Introduction to parallel computing with MPI

Sergiy Bubin

Department of Physics
Nazarbayev University

Distributed Memory Environment

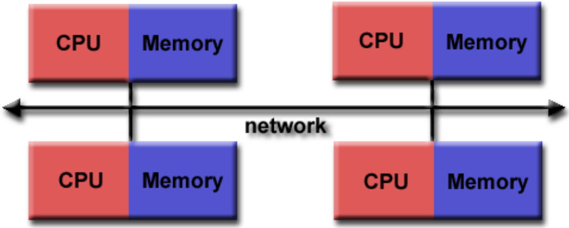
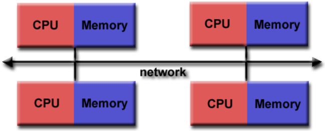


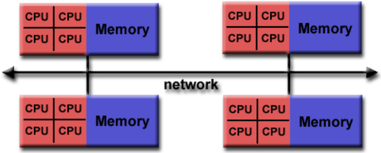
image credit: LLNL

Hybrid Memory Environment

Most modern clusters and supercomputers built using a hybrid approach: Many independent nodes. However, within a node (usually 2-4 CPUS, a few cores total) the memory is shared



Distributed Memory



Hybrid (Shared/Distributed)

What is MPI?

MPI (Message Passing Interface) is a standard of library routines that can be used to create parallel programs in Fortran and C/C++.

- De facto standard for communication among processes for programs running on distributed memory systems
- Dominant tool in today's high-performance computing
- Supported by many vendors of computer hardware
- Standardized and portable
- Includes routines for both point-to-point and collective communication
- Several standard revisions have been released: MPI-1, MPI-1.1, MPI-1.2, MPI-1.3, MPI-2, MPI-3

Use of MPI

- Originally, MPI was designed for distributed memory architectures
- Today, MPI runs on virtually any hardware platform: distributed memory, shared memory, hybrid
- The programming model remains a distributed memory model regardless of the actual physical environment
- While MPI is very functional and contains hundreds of various subroutines, most MPI programs can be written using just a few of them

Open MPI

Open MPI (not to be confused with OpenMP) is a free and open source (BSD license) implementation of MPI. Contributors to Open MPI include many hardware vendors and academic institutions. It is continuously developed, and maintained, and has a huge user base. Open MPI is included in most Linux distributions (usually not installed by default though).

How to compile and run an MPI program

Compile: normally one invokes a wrapper called mpif90 (mpicc, mpicxx, mpif77, etc.). This eliminates the need to include multiple arguments telling the compiler (e.g. gfortran, ifort, etc.) where the MPI libraries are located and how exactly the compilation should be done

```
mpif90 myprog.f90 -o myprog
```

Execute: use mpirun (mpiexec)

```
mpirun -np 4 ./myprog
```

← Runs 4 parallel MPI processes

MPI Programming Model

- An MPI program consists of autonomous processes
- The processes may run either the same code (SPMD style) or different codes (heterogenious). Most often the code is the same.
- Processes communicate with each other via calls to MPI subroutines as necessary
- Execution model allows each process to operate separately
- Processes are created at startup and continue throughout the entire execution
- Synchronization is implicit in each point-to-point or collective data movement collective data movement
- Memory is private to each process

MPI in Fortran

- Subroutine names are in uppercase; e.g., MPI_RECV

```
call MPI_XXXX(parameter, ...  , IErr)
```

- Subroutine return codes are represented by an additional integer argument. The return code for successful completion is MPI_SUCCESS (i.e. 0)
- Compile-time constants are in uppercase and are defined in the `mpif.h` file, which must be included in any program that makes MPI calls

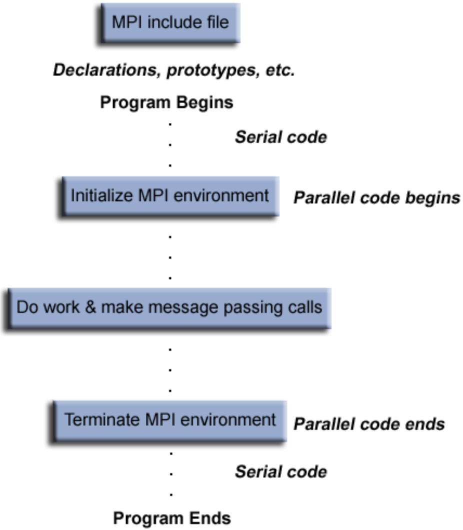
```
include 'mpif.h'
```

- An MPI datatype is defined for each Fortran datatype: MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_COMPLEX, MPI_LOGICAL MPI_CHARACTER, etc.

Basic Subroutines

- MPI_INIT
- MPI_COMM_SIZE
- MPI_COMM_RANK
- MPI_SEND
- MPI_RECV
- MPI_BCAST
- MPI_ALLREDUCE
- MPI_FINALIZE

MPI Program Structure



MPI include file

Declarations, prototypes, etc.

Program Begins

Serial code

Initialize MPI environment *Parallel code begins*

Do work & make message passing calls

Terminate MPI environment *Parallel code ends*

Serial code

Program Ends

Basic Program

A Hello, World! program with MPI would look as follows

```
program myprog
include ’mpif.h’
integer nprocs, rank, ierr
   !initialize MPI
   call MPI_INIT(ierr)
   !get number of tasks
   call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
   !get my rank
   call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
   write(*,*), ’Number of procs=’,nprocs,’  My rank=’,rank
   !finalize MPI
   call MPI_FINALIZE(ierr)
end program myprog
```

Basic Program

When we run the above program we will get:

```
mpirun -np 4 ./myprog
```

```
Number of procs=      4  My rank=      2
Number of procs=      4  My rank=      0
Number of procs=      4  My rank=      1
Number of procs=      4  My rank=      3
```

Basic Program: Integral of sin(x) from 0 to 1

In this program work is divided between processors

```
program integrate
include ’mpif.h’
integer nprocs, rank, ierr, i, npoints
real(8) s, ss, x, h
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
npoints=101;  h=0.01_8
do i=rank,npoints-1,nprocs
  x=i*h
  s=s+sin(x)
enddo
call MPI_ALLREDUCE(s,ss,1,MPI_DOUBLE_PRECISION,MPI_SUM, &
                   MPI_COMM_WORLD,ierr)
if (rank==0) write(*,*), ’Integral=’,ss*h
call MPI_FINALIZE(ierr)
end program integrate
```

Summary

- A computation consists of a (typically fixed) set of heavyweight processes, each with a unique identifier (integers 0,1,. . . ,P-1)
- Using this identifier and the total number of processes (which is known to each process via MPI_COMM_SIZE) the work can be distributed between processes unambigously
- Processes interact by exchanging typed messages, by engaging in collective communication operations, or by probing for pending messages
- Determinism is not guaranteed but can be achieved with careful programming when necessary

References

A huge number of tutorials and reference books exist on the internet. Feel free to google. Check out these, for example:

- https://computing.llnl.gov/tutorials/mpi/
- http://www.mcs.anl.gov/research/projects/mpi/usingmpi/