1967-16

Advanced School in High Performance and GRID Computing

3 - 14 November 2008

General instructions on compiling MPI codes on clusters

BASHEER Ershaad Ahamed

Jawaharlal Nehru Centre for Advanced Scientific Research
Centre for Computational Materials Science, Jakkur P.O.
Bangalore 560064
Karnataka
INDIA

ICTP
Advanced School for High Performance
and GRID Computing

General Instructions for Compiling MPI Codes on Clusters

Ershaad Ahamed
**Jawaharlal Nehru Centre
for Advanced Scientific Research
Bangalore, India**

`ershaad@jncasr.ac.in`

MPI

- MPI is an API specification or standard
- There are several implementations of MPI
    - MPICH, OpenMPI, LAM ...
- These implementations consist of subroutines defined in shared libraries

MPI

- MPI programs use functions that are defined in these libraries
- MPI programs need to be linked to these libraries after compilation
- If the programs are dynamically linked, these libraries must be present when executing the programs

First Step

- Before compiling any package read the README/INSTALL files
- May contain important information and/or instructions

Makefile

- It is a file that specifies dependencies between source files and commands for building targets (executables) from them
- Specifies compiler flags and location of libraries
- Some packages need user to modify Makefile before building

Configure script

- This script generates a Makefile tailored for the system
- Tries to find library locations
- Behaviour can be modified by supplying options and setting environment variables
- Many source packages include a configure script

Configure script

- Typical sequence of commands for a package using a configure script
- After extracting from the archive (like tar.gz)
  - `$ ./configure`
  - `$ make`
  - `$ make install`
- To see all the options available to configure
  - `./configure -help`
  - Example `--prefix` allows you to install a package in your own home directory

Configure script

- Observe the output of `configure`
- configure searches some standard locatons
- Usually need header files mpi.h/mpif.h and libraries libmpi.so/libmpich.so/...
- If it doesn't find them use

  -
    ```
    ./configure ...
    CFLAGS="-I/directory/having/include"
    LDFLAGS="-L/directory/having/libraries"
    ```
  - Useful if you have installed the MPI libraries in your own home directory
  - `-I` and `-L` are compiler flags

Wrapper scripts

- MPI implementations differ in libraries to link
- Wrapper scripts to take care of calling compiler with right flags
  - `mpicc`
  - `mpiCC/mpicxx/mpic++`
  - `mpif77`
  - `mpif90`
- Use this to see the commandline invoked
  - `mpicc -show`

Wrapper scripts

- `$ mpicc -show`
  - `gcc  -L/opt/hpmpi/lib/linux_amd64  -I/opt/hpmpi/include -lhpmpio -lhpmpi -ldl`
- `$ mpiCC -show`
  - `g++  -L/opt/hpmpi/lib/linux_amd64  -I/opt/hpmpi/include -lhpmpio -lhpmpi -ldl -lmpiCC`
- `$ mpif77 -show`
  - `g77  -L/opt/hpmpi/lib/linux_amd64  -I/opt/hpmpi/include/64 -lhpmpio -lhpmpi -ldl`
- `$ mpif90 -show`
  - `f90  -L/opt/hpmpi/lib/linux_amd64  -I/opt/hpmpi/include/64 -lhpmpio -lhpmpi -ldl`

Wrapper scripts

- They are used in the same way as compilers
  - `mpicc -o hello_world_c hello_world.c`

Wrapper scripts

- Let configure use the wrapper scripts instead
  - `./configure CC=mpicc FC=mpif90`
- To make the wrappers use our compilers
  - Specific to MPI implementation. Read `mpicc/mpif90`... man page
  - For OpenMPI
    - `export OMPI_MPICC=icc` (to use the intel compiler)
    - Similarly set
    - `OMPI_MPIXX`, `OMPI_MPIF77` or `OMPI_MPIF90`

Editing makefiles

- For packages that do not use configure, we need to edit the Makefile
- 3 stages in build (preprocess, compile, link)
- Variables usually set in the makefile
  - `CC` (sets the C compiler)
  - `FC` (Fortran compiler)
  - `LD` (Linker)
  - `CPPFLAGS` (Preprocessor flags)
  - `CCFLAGS/FFLAGS` (C/Fortran compiler flags)
  - `LDFLAGS` (linker flags. Location of libraries)

example

```
FFLAGS = -pc64  -xW -O2 -unroll
LFLAGS = -L/sfs1/intel_072007/ict/3.0.1/cmkl/9.1/lib/em64t \
        -lmkl_lapack -lmkl
CFLAGS = -O2 -Wall -m64
CPP = /lib/cpp -P -C -traditional
CPPFLAGS = -D__Linux -D__PGI -DFFT_DEFAULT -DPOINTER8 -DLINUX_IFC
        -DPARALLEL -DMYRINET
CC = mpicc
FC = mpif77 -c
LD = mpif77
```

`-lmkl` option tells the linker to search for the library libmkl.so in the directories specified by the -L options

Common error

```
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `dswap_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `dgemm_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `dger_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `zscal_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `zlaev2_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `sswap_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `izamax_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `zgetrf_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `slaswp_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `dsyr_'
/home/ershaad/local/lib/libpardiso_GNU42_EM64T_INT64_P.so: undefined reference to `zsyr_'
```

•

•

•

Solution

- Error occurs because the linker cannot find a subroutine in any of the libraries included using `-l` flag
- We need to find which library file contains the subroutine
- `nm` lists all the symbols in an object file
  - In the directory containing the libraries
    - `nm -o * |grep subroutine_name`

Shared libraries

- MPI dynamic libraries should be available during program execution
- Library search paths are in environment variable LD_LIBRARY_PATH
- example

-
  ```
  $ echo $LD_LIBRARY_PATH
  /opt/hpmpi/lib/linux_amd64:/opt/hptc/lib:/opt/hptc/lsf/top/6.2
  /linux2.6-glibc2.3-x86_64-slurm/lib
  ```
- Or edit /etc/ld.so.conf (if you are root)
- Check if the dynamic linker is using the right libraries by using `ldd`

Summary

- Rely on wrapper scripts
- Options to configure
  - To use wrapper scripts
  - Locations of include files and libraries
- Edit makefile variables
- Make sure libraries are present during execution