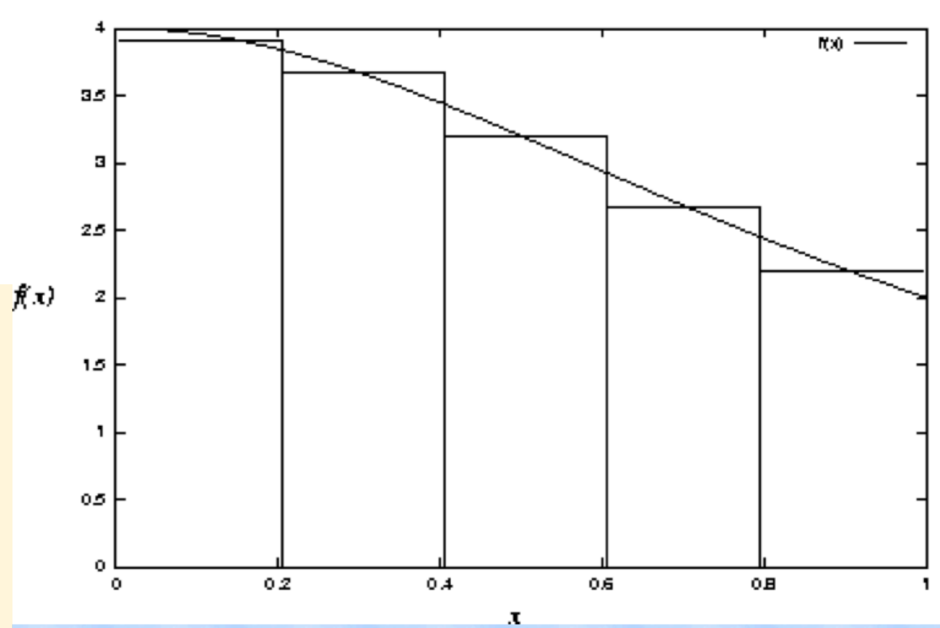
[IN comm] communicator (handle)

Ivan Girotto
igirotto@ictp.it

Approximate PI Using MPI collectives

$$\int_0^1 \frac{1}{1+x^2}dx = arctan(x)\Big|_0^1 = arctan(1) - arctan(0) = arctan(1) = \frac{\pi}{4}$$

$$\pi = 4\int_0^1 \frac{1}{1+x^2}dx$$



Integrate, i.e determine area under function numerically using slices of h * f(x) at midpoints

```
#include <stdio.h>

int main(){
 long n , i ;
 double  w,x,sum,pi,f,a;

 n = 100000000;
 w = 1.0/n;
 sum = 0.0;

 for ( i = 1 ; i <= n ; i++ ) {
     x = w * (i - 0.5);
     sum = sum + (4.0 / (1.0 + x * x ) );
  }

 pi = w * sum ;
 printf("Value of pi: %.16g\n", pi);

 return 0;
}
```

Assignment

1) Implement the PI approximation in parallel using the Message Passing paradigm

STANDARD BLOCKING SEND - RECV

MPI_SEND(buf, count, type, dest, tag, comm, ierr)

MPI_RECV(buf, count, type, dest, tag, comm, status, ierr)

Buf array of MPI type **type**.

Count (INTEGER) number of element of **buf** to be sent/recv

Type (INTEGER) MPI type of **buf**

Dest (INTEGER) rank of the destination process

Tag (INTEGER) number identifying the message

Comm (INTEGER) communicator of the sender and receiver

*** Status** (INTEGER) array of size **MPI_STATUS_SIZE** containing communication status information

Ierr (INTEGER) error code

**** used only for receive operations***

Wildcards

Both in Fortran and C **MPI_RECV** accept wildcard:

- To receive from any source: MPI_ANY_SOURCE
- To receive with any tag: MPI_ANY_TAG
- Actual source and tag are returned in the receiver's status parameter => status.MPI_SOURCE, status.MPI_TAG

MPI_GET_COUNT(status, datatype, count)

[IN status] return status of receive operation (Status)

[IN datatype] datatype of each receive buffer entry (handle)

[OUT count] number of received entries (integer)

```
PROGRAM send_recv
    INCLUDE 'mpif.h'
    INTEGER :: ierr, myid, nproc, status(MPI_STATUS_SIZE)
    REAL A(2)
    CALL MPI_INIT(ierr)
    CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
    CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
    IF( myid .EQ. 0 ) THEN
        A(1) = 3.0
        A(2) = 5.0
        CALL MPI_SEND(A, 2, MPI_REAL, 1, 10, MPI_COMM_WORLD, ierr)
    ELSE IF( myid .EQ. 1 ) THEN
        CALL MPI_RECV(A, 2, MPI_REAL, 0, 10, MPI_COMM_WORLD, status, ierr)
        WRITE(6,*) myid,': a(1)=',a(1),' a(2)=',a(2)
    END IF
    CALL MPI_FINALIZE(ierr)
END
```

NON-BLOCKING SEND - RECV

MPI_ISEND(buf, count, type, dest, tag, comm, request, ierr)

MPI_IRECV(buf, count, type, dest, tag, comm, request, ierr)

Buf array of MPI type **type**.

Count (INTEGER) number of element of **buf** to be sent/recv

Type (INTEGER) MPI type of **buf**

Dest (INTEGER) rank of the destination process

Tag (INTEGER) number identifying the message

Comm (INTEGER) communicator of the sender and receiver

Request (INTEGER) request handler, used for checking the communication status

Ierr (INTEGER) error code

No-Blocking Checkpoint

MPI_WAIT(request, status, ierr)

Request (INTEGER) request handler, used for checking the communication status

Status (INTEGER) array of size **MPI_STATUS_SIZE** containing communication status information

Ierr (INTEGER) error code

Wait until the communication handled by the object request is terminated. For test only use MPI_TEST, for checkpoint of many communication use MPI_WAITALL

Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

MHPC
Master in High Performance Computing

Assignments

1) Implement data exchange between two processes. Perform a test using 100 elements and 100000000 elements.

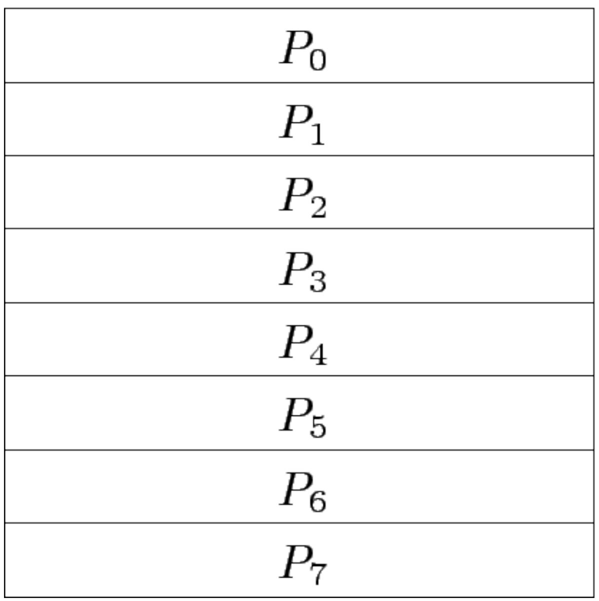
SISSA
Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

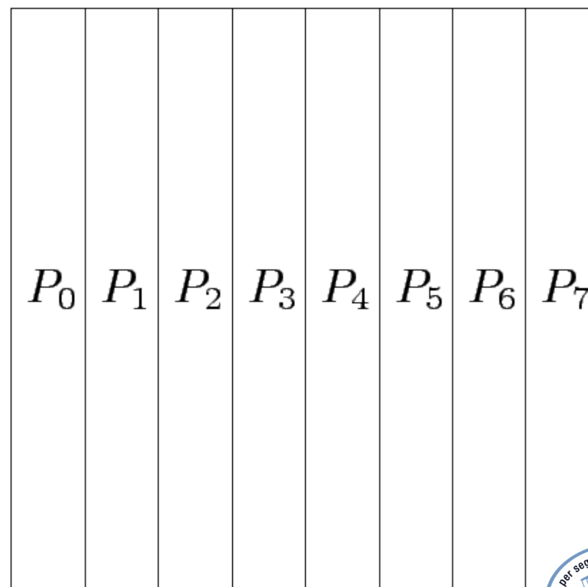
Static Data Partitioning

The simplest data decomposition schemes for dense matrices are 1-D block distribution schemes.

row-wise distribution



column-wise distribution



Master in High Performance Computing

Distributed Data Vs Replicated Data

- Replicated data distribution is useful if it helps to reduce the communication among process at the cost of bounding scalability
- Distributed data is the ideal data distribution but not always applicable for all data-sets
- Usually complex application are a mix of those techniques

SISSA
Scuola Internazionale Superiore di Studi Avanzati

The Abdus Salam International Centre for Theoretical Physics

Global Vs Local Indexes

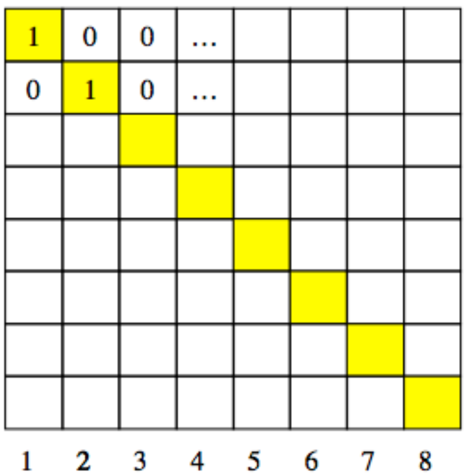
In sequential code you always refer to global indexes

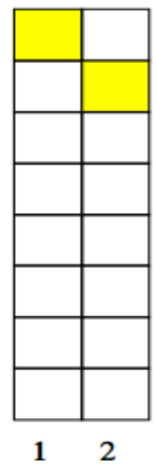
With distributed data you must handle the distinction between global and local indexes (and possibly implementing utilities for transparent conversion)

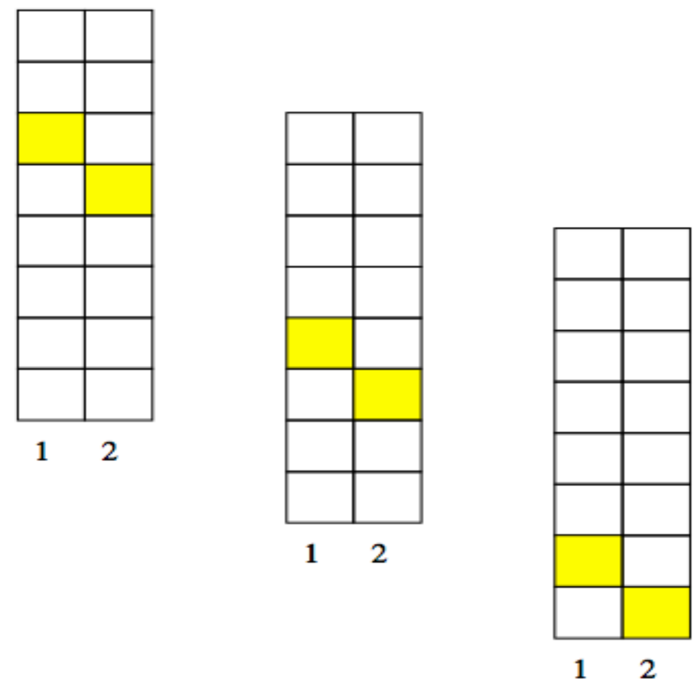
Local Idx	1	2	3	1	2	3	1	2	3
Global Idx	1	2	3	4	5	6	7	8	9

Master in High Performance Computing

Collaterals to Domain Decomposition /1



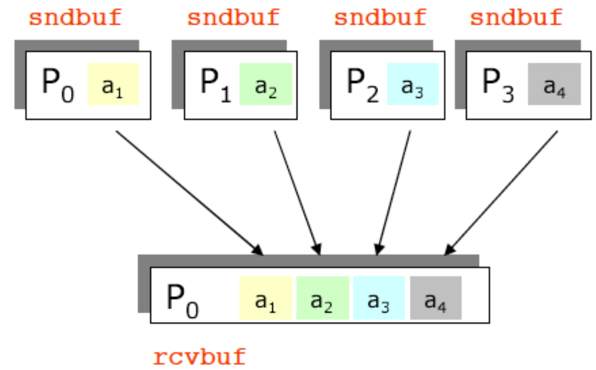




Are all the domain's dimensions always multiple of the number of tasks/processes we are willing to use?

Master in High Performance Computing

MPI_Gatherv



One-to-all communication: different data collected by the root process, from all others processes in the communicator.

Messages can have different sizes and displacements

MPI_GATHERV(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm)

[IN sendbuf] starting address of send buffer (choice)

[IN sendcount] number of elements in send buffer (integer)

[IN sendtype] data type of send buffer elements (handle)

[OUT recvbuf] address of receive buffer (choice, significant only at root)

[IN recvcounts] integer array (of length group size) containing the number of elements that are received from each process (significant only at root)

[IN displs] integer array (of length group size). Entry i specifies the displacement relative to recvbuf at which to place the incoming data from process i (significant only at root)

[IN recvtype] data type of recv buffer elements (significant only at root) (handle)

[IN root] rank of receiving process (integer)

[IN comm] communicator (handle)

Assignments

1) Initialize the Identity Matrix of size *N x N* on distribute data. Consider to implement the option of printing the matrix on standard output, if a DEBUG mode is activated and the Matrix size <= 10. Otherwise print the Matrix on a binary file.
2) Work initially on replicated data. Then optimize the solution for distributed data

External MPI Resources

Here are some links to tutorials and literature:

CI-Tutor at NCSA: http://www.citutor.org/

MPI reference and mini tutorial at LLNL:

http://computing.llnl.gov/tutorials/mpi/

Designing and Building // Programs, by Ian Foster:

http://www.mcs.anl.gov/~itf/dbpp/

MPI standards: http://www.mpi-forum.org/

OpenMPI: http://www.open-mpi.org

MPICH: http://www.mcs.anl.gov/research/projects/mpich2