The IBM High Performance Computing Toolkit

Trace Libraries Documentation

Please send corrections to: David Klepacki
(klepacki@us.ibm.com)

Table of Contents

Overview

This is the documentation for the IBM ACTC Trace libraries. This libraries collect profiling and tracing data for MPI and TurboSHMEM programs. Both IBM production MPI and TurboMPI are supported. The trace libraries consists of the following seven files:

File Name	*MPI*	*SHMEM*	*TRACE*	*PROFILE*
libmpitrace.a	MPI 1.1 / MPI 2		X	X
libmpiprof.a	MPI 1.1 / MPI 2			X
libturbo1trace.a	MPI 1.1		X	X
libturbo1prof.a	MPI 1.1			X
libturbo2trace.a	MPI 2		X	X
libturbo2prof.a	MPI 2			X
libsmaprof.a		TurboSHMEM		X

System Setup & Compiling

The trace libraries using the debugging information stored within the binary to find the information that points to the source code. To use the libraries, the code must be compiled with "-g" as an additional compiler option. To make sure that this works, you have to make sure that the setup for you compiler is correct.

There are basically two files /etc/vac.cfg and /etc/xlf.cfg which defines a default configuration for your compiler and (more important for the following information) for the linker. This files usually can be changed by the System Administrator only.
These files should contain lines like (in a section called DEFLT:)

```
options = -D_AIX,-D_AIX32,-D_AIX41,-D_AIX43,-D_AIX50, ...
options32 = -bpT:0x10000000,-bpD:0x20000000
options64 = -bpT:0x100000000,-bpD:0x110000000
```

The lines starting with “options32” and “options64” are important. If this lines are not present and your program prints messages like

“Trace library waring: unable to map from the instruction address
to ANY source file. Did you compile with -g ?”

and you have compiled with -g add the flags within the “options64” to you command line when linking with -q64.

Compiling and Linking

To link your application with one of the the libraries add two options to your command line. At first, the option “-L/path/to/libraries/”, where /path/to/libraries is the path where the libraries are located. In most cases $IHPCT_BASE/lib should be the choice if you have include the environment file created during the installation of this trace libraries. Please see the README.installation (section 4) for details.

C example for LINKING with the MPI trace library:

```
mpcc_r -o myprog mpiprg.c -L/path/to/trace/libraries -lmpitrace
```

Fortran example for LINKING with the SHMEM profiling library:

```
mpxlf_r -o myprog mpiprog.f -L/path/to/trace/libraries \
      -L/path/to/shmem/library -lsmatrace -lsma
```

C example for LINKING with TurboMPI1 tracing:

```
mpcc_r -o myprog mpiprg.c -L/path/to/trace/libraries \
     -L/path/to/turbo/library -lturbo1trace -lturbo1 -lxlf90_r
```

Please note that all the libturbo* libraries will only work when linked with the appropriate TurboMPI library. The libturbo1trace.a and the libturbo1prof.a will only work when used together with the flag -lturbo1. The mixing of the TurboMPI libraries with the wrong trace libraries will result in undefined symbols.

Please note the usage of the flag "-lxlf90_r" for the C case.

Overhead

The overhead for the collection of a single MPI event (where 'event' is defined as a call to a MPI function) is about 1.7 µs. There is not a significant difference beetwen the overhead for the profile only case and the profile&trace case.

Implementation Status

The Implementation of the MPI2 wrappers consists only of some communication functions. These are all the MPI_Win* functions and functions for one sided communication (put, get, accumulate).

The Implementation of the SHMEM wrappers are complete. Make sure that you always link with the latest libsma.a. If you still use libsmac.a or libsmaf.a the linking will fail, they do not contain the needed profiling interface. Contact David Klepacki (klepacki@us.ibm.com) for new libraries, if needed.

Note: The Fortran trace wrappers are in lower case (mpi_send). If the Fortran source is compiled with the -qmixed option, then the names of MPI routines are not mapped to lower case, and you have to make sure that the wrappers have names that match the case used in the application.

Output and Environment Variables

To use the trace wrappers, link with the appropriate library. All libraries are able to generate to generate both viz files (to be viewed with PeekPerf) and plain text output. Which output a programm run creates depends on the following environment variables:

- TRACE_TEXTONLY :

  If set to "1", plain text output is generated. In not "1", a viz file is generated. The latter case is the default case. The text output shows which functions are called how many times and which message sizes had been used. The output is always produced per task.

- TRACE_PERFILE : (only makes sense when TRACE_TEXTONLY=1)

  If set to "1", the output is shown for each source file. If not set, the output is a summary of all source files.

- TRACE_PERSIZE : (only makes sense when TRACE_TEXTONLY=1)

  If set to "1", the statistic for a function is shown for every message size. Message size really means a 'message size range'. How often a function is called with a special message size is recorded for ranges of 0..4 Byte, 5..16 Bytes, 17..64 Bytes and so one. If you MPI_Send is called twice with 3 Bytes and one time with 2 Bytes you will see three calls between 0..4 Bytes and a sum of 8 transfered bytes. If not set, you will see the summary for all messages sizes.

Note that the mpi*trace libraries will always generate a trace file. So if you only want text output please use the mpi*prof libraries. The generated trace file can be viewed with the PeekView trace viewer.

By default each MPI task will create files in the working directory with names: mpi_profile_0.viz, mpi_profile_1.viz and so on. The mpi*trace libraries will also generate a file called 'single_trace'. The filenames for shmem are shm_profile_0.viz, shm_profile_1.viz, ... There is also a TRACE_DIR environment variable that can be used to specify the directory for mpi trace files - the default is to write the files in the working directory for each MPI task. If you cancel or kill a running process you will see the temporary trace files with the name trace_file_<#task> with in the directory. Just remove them, they are useless if the trace library never reached the MPI_Finalize() call.

More Environment Variables

For all of the versions you can choose to bind MPI tasks to processors if you set an environment variable BIND_TASKS to yes. When BIND_TASKS is set, the wrapper for MPI_Init will attempt to bind the MPI tasks to processors in a way that spreads the tasks out over the avialble CPU's as much as possible. In addition, you can optionally set the variables BIND_BASE and BIND_INC to control the placement of tasks on CPU's.

In some applications there are message-passing wrappers (GAMESS, sweep3d, ...), and one would like the profile to indicate the name of the routine that called the wrapper, not the name of the routine that called the MPI function. In this case, one can set an environment variable TRACELEVEL=1 (or higher, 0 is the 'normal' case), and then run the application (which must be compiled with either -g or -qtbtable=full). It may also be useful to try higher levels such as TRACELEVEL=3, which associates the message-passing time with the great-grandparent in the call chain. If you set TRACELEVEL to a too high value, you program will probably generate a segmentation fault. So try to start with TRACELEVEL=0 and increase on demand.

Final Note

The current version is not thread-safe, so it should be used in single-threaded applications, or when only one thread makes MPI calls. The wrappers could be made thread-safe by adding mutex locks around updates of static data - which would add some additional overhead.