Mai fidarsi

```
int main()
{
  int a,i=2;
  a = -1*abs(i-1);
  printf ( "%d\n", a);
}

$ gcc -W -Wall -o first main.c
$ ./first
1

$ gcc -fno-builtin -o second main.c
$ ./second
-1
```

compilatore dipendente

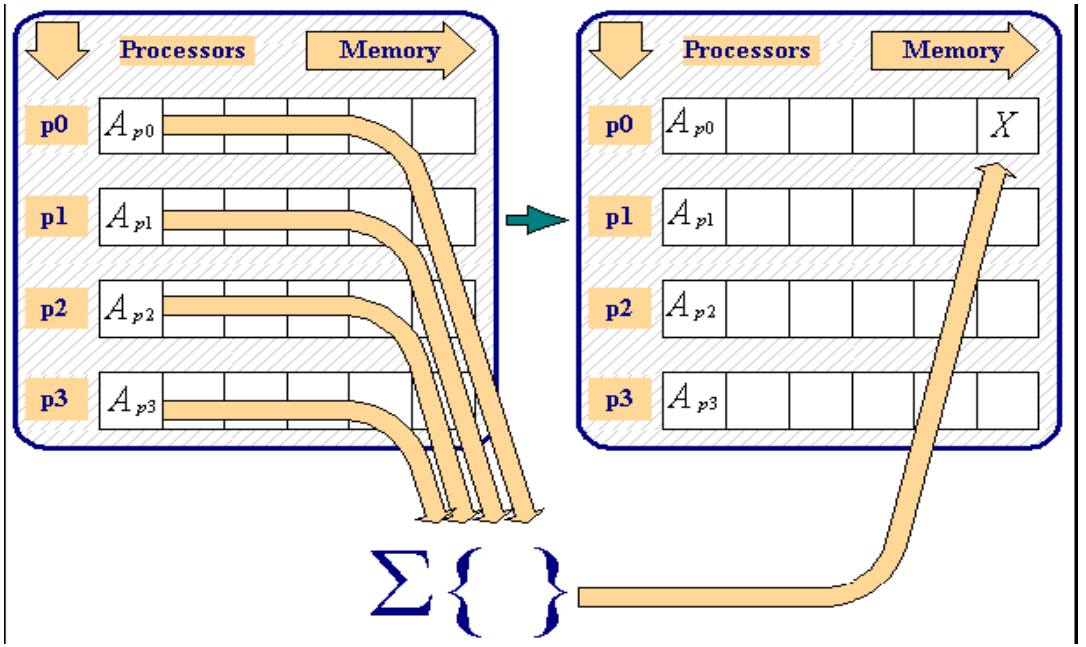
Bcast

Vediamo la soluzione del broadcast **22_bcasttree.**

MPI_Reduce

E' un'operazione All to One. La funzionalita' REDUCE consente di:

- Raccogliere da ogni processo i dati provenienti dalla medesima locazione di memoria
- Ridurre i dati ad un solo valore attraverso un operatore associativo (la somma in figura)
- Salvare il risultato in una locazione di memoria del processo root (p0 in figura)



MPI_Reduce

```
int MPI_Reduce(void* sbuf, void* rbuf, int count,
MPI_Datatype dtype, MPI_Op op, int root,MPI_Comm comm)
```

- [IN] sbuf e' l'indirizzo del send buffer
- [OUT] rbuf e' l'indirizzo del receive buffer
- [IN] count e' di tipo int e contiene il numero di elementi del send/receive buffer
- [IN] dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del send/receive buffer
- [IN] op e' di tipo MPI_Op e referenzia l'operatore di reduce da utilizzare
- [IN] root e' di tipo int e contiene il rank del processo root della reduce
- [IN] comm e' di tipo MPI_Comm ed e' il comunicatore cui appartengono i processi coinvolti nella reduce

Operatori di riduzione

l'utente puo' definirne di nuovi, purche' associativi!

Operatore	***Descrizione***
MPI_MAX	Massimo
MPI_MIN	Minimo
MPI_SUM	Somma
MPI_PROD	Prodotto

Formalmente, un'operazione binaria * su un insieme *S* e' detta **associativa** se soddisfa la **legge associativa:**

$$(x * y) * z = x * (y * z) \quad \text{per ogni } x, y, z \in S$$

L'ordine di valutazione non influisce sul valore di tale espressione, e si dimostra che lo stesso vale per le espressioni che contengono un numero arbitrario di operazioni * .

MPI_Op_create

```
int MPI_Op_create(MPI_User_function *function, int
commute, MPI_Op *op )
```

La “calling list” per la “user function” e':

```
typedef void (MPI_User_function) ( void * a, void * b,
int * len, MPI_Datatype * );
```

dove l'operazione e' `b[i] = a[i] op b[i]`, per `i=0,...,len-1`. Alla “user function” viene passato un pointer al datatype che e' stato passato a MPI_Reduce o etc...

```
int MPI_Op_free( MPI_Op *op);
```

MPI_Op_create

```
void addem(int *invec, int *inoutvec, int *len,
     MPI_Datatype *dtype)
{
    int i;
    for ( i=0; i<*len; i++ )
        inoutvec[i] += invec[i];

    return;
}
```

MPI_Allreduce

```
int MPI_Allreduce(void* sbuf, void* rbuf, int count,
    MPI_Datatype dtype, MPI_Op op, MPI_Comm comm)
```

- [IN] sbuf e' l'indirizzo del send buffer
- [OUT] rbuf e' l'indirizzo del receive buffer
- [IN] count e' di tipo int e contiene il numero di elementi del send/receive buffer
- [IN] dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del send/receive buffer
- [IN] op e' di tipo MPI_Op e referenzia l'operatore di reduce da utilizzare
- [IN] comm e' di tipo MPI_Comm ed e' il comunicatore cui appartengono i processi coinvolti nella reduce

Combina gli elementi presenti nel buffer di input di ogni processo secondo l'operatore associativo prescelto, per poi copiare il risultato nel buffer di output di tutti i processi (MPI_Reduce + MPI_Bcast)

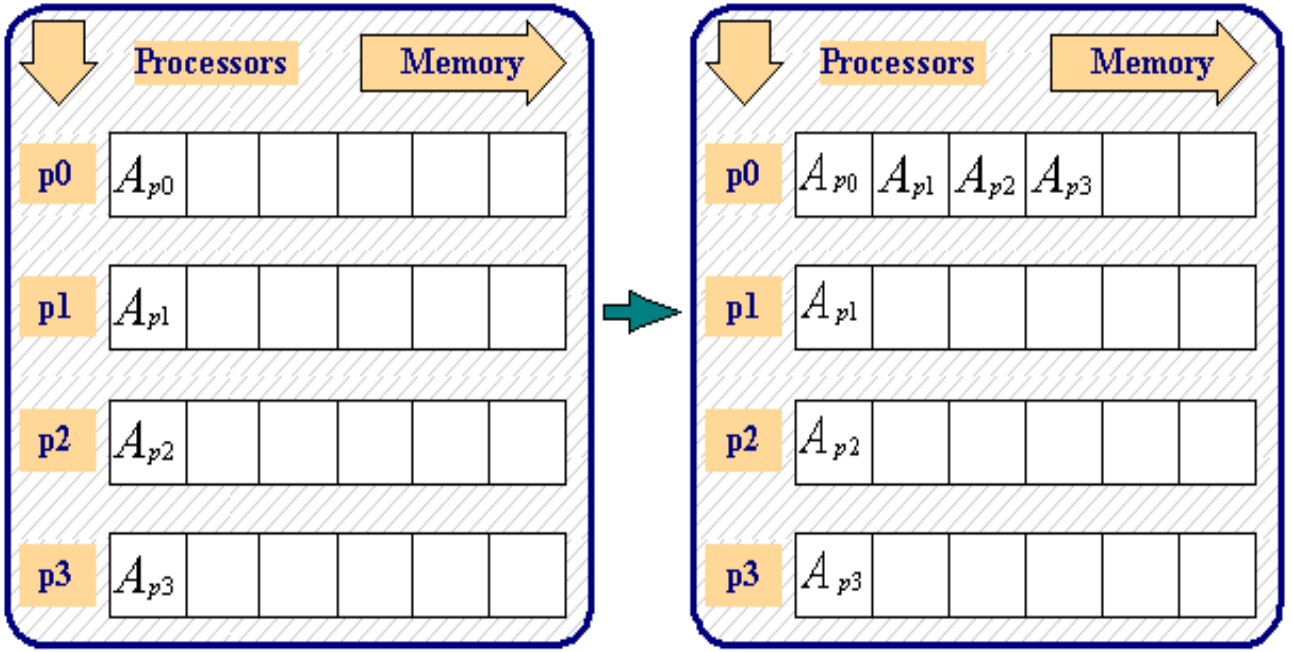
Esempio (21_b_banda): dimostrazione dell'uso di MPI_Op_create (non e' un metodo corretto per la misura della banda)

Esercizio (23_pi): riscrivere il calcolo di PI usando comunicazioni collettive

MPI_Gather

E' un'operazione All to One. Con la funzionalita' GATHER ogni processo (incluso il root) invia il contenuto del proprio send buffer al processo root

Il processo root riceve i dati e li ordina in funzione del rank del processo sender (vedi **22_pimpi**)



MPI_Gather

```
int MPI_Gather(void* sbuf, int scount, MPI_Datatype s_dtype,
void* rbuf, int rcount, MPI_Datatype r_dtype, int root, MPI_Comm
comm)
```

- [IN] sbuf e' l'indirizzo del send buffer
- [IN] scount e' di tipo int e contiene il numero di elementi del send buffer
- [IN] s_dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del send buffer
- [OUT] rbuf e' l'indirizzo del receive buffer
- [IN] rcount e' di tipo int e contiene il numero di elementi di ogni singolo receive (ad esempio uso di `MPI_Type_...`)
- [IN] r_dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del receive buffer
- [IN] root e' di tipo int e contiene il rank del processo root della gather
- [IN] comm e' di tipo MPI_Comm ed e' il comunicatore cui appartengono i processi coinvolti nella gather

MPI_Gatherv

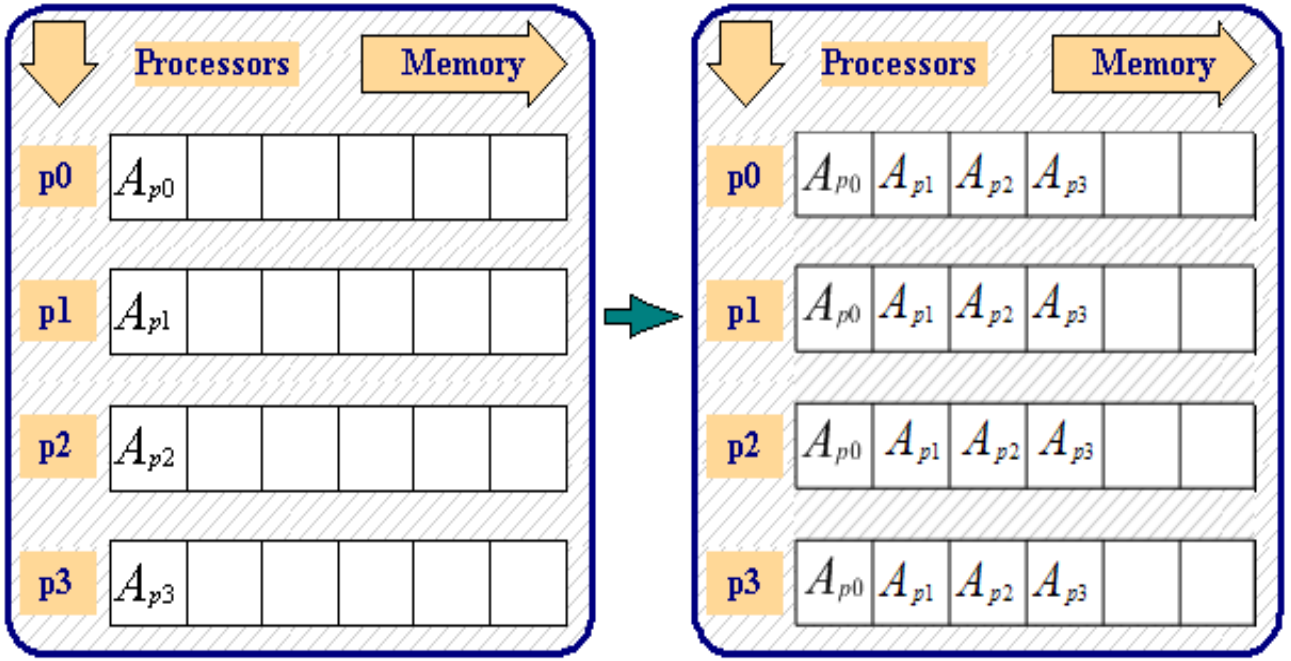
```
int MPI_Gatherv(void* sbuf, int scount, MPI_Datatype s_dtype, void* rbuf
int *rcounts, int *displs, MPI_Datatype r_dtype, int root, MPI_Comm comm
```

- [IN] sbuf e' l'indirizzo del send buffer
- [IN] scount e' di tipo int e contiene il numero di elementi delsend buffer
- [IN] s_dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del send buffer
- [OUT] rbuf e' l'indirizzo del receive buffer
- [IN] rcounts e' un array di tipo int e contiene il numero di elementi da ricevere da ogni processo, ordinati perrank
- [IN] displs e' un array di tipo int e contiene i displacement nell'array rbuf da cui scrivere i dati provenienti da ogni processo, ordinati per rank
- [IN] r_dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del receive buffer
- [IN] root e' di tipo int e contiene il rank del processo root della gather
- [IN] comm e' di tipo MPI_Comm ed e' il comunicatore cui appartengono i processi coinvolti nellagather

Operazione All to One. Consente un'operazione di tipoGATHER in cui ogni processo possa inviare al processoroot un set di dati di dimensione diversa

MPI_Allgather

E' un'operazione All to All. Di fatto e' l'equivalente di un'operazione di GATHER, il cui processo root poi esegua un BROADCAST. E' molto piu' conveniente ed efficiente eseguire una operazione ALL_GATHER piuttosto che la sequenza GATHER+BROADCAST



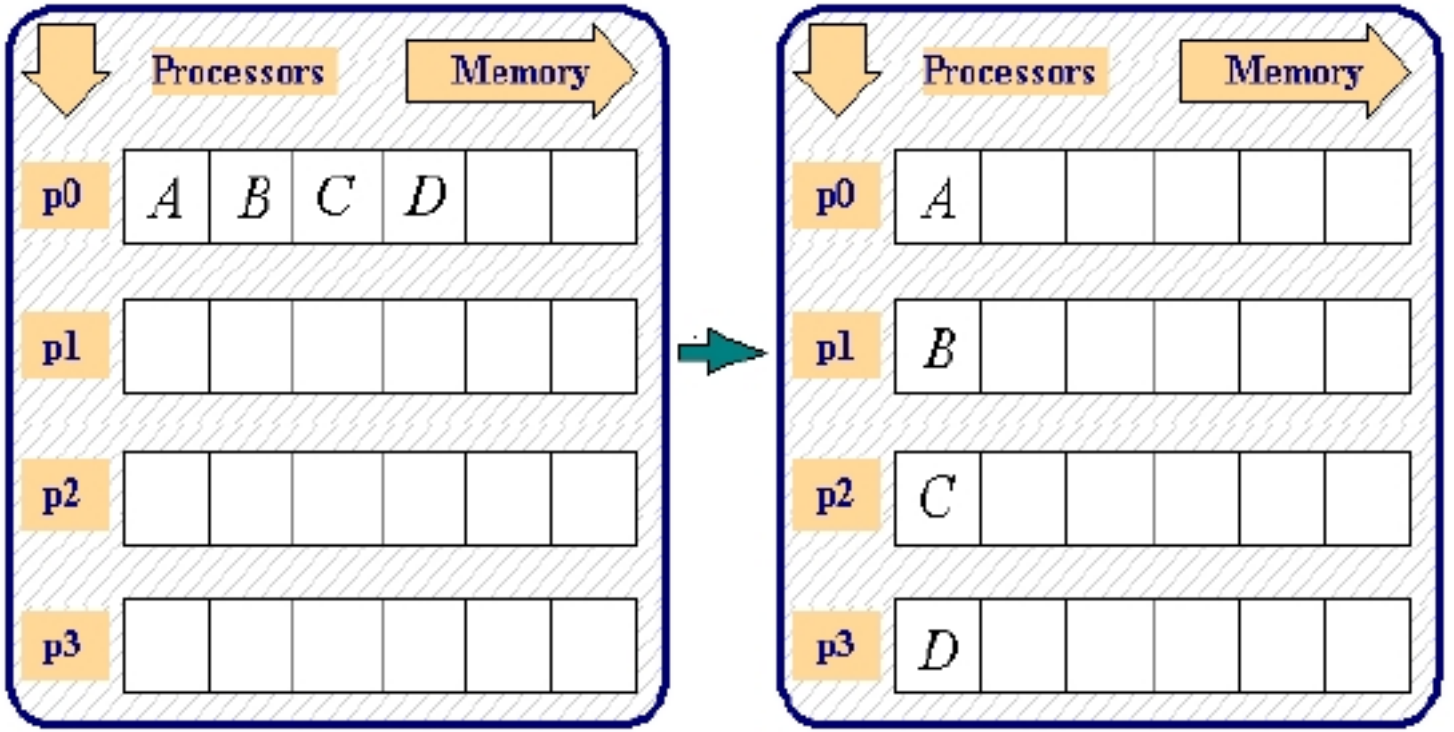
MPI_Allgather

```
int MPI_Allgather(void* sbuf, int scount, MPI_Datatype
s_dtype, void* rbuf, int rcount, MPI_Datatype r_dtype,
MPI_Comm comm)
```

- [IN] sbuf e' l'indirizzo del send buffer
- [IN] scount e' di tipo int e contiene il numero di elementi del send buffer
- [IN] s_dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del send buffer
- [OUT] rbuf e' l'indirizzo del receive buffer
- [IN] rcount e' di tipo int e contiene il numero di elementi del receive buffer
- [IN] r_dtype w' di tipo MPI_Datatype e descrive il tipo di ogni elemento del receive buffer
- [IN] comm e' di tipo MPI_Comm ed e' il comunicatore cui appartengono i processi coinvolti nella allgather

MPI_Scatter

- E' un'operazione One to All
- Il processo root (p0 nella figura)
  - Divide in N parti uguali un insieme di dati contigui in memoria
  - Invia una parte ad ogni processo in ordine di rank



MPI_Scatter

```
int MPI_Scatter(void* sbuf, int scount, MPI_Datatype s_dtype,
void* rbuf, int rcount, MPI_Datatype r_dtype, int root, MPI_Comm
comm)
```

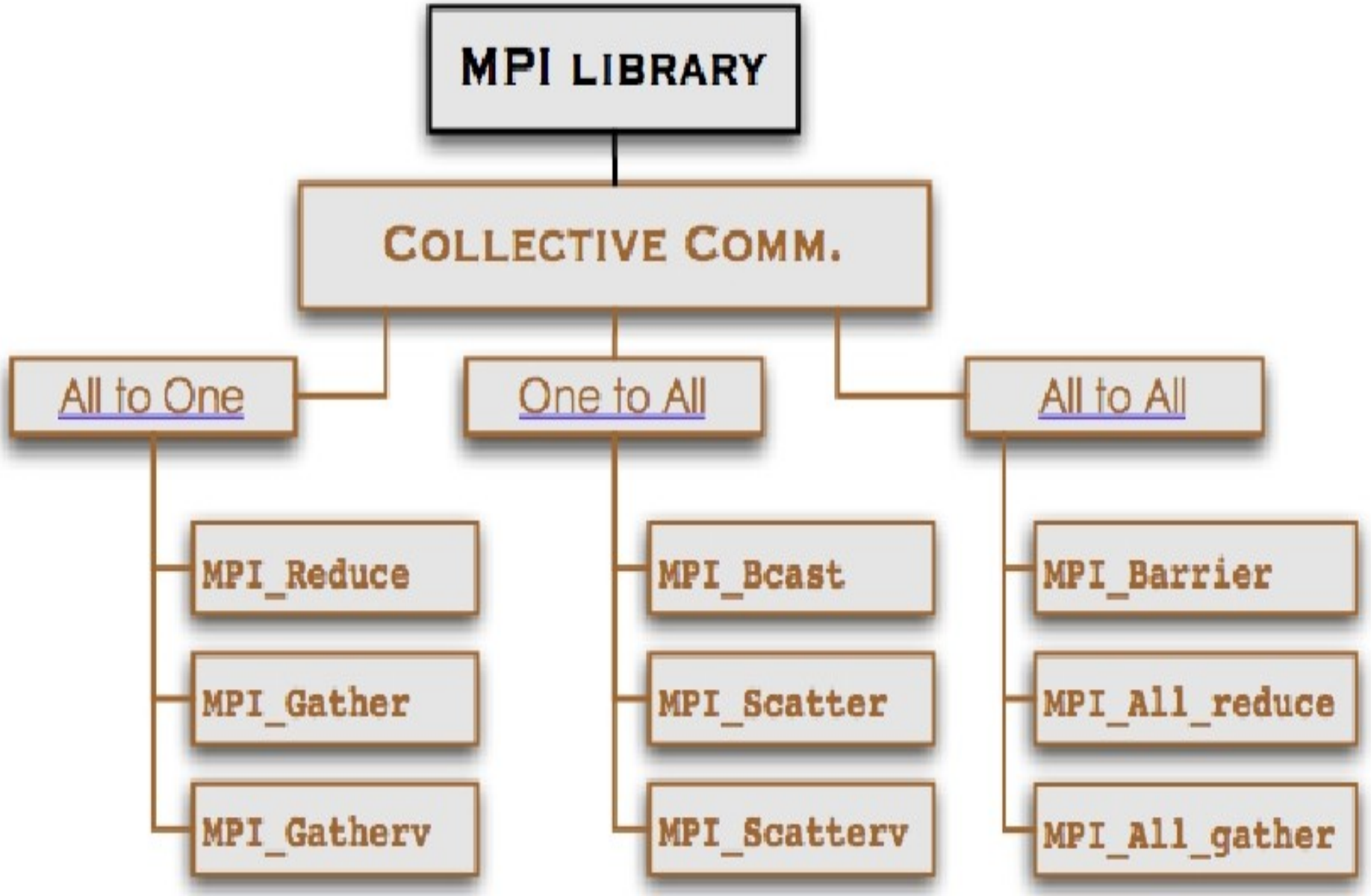
- [IN] sbuf e' l'indirizzo del send buffer
- [IN] scount e' di tipo int e contiene il numero di elementi del send buffer (numero di elementi da spedire ad ogni processo)
- [IN] s_dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del send buffer
- [OUT] rbuf e' l'indirizzo del receive buffer
- [IN] rcount e' di tipo int e contiene il numero di elementi del receive buffer
- [IN] r_dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del receive buffer
- [IN] root e' di tipo int e contiene il rank del processo root della scatter
- [IN] comm e' di tipo MPI_Comm ed w' il comunicatore cui appartengono i processi coinvolti nella scatter

Esercizio (24_matrixscatter): provate a fare lo scatter di una matrice riga per riga

MPI_Scatterv

```
int MPI_Scatterv(void* sbuf, int *scounts, int *displs, MPI_Datatype
s_dtype, void* rbuf, int rcount, MPI_Datatype r_dtype, int root, MPI_Comm
comm)
```

- [IN] sbuf e' l'indirizzo del send buffer
- [IN] scounts e' un array di tipo int e contiene il numero di
- elementi da inviare ad ogni processo, ordinati per rank
- [IN] displs e' un array di tipo int e contiene e contiene i displacement nell'array sbuf da cui partono i dati da inviare ad ogni processo, ordinati per rank
- [IN] s_dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del send buffer
- [OUT] rbuf e' l'indirizzo del receive buffer
- [IN] rcount e' di tipo int e contiene il numero di elementi del receive buffer
- [IN] r_dtype e' di tipo MPI_Datatype e descrive il tipo di ogni elemento del receive buffer
- [IN] root e' di tipo int e contiene il rank del processo root della gather
- [IN] comm e' di tipo MPI_Comm ed e' il comunicatore cui appartengono i processi coinvolti nella gather


MPI LIBRARY
COLLECTIVE COMM.
All to One
One to All
All to All
MPI_Reduce
MPI_Gather
MPI_Gatherv
MPI_Bcast
MPI_Scatter
MPI_Scatterv
MPI_Barrier
MPI_All_reduce
MPI_All_gather

Laboratorio

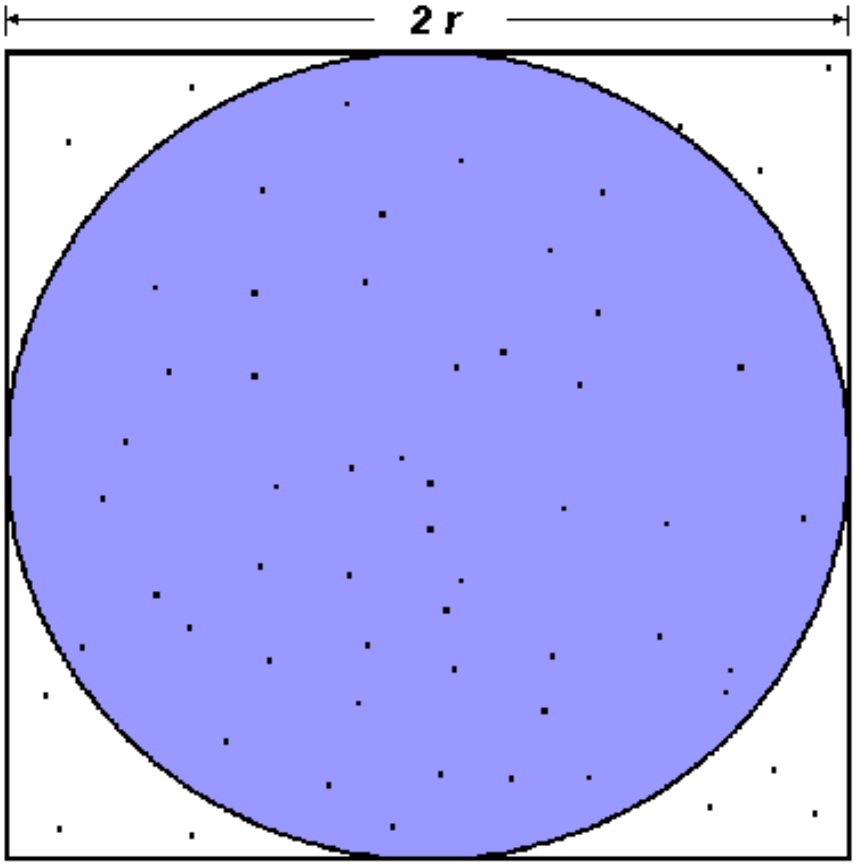
Esercizi di parallelizzazione, calcolare speedup, tempi di calcolo e tempi di comunicazione:

1 – calcolo di PI

2 – prodotto matrice vettore

3 – prodotto matrice matrice

PI – metodo Monte Carlo



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$