Small Linux Cluster Workshop:

Installing MPI and Running Parallel Code

Markus Berndt, berndt@lanl.gov

T-7: Mathematical Modelling and Analysis Group
Los Alamos National Laboratory

July 8, 2001
• What is MPI?

• Installation of MPICH.

• Installation of LAM/MPI.

• Run some parallel code.
Why Message Passing?

- Memory hierarchy on a serial computer:
  - register
  - cache (L1, L2, ...)
  - ram

- All memory is directly accessible by the CPU

- Memory hierarchy on a cluster ... one additional level:
  - register
  - cache (L1, L2, ...)
  - ram
  - ram on a different node

- Ram on a different node is only accessible through communication (very slow in comparison to local memory)
• MPI stands for message-passing application programmer interface.

• Protocol and semantic specifications for how its features must behave in any implementation

• Provides abstractions for processes at two levels:
  – Processes are named according to the rank of the group in which the communication is being performed
  – Virtual topologies allow for graph or Cartesian naming of processes (this helps relating the application semantics to the message passing semantics in a convenient, efficient way)

• Provides three additional classes of services:
  – environmental inquiry,
  – basic timing information for application performance measurement,
  – profiling interface for external performance monitoring.
MPI – Available Implementations

- **MPICH** ([http://www-unix.mcs.anl.gov/mpi/mpich](http://www-unix.mcs.anl.gov/mpi/mpich) at ANL, MSU)
  - Systems that are supported:
    - Workstation clusters (with ch_p4 or ch_nexus)
    - Windows NT and Windows 2000
    - IBM SP (ch_mpi)
    - Intel i860, Delta, and Paragon (ch_nx)
    - Shared Memory systems (SMPs) (with ch_shmem)
    - CRAY T3D (t3d)
  - Many vendor implementations of the MPI are based on the MPICH implementation.

- **LAM-MPI** ([http://www.lam-mpi.org](http://www.lam-mpi.org) at UND)
  - Aimed at (heterogeneous) workstation clusters.
  - Not licensed under the GPL, but its license is ’open’.
MPICH
MPICH Installation

- Unpack MPICH ... tar xvfz mpich.tar.gz
- Apply all patches: patch -p0 < patch.all
- Configure MPICH ...
  - Change directory to the MPICH directory
  - Read the README file!
  - Read the documentation referenced in www/index.html!
  - Configure MPICH ... ./configure --with-device=ch_p4 ...
- Compile MPICH ... make
- Install MPICH ... make install
guero(20)% ./configure --help
Configuring with args --help
Configuring MPICH Version 1.2.1 of: 2000/09/05 15:06:05
Usage: ./configure [--with-arch=ARCH_TYPE] [-comm=COMM_TYPE]
    [--with-device=DEVICE]
    [--with-mpe] [--without-mpe]
    [--disable-f77] [--disable-f90] [--with-f90nag] [--with-f95nag]
    [--disable-f90modules]
    [--enable-c++] [--disable-c++]
    [--enable-mpedbg] [--disable-mpedbg]
    [--enable-devdebug] [--disable-devdebug]
    [--enable-debug] [--disable-debug]
    [--enable-long-long] [--disable-long-long]
    [--enable-long-double] [--disable-long-double]
    [-prefix=INSTALL_DIR]

    [-c++=[C++_COMPILER]] [noc++]
    [-opt=OPTFLAGS]
    [-cc=C_COMPILER] [-fc=FORTRAN_COMPILER]
    [-clinker=C_LINKER] [-flinker=FORTRAN_LINKER]
    [-c++linker=CC_LINKER]
    [-cflags=CFLAGS] [-fflags=FFLAGS] [-c++flags=CCFLAGS]
    [-optcc=C_OPTFLAGS] [-optf77=F77_OPTFLAGS]
    [-f90=F90_COMPILER] [-f90flags=F90_FLAGS]
    [-f90inc=INCLUDE_DIRECTORY_SPEC_FORMAT_FOR_F90]
    [-f90linker=F90_LINKER]
    [-f90libpath=LIBRARY_PATH_SPEC_FORMAT_FOR_F90]
    [-lib=LIBRARY] [-mpilibname=MPINAME]
(...)

7
Configuration options for MPICH

• A typical installation ...

./configure --prefix=/packages/mpich/mpich-1.2.1-absoft-7.0.1
--device=ch_p4 -rsh=ssh -cc=/packages/gcc/bin/gcc
-c++=/packages/gcc/bin/g++
-fc=/vendor/absoft/Pro_Fortran-7.0-1/bin/f77
-f90=/vendor/absoft/Pro_Fortran-7.0-1/bin/f95

– Install in /packages/mpich/mpich-1.2.1-absoft-7.0.1.
– Use device ch_p4.
– Use ssh to log in to the nodes.
– Use the GNU C and C++ compilers.
– Use the Absoft F77 and F95 compilers.

• If a production MPICH is to be built, use -opt=-O -disable-devdebug. This will produce smaller libraries and slightly faster code.
Testing the MPICH Installation

- Correctness: After the compilation of MPICH type

  make testing

  This validates the functionality of the MPI by running a number of tests.

- Performance: Change directory to examples/perftest and type make. Then you can run a number of performance tests (view the README file for details). For example:

  ./rungoptest -maxnp 2 -add -bcast -gnuplot -fname bcast.mpl

  The result can be viewed using

  gnuplot bcast.mpl
Some Important Notes

- Make sure that users can log in to any node using either `rsh` or `ssh` (depending on how you configured MPICH) without being prompted for a password.

- Users must have the directory that contains the MPICH installation in their `PATH`.

- Users should have the directory that contains the MPICH man pages in their `MANPATH`.

- If shared libraries were built, these libraries must be in the same directory on all nodes of the cluster. Users must have this directory in their `LD_LIBRARY_PATH`. 
Features of MPICH

- **ROMIO** is a high-performance, portable implementation of MPI-IO, the I/O chapter in MPI-2.

- **MPE** provides performance and correctness debugging, graphics, and some common utility routines.
  - A set of routines for creating logfiles for examination by various graphical visualization tools: upshot, nupshot, Jumpshot-2 or Jumpshot-3.
  - A shared-display parallel X graphics library.
  - Routines for sequentializing a section of code being executed in parallel.
  - Debugger setup routines.
Debugging Code with MPICH

- Use `write` or `printf` statements.

- The command line option `-gdb` will start the code on node 0 in the debugger `gdb`. (This does not work in conjunction with `-nolocal`)

- MPE library: Compile with
  - `-mpitrace` to trace every call to an MPI function.
  - `-mpianim` to view an animation of the communication (must link with `-lX11`)
  - `-mpilog` to create a log file that can be viewed with `upshot` after conversion to the alog format (use `clog2alog`).

- The totalview debugger can be used in conjunction with MPICH.
LAM/MPI
- Download LAM/MPI ... http://www.lam-mpi.org/download/ (the current version is 6.5.2)

- Unpack LAM/MPI ... tar xvfz lam-6.5.3.tar.gz

- Read the README and INSTALL files.

- Configure LAM/MPI:
  - ./configure --prefix=/packages/lam-6.5.2
  - make
  - make install
Conguration Options in LAM/MPI

```
mole(18)% ./configure --help
Usage: configure [options] [host]
Options: [defaults in brackets after descriptions]
(...)  
directory and file names:
  --prefix=PREFIX  install architecture-independent files in PREFIX
                  [/usr/local]
  --with-cc=CC     use C compiler CC
  --with-cflags=CFLAGS  use C compiler flags CFLAGS
  --enable-shared=[PKGS]  build shared libraries [default=no]
  --without-romio  disable ROMIO support in LAM/MPI
  --with-romio-flags=FLAGS  pass FLAGS to ROMIO’s configure script
  --without-mpi2cpp  build LAM without MPI 2 C++ bindings support
  --with-cxx=CXX   use C++ compiler CXX
  --with-cxxflags=CXXFLAGS  use C++ compiler flags CXXFLAGS
  --with-exceptions enable support for C++ exceptions
  --with-impi compile with IMPI support (6.4.x only)
  --with-exflags Specify flags necessary to enable exceptions
  --without-profiling disable the MPI profiling interface
  --with-trillium enable installation of Trillium header/man/binary files (not required for MPI)
  --with-ldflags=LDFLAGS use LD linker flags LDFLAGS
  --with-cxxldflags=CXXLDFLAGS use C++ LD linker flags CXXLDFLAGS
  --with-fc=FC    use Fortran compiler FC,
                  specify no to disable Fortran support
  --with-fflags=FFLAGS use Fortran compiler flags FFLAGS
  --with-rpi=RPI  build with RPI comm layer RPI
                  (where RPI=tcp|sysv|usysv|myri|via -- default is tcp)
(...)  
```
• Include the directory where you installed LAM/MPI in your path. Note: You must be able to ssh or rsh between the nodes.

• Edit the file LAMHOME/etc/lam-bhost.def to include one line for each node in your cluster:
  siam00 cpu=1
  siam01 cpu=1
  siam02 cpu=1
  siam03 cpu=1

• Log in to one of these nodes and start the LAM environment: lamboot

• Now we can use mpirun to run our code:
  siam00# mpirun -np 4 ./hello_world

• When you’re done, you must remove the LAM/MPI environment by typing wipe
Testing the LAM/MPI installation

- Download the file lamtests-6.5.2.tar.bz2, unpack it and cd into the directory lamtests-6.5.2.

- Read the README file!

- If you’ve installed LAM/MPI correctly and the binaries are in your path, no editing of the file Makefile.inc will be necessary.

- Use lamboot to start the LAM/MPI on at least one node.

- Type make to run all the tests.

- The hope is that at the end of the tests you will see the line Total errors: 0.

- Use wipe to finalize LAM/MPI.
Hello World!!

- Example code
  
  ```fortran
  program hello_world
  include 'mpif.h'
  integer nproc, myproc, ierror

  call MPI_Init(ierr)
  call MPI_Comm_size(MPI_COMM_WORLD, nproc, ierr)
  call MPI_Comm_rank(MPI_COMM_WORLD, myproc, ierr)
  call MPI_Finalize(ierr)

  write(*,*) 'I am node ',myproc,' of ',nproc
  end
  ```

  - Compile this using the `mpif77` command.

  - Create a file that contains the names of the nodes, let's call it `mynodes`
    
    `siam00`
    `siam01`
    `siam02`
    `siam03`
To run this program, type on guero

guero[12]: mpirun -machinefile mynodes -nolocal -np 4 ./hello_world

I am node 2 of 4
I am node 1 of 4
I am node 3 of 4
I am node 0 of 4
Debugging Code with LAM/MPI

- If you must, use `write` or `printf` statements.

- Use a script to start the code within a debugger, let’s call it `run_gdb.csh`

```csh
#!/bin/csh -f

echo "Running xterm on 'hostname'"
xterm -e gdb $*
exit 0
```

Note: This script must be executable.

We can now run in parallel within gdb, for example

```bash
mpirun -np 2 run_xterm hello_world
```
Some Applications

  The standard yardstick that is used to measure the numerical performance of a parallel computer.

  An automatically tuned version of the BLAS and some of the LAPACK routines. Without using these, Linpack will be very slow!

- **NAS benchmarks**: [http://www.nas.nasa.gov/Software/NPB/](http://www.nas.nasa.gov/Software/NPB/)
  This benchmark gives a more realistic assessment of the computational performance that can be expected from the cluster in applications.
Comparison of 100Mb/s, bonded 100Mb/s, and 1Gb/s:

<table>
<thead>
<tr>
<th>Size  \ Configuration</th>
<th>100Mb/s</th>
<th>bonded 100Mb/s</th>
<th>1Gb/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000 × 5000</td>
<td>2.032 GFlop/s</td>
<td>2.269 GFlop/s</td>
<td>2.493 GFlop/s</td>
</tr>
</tbody>
</table>

'Peak' Linpack Performance (1Gb/s Configuration):

<table>
<thead>
<tr>
<th># procs, problem size \</th>
<th>GFlop/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 10000 × 10000</td>
<td>.856</td>
</tr>
<tr>
<td>4, 20000 × 20000</td>
<td>3.036</td>
</tr>
</tbody>
</table>
The NAS benchmark suite form NASA ...

<table>
<thead>
<tr>
<th></th>
<th>BT</th>
<th>CG</th>
<th>EP</th>
<th>IS</th>
<th>LU</th>
<th>MG</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>280</td>
<td>41</td>
<td>8.5</td>
<td>1.3</td>
<td>431</td>
<td>115</td>
<td>114</td>
</tr>
<tr>
<td>B</td>
<td>333</td>
<td>52</td>
<td>8.5</td>
<td>1.3</td>
<td>463</td>
<td>125</td>
<td>152</td>
</tr>
<tr>
<td>C</td>
<td>59</td>
<td>59</td>
<td>8.6</td>
<td>1.3</td>
<td>518</td>
<td>212</td>
<td>193</td>
</tr>
</tbody>
</table>

Numbers are in MFlop/s; A, B, C are different problem sizes.