Using MPI on Atlas
Grace Foo
(SVU/Academic Computing, NUS)

This article is MPI 101, an introduction to using MPI on the atlas cluster.

Atlas is the new 16-node high performance Linux cluster at SVU. Each node has a dual Intel Xeon 3.06 GHz CPU and 2 GB of memory. Connecting the nodes are two switches, a fast Ethernet switch and a high-speed, low-latency Myrinet switch. Batch queues have been set up to facilitate job/load management on the cluster. Serial and parallel jobs are supported. More information about the cluster, such as software, libraries and tools, may be found at the following link: http://e-svu.nus.edu.sg/linuxdocs

MPI

MPI (Message Passing Interface) is a specification of a message passing library/interface for parallel computing. The effort was led by Argonne National Laboratory and other parties in business and academia. Since its release in 1994, MPI has become widely accepted and is the de facto standard for message passing. More information on MPI may be found at the website http://www-unix.mcs.anl.gov/mpi/.

There are currently two standards MPI 1 and MPI 2. MPI 1 is the original Message Passing Interface specification, supporting C and Fortran 77. MPI 2 contains enhancements and additions to MPI 1 and supports Fortran 90 and C++.

MPI is large with 129 functions, but not necessarily complex to use. Many useful programs may be written with just 6 basic functions. In MPI, a communicator is a group or collection of processes which can send messages to each other. Each process is associated with a unique rank (0 to N-1) in the set. There is a predefined communicator MPI_COMM_WORLD. Point-to-point or collective communication is possible.

The six basic MPI functions:
- MPI_INIT
- MPI_FINALIZE
- MPI_COMM_SIZE
- MPI_COMM_RANK
- MPI_SEND
- MPI_RECV

Hello world!

A minimal MPI program needs only MPI_INIT and MPI_FINALIZE. These functions handle the setup and tear down of the MPI execution environment. Also remember to include the MPI header file “mpif.h” which contains necessary definitions, macros and prototypes. Here is the infamous "hello world" program in fortran, parallelized using MPI:

```fortran
program hello
```

```fortran
```

```fortran
end program hello
```

```fortran
```
implicit none
include 'mpif.h'

integer ierr

call MPI_INIT( ierr )
write(*,*) 'hello world! '
call MPI_FINALIZE ( ierr )
end

Compiling the MPI code

MPI compilers from Portland Group (PG), GNU and Intel are available. Users may compile to run their codes using the Ethernet or the Myrinet switch. The later is appropriate for codes that have high communication to computation ratio. By default, the PG MPI compiler and Myrinet switch are used:

mpif77 –o hellomp hello.f

To compile for the Ethernet switch, specify the correct path:
/opt/mpich/ethernet/pgi/bin/mpif77 –o helloep hello.f

Using the GNU compilers:
/opt/mpich/myrinet/gnu/bin/mpif77 –o hellomg hello.f
/opt/mpich/ethernet/gnu/bin/mpif77 –o helloeg hello.f

Using the Intel compilers:
/opt/mpich/myrinet/intel/bin/mpif77 –o hellomi hello.f
/opt/mpich/ethernet/intel/bin/mpif77 –o helloei hello.f

Running the MPI code

Use the mpijob-gm command to launch the MPI job through the LSF batch queues to run using the myrinet switch, as we show in the following submission to the 4 processor queue:

bsub -q linux_4p –n 4 -o hellomp.out "mpijob-gm mpirun hellomp"

Use mpijob command for the same using the ethernet switch:

bsub -q linux_4p –n 4 -o helloep.out "mpijob /opt/mpich/ethernet/pgi/bin/mpiexec helloep"

Some more MPI

The next program uses two more functions, MPI_COMM_SIZE and MPI_COMM_RANK. MPI_COMM_SIZE reports the group size (number of processes) of the communicator while MPI_COMM_RANK reports the rank of the calling process. It shows each process reporting its own ID:

program hello2
implicit none
include 'mpif.h'

integer procid, numproc, ierr
C procid : ID of current process
C numproc : number of processes

call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, procid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numproc, ierr )
write(*,*) 'hello world! from process ', procid, ' of ', numproc

call MPI_FINALIZE(ierr)
end

With 4 processes (np 4), we have the following output:

hello world! from process 1 of 4
hello world! from process 0 of 4
hello world! from process 2 of 4
hello world! from process 3 of 4

Summary

We have looked at 2 simple MPI programs using 4 basic MPI functions. In a subsequent article, we will look at MPI functions for different modes of communication.

Meanwhile, please send your queries/problems on MPI to the SVU helpdesk at ccsvuhelp@nus.edu.sg.