

Today:

- MPI concepts
- Communicators, broadcast, reduce

Next week:

- MPI send and receive
- Iterative methods

Read: Class notes and references

`$CLASSHG/codes/mpi`

MPI — Message Passing Interface

OpenMP can only be used on **shared memory** systems with a single address space used by all threads.

Distributed memory systems require a different approach.

e.g. clusters of computers, supercomputers, heterogeneous networks.

MPI — Message Passing Interface

OpenMP can only be used on **shared memory** systems with a single address space used by all threads.

Distributed memory systems require a different approach.

e.g. clusters of computers, supercomputers, heterogeneous networks.

Message Passing:

SPMD model: All processors execute same program, but with different data.

Program manages memory by placing data in processes.

Data that must be shared is explicitly sent between processes.

MPI References

See the [class notes: MPI section](#).

There are several implementations of MPI available.

The VM has Open MPI installed, see www.open-mpi.org.

The Argonne National Lab version [MPICH](#) is also widely used.

See also the [MPI Standard](#)

[Standard reference book](#):

W. Gropp, E. Lusk, A. Skjellum, *Using MPI*, Second Edition, MIT Press, 1999. [link](#)

Some of my slides are from [Bill Gropp's tutorials](#)

MPI — Simple example

```
program test1
  use mpi
  implicit none
  integer :: ierr, numprocs, proc_num,

  call mpi_init(ierr)
  call mpi_comm_size(MPI_COMM_WORLD, numprocs, ierr)
  call mpi_comm_rank(MPI_COMM_WORLD, proc_num, ierr)

  print *, 'Hello from Process ', proc_num, &
    ' of ', numprocs, ' processes'

  call mpi_finalize(ierr)
end program test1
```

Always need to: use mpi,
Start with mpi_init,
End with mpi_finalize.

Compiling and running MPI code (Fortran)

Try this test:

```
$ cd $CLASSHG/codes/mpi
$ mpif90 test1.f90
$ mpiexec -n 4 a.out
```

You should see output like:

```
Hello from Process number 1 of 4 processes
Hello from Process number 3 of 4 processes
Hello from Process number 0 of 4 processes
Hello from Process number 2 of 4 processes
```

Note: Number of processors is specified with `mpiexec`.

MPI Communicators

All communication takes place in **groups of processes**.

Communication takes place in some **context**.

A group and a context are combined in a **communicator**.

`MPI_COMM_WORLD` is a communicator provided by default that includes **all processors**.

MPI Communicators

All communication takes place in **groups of processes**.

Communication takes place in some **context**.

A group and a context are combined in a **communicator**.

`MPI_COMM_WORLD` is a communicator provided by default that includes **all processors**.

`MPI_COMM_SIZE(comm, numprocs, ierr)` returns the number of processors in communicator `comm`.

`MPI_COMM_RANK(comm, proc_num, ierr)` returns the rank of this processor in communicator `comm`.

The `mpi` module includes:

Subroutines such as `mpi_init`, `mpi_comm_size`,
`mpi_comm_rank`, ...

Global variables such as

`MPI_COMM_WORLD`: a communicator,

`MPI_INTEGER`: used to specify the type of data being sent

`MPI_SUM`: used to specify a type of reduction

Remember: Fortran is **case insensitive**:

`mpi_init` is the same as `MPI_INIT`.

MPI functions

There are 125 MPI functions.

Can write many program with these 8:

- `MPI_INIT(ierr)` Initialize
- `MPI_FINALIZE(ierr)` Finalize
- `MPI_COMM_SIZE(...)` Number of processors
- `MPI_COMM_RANK(...)` Rank of this processor
- `MPI_SEND(...)` Send a message
- `MPI_RECV(...)` Receive a message
- `MPI_BCAST(...)` Broadcast to other processors
- `MPI_REDUCE(...)` Reduction operation

Example: Approximate π

$$\begin{aligned}\text{Use } \pi &= 4 \int_0^1 \frac{1}{1+x^2} dx \\ &\approx 4\Delta x \sum_{i=1}^n \frac{1}{1+x_i^2} \quad (\text{midpoint rule})\end{aligned}$$

where $\Delta x = 1/n$ and $x_i = (i - 1/2)\Delta x$.

Fortran:

```
dx = 1.d0 / n
pisum = 0.d0
do i=1,n
    x = (i-0.5d0) * dx
    pisum = pisum + 1.d0 / (1.d0 + x**2)
enddo
pi = 4.d0 * dx * pisum
```

Approximate π using OpenMP parallel do

```
dx = 1.d0 / n
pisum = 0.d0
!$omp parallel do reduction(+: pisum) &
!$omp                private(x)
do i=1,n
    x = (i-0.5d0) * dx
    pisum = pisum + 1.d0 / (1.d0 + x**2)
enddo
pi = 4.d0 * dx * pisum
```

Approximate π using OpenMP — parallel chunks

```
points_per_thread = (n + nthreads - 1) / nthreads
pisum = 0.d0

!$omp parallel private(i,pisum_thread, &
!$omp                               istart,iend,thread_num)

!$ thread_num = omp_get_thread_num()
istart = thread_num * points_per_thread + 1
iend = min((thread_num+1) * points_per_thread, n)

pisum_thread = 0.d0
do i=istart,iend
    x = (i-0.5d0)*dx
    pisum_thread = pisum_thread + &
                    1.d0 / (1.d0 + x**2)
enddo

!$omp critical
    pisum = pisum + pisum_thread
!$omp end critical
!$omp end parallel

pi = 4.d0 * dx * pisum
```

Approximate π using MPI

```
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, proc_num, ierr)

if (proc_num == 0) n = 1000

! Broadcast to all processes:
call MPI_BCAST(n, 1, MPI_INTEGER, 0, &
               MPI_COMM_WORLD, ierr)

dx = 1.d0/n

points_per_proc = (n + numprocs - 1)/numprocs
istart = proc_num * points_per_proc + 1
iend = min((proc_num + 1)*points_per_proc, n)

pisum_proc = 0.d0
do i=istart,iend
    x = (i-0.5d0)*dx
    pisum_proc = pisum_proc + 1.d0 / (1.d0 + x**2)
enddo
call MPI_REDUCE(pisum_proc, pisum, 1, &
               MPI_DOUBLE_PRECISION, MPI_SUM, 0, &
               MPI_COMM_WORLD, ierr)

if (proc_num == 0) then
    pi = 4.d0 * dx * pisum
endif
```

MPI Broadcast

Broadcast a value from Process **root** to all other processes.

General form:

```
call MPI_BCAST(start, count, &
                datatype, root, &
                comm, ierr)
```

where:

- **start**: starting address (variable, array element)
- **count**: number of elements to broadcast
- **datatype**: type of each element
- **root**: process doing the broadcast
- **comm**: communicator

MPI Broadcast Examples

```
call MPI_BCAST(start, count, &
               datatype, root, &
               comm, ierr)
```

Broadcast 1 double precision value:

```
call MPI_BCAST(x, 1, &
               MPI_DOUBLE_PRECISION, 0, &
               MPI_COMM_WORLD, ierr)}
```

Broadcast j th column of a matrix (contiguous in memory):

```
real(kind=8), dimension(nrows, ncols) :: a
...
call MPI_BCAST(a(1, j), nrows, &
               MPI_DOUBLE_PRECISION, 0, &
               MPI_COMM_WORLD, ierr)
```


MPI Broadcast Examples

Broadcast *i*th row of a matrix (not contiguous!):

```
real(kind=8), dimension(nrows, ncols) :: a
real(kind=8), dimension(ncols) :: buffer
...
do j=1,ncols
    buffer(j) = a(i,j)
enddo

call MPI_BCAST(buffer, ncols, &
               MPI_DOUBLE_PRECISION, 0, &
               MPI_COMM_WORLD, ierr)
```

MPI Reduce

Collect values from all processes and reduce to a scalar.

General form:

```
call MPI_REDUCE(sendbuf, recvbuf, count, &
                datatype, op, root, &
                comm, ierr)
```

where:

- `sendbuf`: source address
- `recvbuf`: result address
- `count`: number of elements to send / receive
- `datatype`: type of each element
- `op`: reduction operation
- `root`: process receiving and reducing
- `comm`: communicator

A few possible reduction operations op :

- `MPI_SUM`: add together
- `MPI_PROD`: multiply together
- `MPI_MAX`: take maximum
- `MPI_MIN`: take minimum
- `MPI_LAND`: logical and
- `MPI_LOR`: logical or

MPI Reduce

Examples: Compute $\|x\|_\infty = \max_i |x_i|$ for a distributed vector:

```
xnorm_proc = 0.d0
do i=istart,iend
    xnorm_proc = max(xnorm_proc, abs(x(i)))
enddo

call MPI_REDUCE(xnorm_proc, xnorm, 1, &
               MPI_DOUBLE_PRECISION, MPI_MAX, 0, &
               MPI_COMM_WORLD, ierr)

if (proc_num == 0) print "norm of x = ", xnorm
```

Note: Do not need an `MPI_BARRIER` before or after the Reduce.

Processors do not exit from `MPI_REDUCE` until all have called the subroutine.

MPI Reduce

This code is wrong:

```
if (proc_num /= 0) then
  call MPI_REDUCE(xnorm_proc, xnorm, 1, &
                 MPI_DOUBLE_PRECISION, MPI_SUM, 0, &
                 MPI_COMM_WORLD, ierr)
  print "Done with Reduce: ", proc_num
endif
if (proc_num == 0) print "norm of x = ", xnorm
```

With more than one process, the Reduce statement is called by all but one.

None of them will ever print the “Done with Reduce” statement or continue to run. (Code hangs.)

MPI Reduce

This code is wrong:

```
if (proc_num /= 0) then
  call MPI_REDUCE(xnorm_proc, xnorm, 1, &
                 MPI_DOUBLE_PRECISION, MPI_SUM, 0, &
                 MPI_COMM_WORLD, ierr)
  print "Done with Reduce: ", proc_num
endif
if (proc_num == 0) print "norm of x = ", xnorm
```

With more than one process, the Reduce statement is called by all but one.

None of them will ever print the “Done with Reduce” statement or continue to run. (Code hangs.)

If only processors 1, 2, ... should participate in Reduce, need a different **communicator** than `MPI_COMM_WORLD`.

MPI Reduce for vectors

Compute: $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|$ for an $m \times n$ matrix A .

Suppose there are m processes and the i th process has a vector `arow(1:n)` containing the i th row of A .

Sum the row vectors (using Reduce) and then take maximum of resulting vector...

```
real(kind=8) :: arow(n), arow_abs(n), colsum(n)
...
arow_abs = abs(arow)

call MPI_REDUCE(arow_abs(1), colsum, n, &
               MPI_DOUBLE_PRECISION, MPI_SUM, 0, &
               MPI_COMM_WORLD, ierr)

if (proc_num == 0) then
    anorm = 0.d0
    do j=1,n
        anorm = max(anorm, colsum(j))
    enddo
    print "1-norm of A = ", anorm
endif
```