MPICH2 Logging
Version 0.1
DRAFT of May 3, 2010
Mathematics and Computer Science Division
Argonne National Laboratory

David Ashton

May 3, 2010

1 Introduction

This manual assumes that MPICH2 has already been installed. For instructions on how to install MPICH2, see the MPICH2 Installer's Guide, or the README in the top-level MPICH2 directory. This manual will explain how the internal logging macros are generated and how the user can generate log files viewable in Jumpshot. Use of Jumpshot is described in the mpe documentation.

2 Configuring mpich2 to create log files

When users run configure they can specify logging options. There are three configure options to control logging.

`--enable-timing=<timing_type>`
: Add this option to enable timing. The two options for timing_type are `log` and `log_detailed`. The `log` option will log only the MPI functions just like the MPE logging interface does. The `log_detailed` will log every function in mpich2. This option gives fine grained logging information and also creates large log files. It must be used in conjunction with a timer-type that can log very short intervals on the order of 100's of nanoseconds.

`--with-logging=<logger>`
: Specify the logging library to use. Currently the only logger option is `rlog`.

`--enable-timer-type=<timer_type>`
: Specify the timer type. The options are

  - `gethrtime` - Solaris timer (Solaris systems only)
  - `clock_gettime` - Posix timer (where available)
  - `gettimeofday` - Most Unix systems
  - `linux86_cycle` - Linux x86 cycle counter*
  - `linuxalpha_cycle` - Like linux86_cycle, but for Linux Alpha*
  - `gcc_ia64_cycle` - IA64 cycle counter*

* Note that CPU cycle counters count cycles, not elapsed time. Because processor frequencies are variable, especially with modern power-aware hardware, these are not always reliable for timing and so should only be used if you're sure you know what you're doing.

Here is an example:

```
mpich2/configure
    --enable-timing=log
    --with-logging=rlog
    --enable-timer-type=gettimeofday
    ...
```

3 Generating log files

Run your mpi application to create intermediate `.irlog` files.

```
mpicc myapp.c -o myapp
mpiexec -n 3 myapp
```

There will be .irlog files created for each process:

```
log0.irlog
log1.irlog
log2.irlog
```

4 RLOG tools

For performance reasons each process produces a local intermediate log file that needs to be merged into a single rlog file. Use the rlog tools to merge the `.irlog` files into an `.rlog` file. The rlog tools are found in `mpich2_build/src/util/logging/rlog`. Currently they are not copied to the install directory.

`irlog2rlog`
: This tool combines the intermediate `.irlog` files into a single `.rlog`

file. The usage is: "`irlog2rlog outname.rlog input0.irlog input1.irlog ...`" A shortcut is provided: "`irlog2rlog outname.rlog <num_files>`". Execute `irlog2rlog` without any parameters to see the usage options.

`printrlog`
: This tool prints the contents of an `.rlog` file.

`printirlog`
: This tool prints the contents of an `.irlog` file.

Continuing the example from the previous section:

```
irlog2rlog myapp.rlog 3
```

will convert `log0.irlog`, `log1.irlog` and `log2.irlog` to `myapp.rlog`.

5 Viewing log files

This section describes how to view a log file

`.rlog` files can be printed from a command shell using the `printrlog` tool but the more interesting way to view the log files is from Jumpshot. Jumpshot displays slog2 files and has a built in converter to convert `.rlog` files to `.slog2` files. Start Jumpshot and open your `.rlog` file. Jumpshot will ask you if you want to convert the file and you say yes.

6 Logging state code generation

This section can be skipped by users. It describes the internal scripts used to develop the logging macros.

This is how the `maint/genstates` script works:

1. `maint/updatefiles` creates `genstates` from `genstates.in` replacing `@PERL@` with the appropriate path to perl and then runs `genstates`.
2. `genstates` finds all `.i`, `.h` and `.c` files in the mpich2 directory tree, searches for `_STATE_DECL` in each file and builds a list of all the MPID_STATEs.

It validates that the states start in a _STATE_DECL statement, followed by a FUNC_ENTER statement, and then at least one FUNC_EXIT statement. Errors are printed out if the code does not follow this format except for macros. State declarations in macros are assumed to be correct.

3. genstates finds all the describe_states.txt files anywhere in the mpich2 tree. describe_states.txt files are optional and are used to set the output name of the state and its associated color.

4. The describe_states.txt file format is this:

   ```
   MPID_STATE_XXX <user string for the state> <optional rgb color>
   ```

   Here is an example line:

   ```
   MPID_STATE_MPI_SEND MPI_Send 0 0 255
   ```

   If you don't specify a state in a describe_states.txt file then the state user name will be automatically created by stripping off the MPID_STATE_ prefix and the color will be assigned a random value.

5. genstates ouputs mpich2/src/include/mpiallstates.h with this enum in it:

   ```
   enum MPID_TIMER_STATE
   {
       MPID_STATE_XXX,
       ...
   };
   ```

6. genstates outputs mpich2/src/util/logging/describe_states.c with the MPIR_Describe_timer_states() function in it. Currently, only the rlog version of MPIR_Describe_timer_states() is generated.