NTNU – Trondheim
Norwegian University of
Science and Technology

OpenMP Part 2.

Agenda:
Parallel sections

Tasks

Hybrid programming

Parallel sections

The section worksharing construction gives a different structured block to each thread.

Example: 2 threads (c and fortran).

```
#pragma omp parallel                    !$OMP PARALLEL
{
   #pragma omp sections                 !$OMP SECTIONS
   {
     #pragma omp section                !$OMP SECTION
         calculate_x ( );               call calculate_x ()
     #pragma omp section                !$OMP SECTION
         calculate_y ( );               call calculate_y ()
   }                                    !$OMP END SECTIONS
}                                       !$OMP END PARALLEL
```

Note! By default, there is a barrier at end of ”omp section”. Use the “nowait” clause to turn off the barrier.

Example: Reduction and private

```
double sum, t;
#pragma omp parallel
{
   sum=0;  t=1;
   #pragma omp sections firstprivate (t) reduction (+:sum)
   {
     #pragma omp section
         sum=calculate_x ( t );
     #pragma omp section
         sum=calculate_y ( t );
   }
}
```

Note that the reduction and firstprivate also be set after “omp parallel“

NTNU – Trondheim
Norwegian University of
Science and Technology

Exercise 1 (sec_helloworld.c)

Idun: /cluster/shared/floan/tutorials/

Helloworld. Create 4 sections and print out “Hello world from thread no 1“ etc.

Idun: module load intel/2023b (only once)

make sec_helloworld

sbatch sec_helloworld_c.job (or _f fortran)

Exercise 2 (section.c)

Modify the sections.c program, and split up the for-loop to 2 sections (threads).

Run:

make sections

sbatch sections_c.job (or _f fortran)

(Mac pc: If error when compiling, write: export LC_ALL=C)

Task

Typical use of tasks are for recursive function and while loop.

NOTE! In fortran you must end with !$OMP END TASK

Task construct

```
#pragma omp task [clauses]
        Structured-block
```

where clause can be one of:

if (expression)

untied

shared (list)

private (list)

firstprivate (list)

default(shared | none)

NTNU – Trondheim
Norwegian University of
Science and Technology

Example: Linked list

```
#pragma omp parallel
{
    #pragma omp single private (p)
    {
  p = head;
  while (p != NULL)
  {
#pragma omp task // p is first private inside task
    process(p);

p=p->next;
  }
   }
}
```

Variable, arrays or pointers are firstprivate inside a task directive.

If variables, arrays or pointers are shared before a task,

there are also shared inside a task directive.

Example: Task data scoping

```
int a;
void myfunc() {
    int b,c,d;
   #pragma omp parallel private (c) shared(d)
    {
         int e;
        #pragma omp task
         {
            int f;
a is ? (data clause:shared, private, firstprivate)
b is ?
            c is ?
            d is ?
            e is ?
f is
–    }}}
```

Example: Task data scoping

```
int a
void myfunc() {
    int b,c,d;
   #pragma omp parallel private (c) shared(d)
    {
         int e;
        #pragma omp task
         {
            int f;
```

a is shared

b is shared

c is firstprivate

d is shared

e is firstprivate

f is private

– }}}

NTNU – Trondheim
Norwegian University of
Science and Technology

Task synchronization (taskwait)

All children tasks are spread to individual thread and core, and to be sure that all tasks are finished at same time; use taskwait.

Example

```
#pragma omp parallel
{
    #pragma omp single
    {
            #pragma omp task
            res1 = func1();
            #pragma omp task
            res2 = func2();
            #pragma omp taskwait
            sum = res1 + res2;
    }
}
```

Exercise 1. Task_array

Modify the program task_array.c (or .f90) with parallel tasks.

To run the program (use taskq)

make task_array

sbatch task_array_c.job (or _f.job)

Exercise 2. Linked list.

Modify the program task_linkedlist.c (or .f90) with parallel tasks.To run the program

make task_linkedlist

sbatch task_linkedlist_c.job (or _f.job)

Exercise 3: Fibonacci (Advanced)

Fibonacci:

$f(0) = 0$, $f(1) = 1$,

For $n > 1$, $f(n) = f(n-1) + f(n-2)$

Sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

To run the program

make task_fib

sbatch task_fib_c.job (or f_.job)

1. Modify the main program and the rec_fib with OpenMP directives.

2. Home work: Check the performance with omp_get_wtime. Do you see any improvement?

Hybrid Programming

In this section we shall look at the hybrid MPI and OpenMP programming.

-**OpenMP:** Multicore shared memory system.

-**MPI:** Message Passing between nodes in a cluster (Note! You can also have message passing between cores)

For MPI programming; you work on several node in same time, and you must switch between this nodes in your mind when you programming. ("I am now working on the node 1, and now I working on node 2 etc")

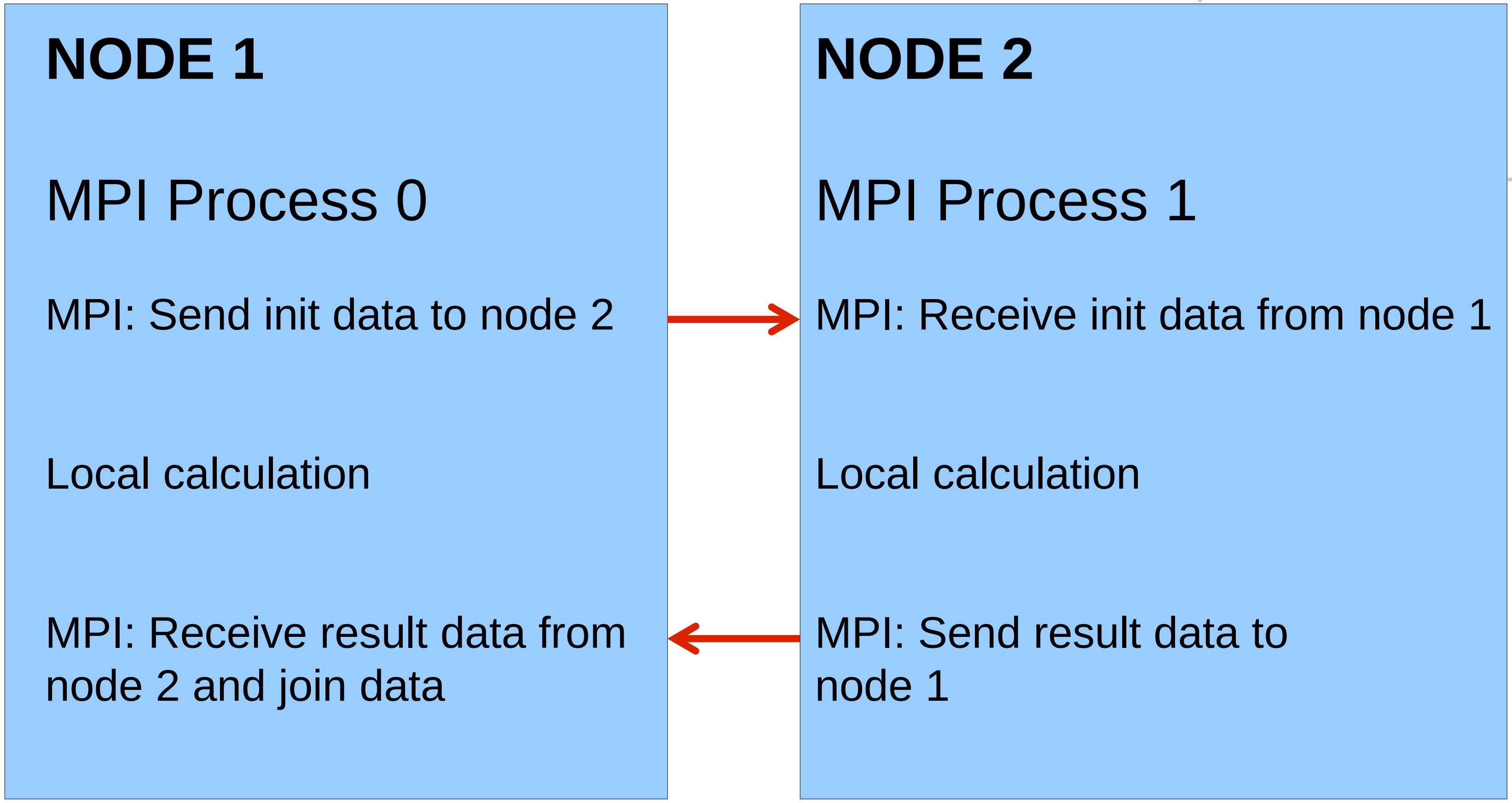
(**MPI: Message Passing Interface**)

www.mpi-forum.org/

openmp.org

www.cs.usfca.edu/~peter/ppmpi/

Example: MPI program



NODE 1

MPI Process 0

MPI: Send init data to node 2

Local calculation

MPI: Receive result data from node 2 and join data

NODE 2

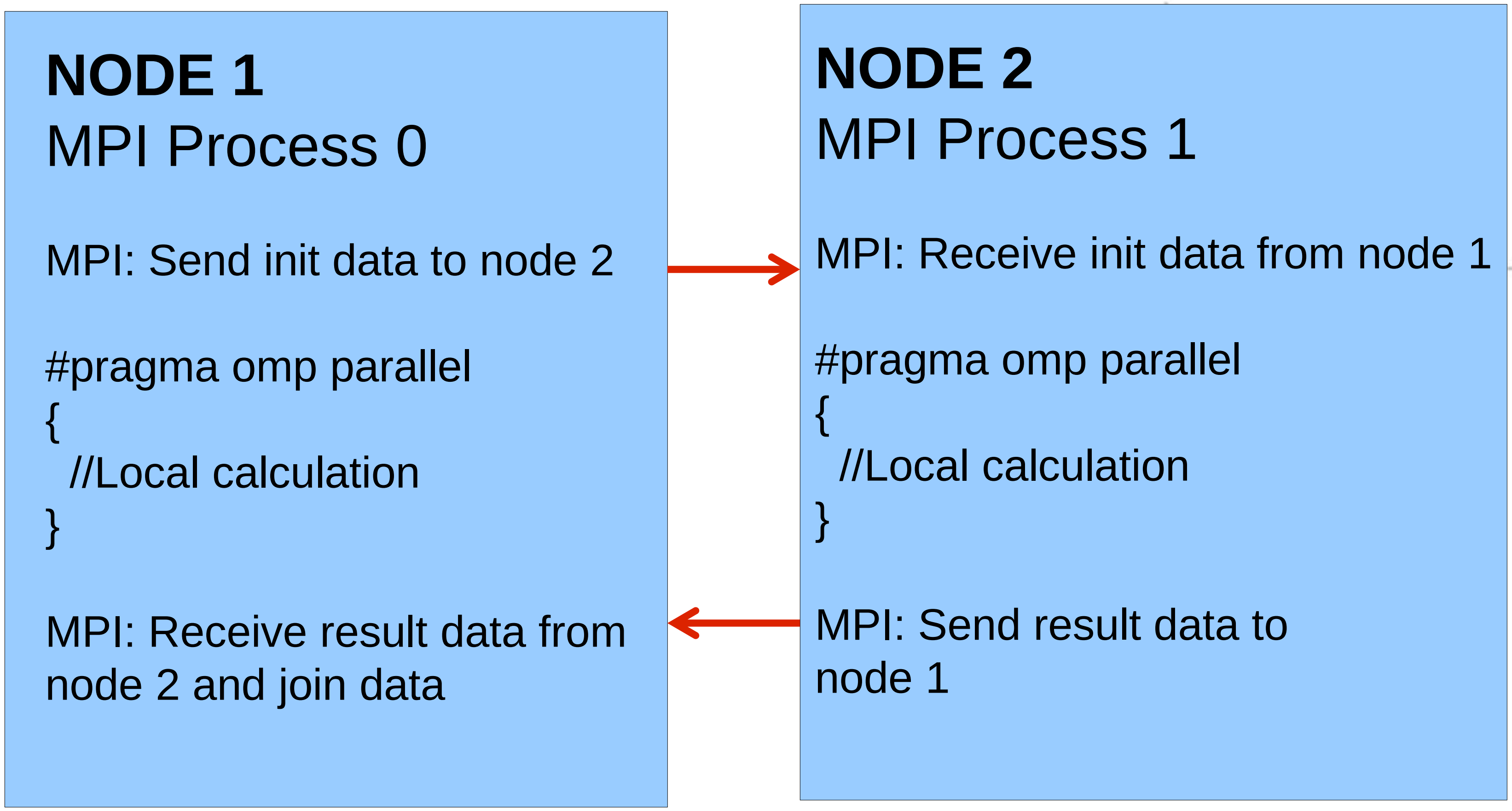
MPI Process 1

MPI: Receive init data from node 1

Local calculation

MPI: Send result data to node 1

Example: Hybrid program



NODE 1
MPI Process 0

MPI: Send init data to node 2

```
#pragma omp parallel
{
  //Local calculation
}
```

MPI: Receive result data from node 2 and join data

NODE 2
MPI Process 1

MPI: Receive init data from node 1

```
#pragma omp parallel
{
  //Local calculation
}
```

MPI: Send result data to node 1

MPI (Message Passing Interface)

MPI Initializing and Finalizing

```
void main (int argc, char * argv[])
{
// "myrank" is the individual MPI process
and "ranks" is the number of MPI
processes.
int myrank,ranks;

MPI_Init(&argc,&argv); //Parallel region starts here
MPI_Comm_size(MPI_COMM_WORLD,&ranks);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

// Your parallel program
```

NTNU – Trondheim
Norwegian University of
Science and Technology

Exercise 1. Hello world

Modify the hyb_helloworld.c program to printout

“Hello world from rank 1 and thread 1”

“Hello world from rank 1 and thread 2” …

“Hello world from rank 2 and thread 1” etc

Before compiling (only once,if you not have done this)

Idun: module load intel/2023b

Compiling

make hyb_helloworld

Submit

sbatch hyb_helloworld_c.job

Some MPI functions:

- MPI_Send and MPI_Recv
- MPI_Sendrecv
- MPI_Bcast
- MPI_Barrier
- MPI_Scatter and MPI_Gather
- MPI_Reduce

MPI Send and MPI_Recv

Send and receive message between ranks

Synopsis

int **MPI_Send** (void* buf, int count, MPI_Datatype datatype,
    int dest, int tag, MPI_Comm comm)

int **MPI_Recv**(void* buf, int count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm, MPI_Status *status)

•**buf**: buffer (write &buf if a variable)

•*(Note that the buffer must have different name if send and recv are inside same rank)*

•**count**: Number of elements in the array (set 1 if a variable)

•**datatype**: MPI datatype (MPI_INT, MPI_CHAR, MPI_DOUBLE ...)

•**source**: The receiver rank.

•**tag**: Message identifier. Extra information to the receiver (integer)

•**comm**: MPI Communicator: MPI_COMM_WORLD.

•**Status**: Receiver communication status.

Example Send and Recv (point to point communication)

```
double *buf = (double *) malloc( sizeof (double) * n);
Int source,destination;
Int myrank,ranks;
MPI_Status status;
int tag=0;
MPI_Init(&argc,&argv);
MPI_Comm_size (MPI_COMM_WORLD, &ranks);
MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
if ( myrank == 0 ) {
destination = 1;
init ( n , buff)
MPI_Send ( buf , n , MPI_DOUBLE , destination , tag ,
          MPI_COMM_WORLD);
}
else if ( myrank == 1 ) {
source=0;
    MPI_Recv( buf , n , MPI_DOUBLE , source , tag ,
               MPI_COMM_WORLD , &status);
}
```

NTNU – Trondheim
Norwegian University of
Science and Technology

Deadlock.

The program will deadlock if a program is like this:

```
if ( myrank == 0 ){ // Send and recv to/from rank 1
    MPI_Send (sendbuf, n, MPI_INT, 1, tag, MPI_COMM_WORLD);
    MPI_Recv (recvbuf, n, MPI_INT, 1, tag, MPI_COMM_WORD,stat);
}
else if ( myrank == 1 ){//Send and recv to/from rank 0
    MPI_Send (sendbuf, n, MPI_INT, 0, tag, MPI_COMM_WORLD);
    MPI_Recv (recvbuf, n, MPI_INT, 0, tag, MPI_COMM_WORD,stat);
}
```

MPI_Send is a blocking operation and have to wait to the message is completed received (MPI_Recv) from the receiver rank, before next step (and visa versa)

Deadlock.

To avoid deadlock with send and receive you can do this:

```
if (myrank == 0) { // Send and recv to/from rank 1
MPI_Send(buffS,n,MPI_INT,1,tag,MPI_COMM_WORLD);
MPI_Recv(buffR,n,MPI_INT,1,tag,MPI_COMM_WORD,&stat);
}
else if (myrank==1) {// Recv and send from/to rank 0
MPI_Recv(buffR,n,MPI_INT,0,tag,MPI_COMM_WORD,&stat);
MPI_Send(buffS,n,MPI_INT,0,tag,MPI_COMM_WORLD);
}
```

* Other way to prevent deadlook; use MPI_Isend and MPI_Sendrecv

MPI_Sendrecv (Point to point communication)

Synopsis

int **MPI_Sendrecv (**void *sendbuf , int sendcount , MPI_Datatype sendtype, int dest, int sendtag,

void *recvbuf, int recvcount, MPI_Datatype recvtype,

int source, int recvtag,

MPI_Comm comm, MPI_Status *status)

(Note! Sendrecv prevent deadlook)

Ex

```
If (rank == 0)
{
to = 1;    // Send to rank/node number
from = 1;  // Receive from rank/node number
}
else if (rank == 1)
{
to = 0;
from = 0;
}
MPI_Sendrecv(sendbuffer,n,MPI_INT,to,sendTag,
             recvbuffer, n, MPI_INT, from, recvTag,
             MPI_COMM_WORLD)
```

Example: Taken ring

Each node get message from rank before and send to next rank.

Recv from rank-1 and Send to rank+1

```
to = (rank+1) % ranks;
from = (rank + ranks -1 ) % ranks;

MPI_Sendrecv(sendbuf,size,MPI_INT, to, sendtag,
             recvbuf,size,MPI_INT, from, recvtag,
             MPI_COMM_WORLD)
```

NTNU – Trondheim
Norwegian University of
Science and Technology

•MPI_Bcast (Collective communcation)

•MPI Bcast broadcast a message to all MPI ranks.

•Synopsis

•int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
int root, MPI_Comm comm)

•root: rank of broadcast root.

•MPI_Barrier (Synchronization)

•Block all processes, to all MPI ranks have called the MPI_Barrier.

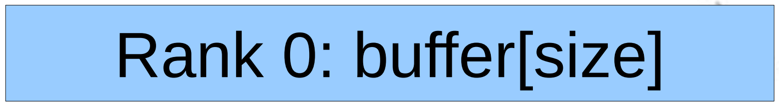
•Synopsis:

•int MPI_Barrier(MPI_Comm comm)

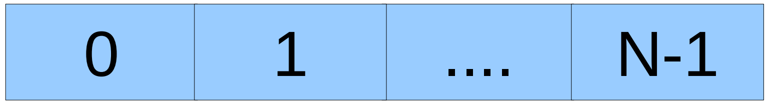
MPI_Scatter (Collective communication)

MPI_Scatter spreading a 1 dim array to all MPI processes (N ranks) as :

1.Before scattering:



2. MPI divide the array to N chunks of data:



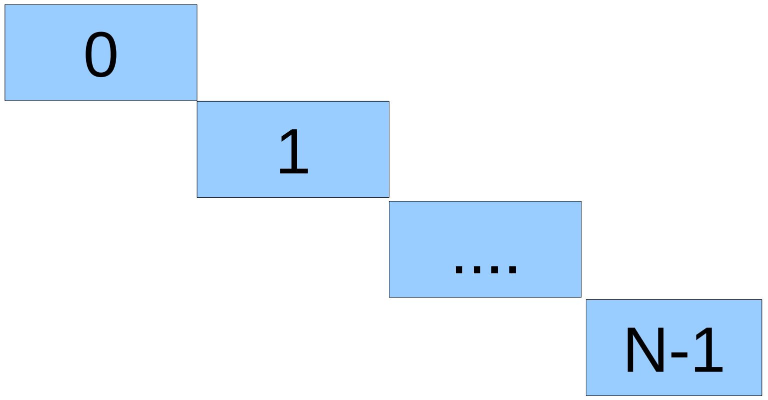
3. All ranks receive its part of the chucked array

Rank 0

Rank 1

.....

Rank N-1



Chunk size is: size/N (must be dividable)

MPI_Gether

MPI_Gather join chunks into one array as:

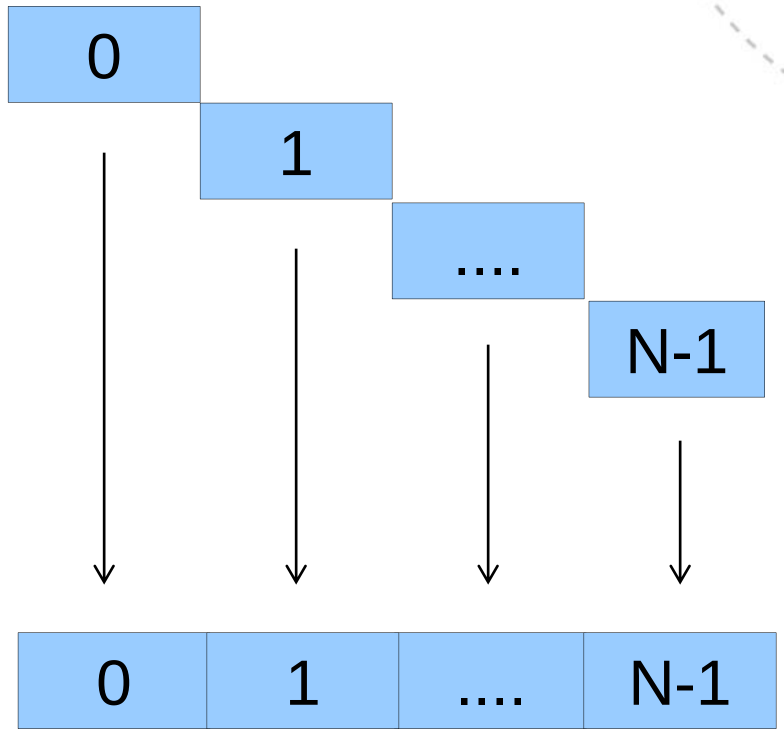
1. Before gathering:

Rank 0

Rank 1

.....

Rank N-1



2. After gathering:

MPI_Scatter and MPI_Gather

Join together values from a group of processes

Synopsis

```
int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root,
                MPI_Comm comm)
int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf, int recvcnt, MPI_Datatype recvtype,
                int root, MPI_Comm comm)
```

Scatter: Number of elements in sendbuf = Numb of el. in recvbuf * ranks.

sendcnt = recvcnt = Number of elements in recvbuf

Gather: Number of elements in recvbuf = Numb of el. in sendbuf * ranks.

sendcnt = recvcnt = Number of elements in sendbuf

NTNU – Trondheim
Norwegian University of Science and Technology

Example: Scatter and gather

Calculate: M = M * c (M is a nxm matrix and c is a constant)

```
ln = 300 ; //Local n
n = ln * ranks; // Note that ln and n must be dividable with ranks
root = 0; // Master rank
double c;
double *M; // nxm matrix
// Local M lnxm matrix
double *lM = (double *) malloc (sizeof (double) *  ln*m );
if ( myrank==0) {
c=10.0;
M = (double*) malloc ( sizeof (double) * n * m);
init(M);
}
MPI_Bcast(&c,1,MPI_DOUBLE,root,MPI_COMM_WORLD);
MPI_Scatter ( M, ln * m , MPI_DOUBLE, lM , ln * m , MPI_DOUBLE,
     root , MPI_COMM_WORLD);

for (i=0 ; i<ln*m ; i++) lM[i] *= c; // Calculation: lM = lM * c

MPI_Gather( lM , ln * m, MPI_DOUBLE, M , ln * m, MPI_DOUBLE,
root,MPI_COMM_WORLD;
```

NTNU – Trondheim
Norwegian University of
Science and Technology

MPI_Reduce

Synopsis

```
int MPI_Reduce( void *sendbuf, void *recvbuf, int count,
MPI_Datatype datatype,  MPI_Op op,  int root,
MPI_Comm comm);
```

<u>MPI reduce operators:</u>

MPI_MAX maximum

MPI_MIN minimum

MPI_SUM sum

MPI_PROD product

MPI_LAND logical and

MPI_BAND bit-wise and

MPI_LOR logical or

MPI_BOR bit-wise or

MPI_LXOR logical xor

MPI_BXOR bit-wise xor

MPI_MAXLOC max value and location

MPI_MINLOC min value and location

Example MPI_Reduce

Average of the array A.

```
...
for ( i=0; i< local_n ; i++)
local_sum += local_A[i];

MPI_Reduce( &local_sum, &global_sum, 1 , MPI_DOUBLE,  MPI_SUM ,
 MASTER_RANK , MPI_COMM_WORLD);

average = global_sum / n;
....
```

Exercise Pi.

Modify hyb_pi program with OpenMP for 2 nodes:

```
make hyb_pi
sbatch hyb_pi_c.job  (or _f for fortran)
```

NTNU – Trondheim
Norwegian University of
Science and Technology