tool for analysing and checking MPI programs

May 19, 2009

Contents

1 Introduction

1.1 What is Marmot?

- Marmot is a library written in C++, which has to be linked to your application in addition to the existing MPI library.
- It will check if your application conforms to the MPI standard and will issue warnings if there are errors or non-portable constructs.
- You need not modify your source code, you only need one additional process working as Marmot's debug server.
- Marmot's output is a human-readable text file, an HTML file or uses a format that allows display in other tools, e.g. Cube.
- The tool can be configured via environment variables.

1.2 Design of Marmot

Marmot makes use of the so-called "profiling interface" defined in the MPI standard, i.e. it intercepts the MPI calls from the application for examination before they are passed to the underlying MPI implementation. Marmot maps MPI resources such as communicators, datatypes etc. to its own resources to keep track of proper construction and usage.

1.3 Current Status

- Full support of MPI-1.2
- C/Fortran language binding
- Support for hybrid applications using MPI and OpenMP
- Support for CMake [CMAKE] building process (not fully completed yet)
- Support for the Cube [SCALASCA] Visualizer

1.4 Future Plans

- Extended functionality, e.g. one sided communication and parallel file I/O as defined in MPI-2
- Furhther integration into IDE's and high performance tools
- Enhanced deadlock detection

2 Installation

2.1 Software requirements

2.1.1 MPI implementation

1. Since the application that is to be verified is written using MPI, the MPI library is needed to run the application. Marmot verifies the calls made by the program with the use of the so called profiling interface (PMPI). This profiling interface is part of the MPI standard. Any MPI implementation that conforms to the MPI standard needs to provide this interface. Therefore, this requirement should not limit the selection of possible MPI libraries.

2. The MPI implementation itself may require some other software, for example globus libraries when using mpich-g2.

3. Some of Marmot's source files include mpi.h, therefore MPI (i.e. at least mpi.h) is needed for the compilation of the Marmot libraries, which can then be linked to an application.

2.1.2 OpenMP

Marmot supports hybrid programs using MPI and OpenMP. In order to do so, Marmot has to use an internal synchronisation, thus enabling full MPI_THREAD_MULTIPLE support. This synchronisation uses OpenMP directives which also allows Marmot to gather additional information about the threads used. See Section 2.4 on how to configure Marmot with OpenMP support. Your compiler must of course support OpenMP as well.

2.1.3 C++ compiler

1. Marmot is implemented in C++. The compiler should implement the ISO/IEC 14882 language specification of C++. We have succeeded in using gcc 2.96 or later, earlier versions might not work properly. Intel Compilers are an alternative, they are available for no cost for non-commercial use on linux platforms. For example, on our local environment, Intel compilers version 10.0 have been used successfully.

2. To link Marmot to a C or Fortran application with the C/Fortran linker instead of the C++ linker, some C++ libraries will have to be linked, too (for example libstdc++, using gcc or g77).

2.1.4 Fortran Compiler

To support the Fortran binding of the MPI standard a Fortran compiler is required. The same Fortran compiler should be used to compile the application.

2.1.5 C compiler

To support C applications, a C compiler is required (any C compiler should do).

2.1.6 Other tools that users need

make (tested with GNU make 3.79.1 and later versions) for compilation. Also, some of the Marmot helper tools, e.g. compiler wrappers, need the *bash* shell and an *awk* interpreter.

2.1.7 Other tools that users do not necessarily need[1]

1. Doxygen [DOXYGEN] (tested with version 1.2.14 and later versions) is used to automatically generate documentation. This documentation is supposed to provide sufficient information for users even if they do not have Doxygen.
2. autoconf-2.63 [AUTOCONF] or higher is required to generate a configure script from the configure.ac (tested with GNU Autoconf 2.63).
3. aclocal/automake [AUTOMAKE] (tested with GNU automake 1.10).
4. perl (tested with v5.6.1) is required by automake.
5. Scalasca [SCALASCA], if you want to use Cube logging (see Section 3.3)
6. CMake [CMAKE] if you want to build Marmot with CMake

2.2 Hardware requirements

Marmot does not require any specific hardware (any UNIX or Windows environment should do). It has been tested on the following platforms:

- LINUX IA32/IA64 clusters
- SGI Altix 4700
- Cray T3e
- Regatta (IBM-cluster)
- NEC SX6, SX8
- Windows Server 2003 & 2008 cluster

[1] Marmot's distribution comes with all the required files that users just have to run "configure" and "make", users do not need automake, autoconf, libtool etc. However, if you plan to create all these files yourself with autobuild tools, have a look at the autogen.sh script.

2.3 Basics

2.3.1 Installation

Marmot can be built using autotools. Basically, the following commands are sufficient:

```
$cd MARMOT
$./configure <OPTIONS>
$make
$make install
```

However, for more details, read also the installation examples in Appendix A on page 25 . Sometimes it might be necessary to provide ./configure with options, e.g. specifications of paths or compilers.

An alternate way to configure Marmot is the usage of CMake. CMake provides a command line tool and also graphical user interfaces for an easy configuration process. Basically you just have to create a folder for example inside the Marmot trunk, e.g. "MyMarmot", change into this folder and issue the command `$ccmake ..` On windows you could use the cmakesetup.exe application.

Figure 1: Ncurses display of CMake

You can then adapt the configuration to your needs. However, not every option provided by "configure" is covered by the CMake building process yet, e.g. building of shared libraries. Some of the important CMake configuration options for Marmot are:

`CMAKE_INSTALL_PREFIX` determines the installation path

`USED_MPI_PACKAGE` determines which MPI find module is used. You may choose: `MPICH`, `OPENMPI` or `MPI`.

`MARMOT_ADDIN` when switched to ON the AddIn for VisualStudio is compiled (windows specific)

`MARMOT_ENABLE_FORTRAN` specifies whether Marmot is built with Fortran support

`USE_CUBE` specifies whether Marmot is built with Cube support

In CMake frontends such as ccmake or cmakesetup you may switch to an advanced mode where you can alter all the cmake-variables relevant to the configuration process. An important thing to note here is that although the compiler variables are listed the compiler used (for example `CMAKE_C_COMPILER`) cannot be changed here. In case you would like to specify a different compiler you have to set the `CC` and `CXX` environment variables before the first call of ccmake.

2.3.2 Installation with OpenMP support

A Marmot installation with OpenMP support can only be used for programs linked with OpenMP. So on most systems one will use an extra installation of Marmot for an OpenMP version of Marmot. Many MPI implementations use a different MPI library for hybrid programs using MPI_THREAD_MULTIPLE. For Marmot you should specify this library, i.e. by using `--with-mpi-libs=-lmpi_mt` (if your multithreaded MPI library is named "libmpi_mt").

2.4 Configure Options

Marmot provides the following configure options:

OPTION : most important options and environment variables
OPTION : options needed for CUBE support
OPTION : options needed for OpenMP support

2.4.1 Installation directories

`--prefix=PREFIX`	install architecture-independent files in PREFIX [/usr/local]
`--exec-prefix=EPREFIX`	install architecture-dependent files in EPREFIX [PREFIX]
`--bindir=DIR`	user executables [EPREFIX/bin]
`--sbindir=DIR`	system admin executables [EPREFIX/sbin]
`--libexecdir=DIR`	program executables [EPREFIX/libexec]
`--sysconfdir=DIR`	read-only single-machine data [PREFIX/etc]
`--sharedstatedir=DIR`	modifiable architecture-independent data [PREFIX/com]
`--localstatedir=DIR`	modifiable single-machine data [PREFIX/var]
`--libdir=DIR`	object code libraries [EPREFIX/lib]
`--includedir=DIR`	C header files [PREFIX/include]
`--oldincludedir=DIR`	C header files for non-gcc [/usr/include]
`--datarootdir=DIR`	read-only arch.-independent data root [PREFIX/share]
`--datadir=DIR`	read-only architecture-independent data [DATAROOTDIR]
`--infodir=DIR`	info documentation [DATAROOTDIR/info]
`--localedir=DIR`	locale-dependent data [DATAROOTDIR/locale]
`--mandir=DIR`	man documentation [DATAROOTDIR/man]
`--docdir=DIR`	documentation root [DATAROOTDIR/doc/PACKAGE]
`--htmldir=DIR`	html documentation [DOCDIR]
`--dvidir=DIR`	dvi documentation [DOCDIR]
`--pdfdir=DIR`	pdf documentation [DOCDIR]
`--psdir=DIR`	ps documentation [DOCDIR]

2.4.2 Program names

`--program-prefix=PREFIX`	prepend PREFIX to installed program names
`--program-suffix=SUFFIX`	append SUFFIX to installed program names
`--program-transform-name=PROGRAM`	run sed PROGRAM on installed program names

2.4.3 System types

`--build=BUILD`	configure for building on BUILD [guessed]
`--host=HOST`	cross-compile to build programs to run on HOST [BUILD]

2.4.4 Optional Features

`--disable-FEATURE`	do not include FEATURE (same as –enable-FEATURE=no)
`--enable-FEATURE[=ARG]`	include FEATURE [ARG=yes]
`--disable-doc`	disable building documentation, default is to build it
`--enable-tests`	enable building test executables, default is not to build them
`--enable-signal-based-mpi`	use signal-based MPI library on IBM, default=no
`--enable-globus`	use globus, default=no
`--enable-myrinet`	use Myrinet libraries, default=no
`--enable-mpichp4`	use MPICH with p4 device, default=no
`--enable-cube`	use cube when you want to be able to use the CUBE display for logging, default=no
`--disable-dependency-tracking`	speeds up one-time build
`--enable-dependency-tracking`	do not reject slow dependency extractors
`--enable-openmp`	enable OpenMP usage if you want to use Marmot for applications using OpenMP threading, default=no Note: The `--enable-openmp` flag switches the OpenMP support on. The other flags are used to specify the additional flags, paths and libraries needed for OpenMP on your system. Usually you will only need `--with-openmp-flag=-openmp` (or the appropriate flag for your compiler).

2.4.5 Optional Packages

`--with-PACKAGE[=ARG]`	use PACKAGE [ARG=yes]
`--without-PACKAGE`	do not use PACKAGE (same as –with-PACKAGE=no)
`--with-cxx-lib-dir=CXX_LIBDIR`	give the path for CXX-libraries, default: /usr/lib
`--with-cxx-libs=CXX_LIBS`	give the CXX-libraries, default: -lstdc++
`--with-cldflags=CLDFLAGS`	give the linker flags to use in TEST_C directory, default: LDFLAGS set by user (empty if none).

--with-fldflags=FLDFLAGS	give the linker flags to use in TEST_F directory, default: LDFLAGS set by user (empty if none).
--with-mpi-dir=MPIDIR	give the path for MPI, default: /usr/local/mpich
--with-mpi-inc-dir=MPI_INCDIR	give the path for MPI-include-files, default: MPIDIR/include
--with-mpif-inc-dir=MPIF_INCDIR	give the path for MPIF-include-files, default: MPIDIR/include
--with-mpi-lib-dir=MPI_LIBDIR	give the path for MPI-libraries, default: MPIDIR/lib
--with-mpi-libs=MPI_LIBS	give the MPI-libraries to link to application (including calls for profiling interface!), default: libraries found automatically by configure
--with-mpi-bin-dir=MPI_BINDIR	give the path for MPI-binaries, default: MPIDIR/bin
--with-mpicxx=MPICXX	give the path for the MPI cxx compiler, default: compiler found automatically by configure
--with-mpicc=MPICC	give the path for the MPI c compiler, default: compiler found automatically by configure
--with-mpif77=MPIF77	give the path for the MPI f77 compiler, default: compiler found automatically by configure
--with-mpif90=MPIF90	give the path for the MPI f90 compiler, default: compiler found automatically by configure
--with-mpif95=MPIF95	give the path for the MPI f95 compiler, default: compiler found automatically by configure
--with-cube-inc-dir=CUBE_INCDIR	give the path for the CUBE include directory (only needed when –enable-cube was set), default: /usr/local/include
--with-cube-lib-dir=CUBE_LIBDIR	give the path for the CUBE library (only needed when –enable-cube was set), default: /usr/local/lib
--with-cube-lib=CUBE_LIB	give the name of the cube library (only needed when –enable-cube was set), default: -lcube
--with-xml2-lib-dir=XML2_LIBDIR	give the path for the xml2 library (only needed when –enable-cube was set), default: /usr/lib

`--with-xml2-lib=XML2_LIB`	give the name of the xml2 library (only needed when –enable-cube was set), default: -lxml2
`--with-globus-dir=GLOBUSDIR`	give the path for globus directory, default: /opt/globus
`--with-globus-lib-dir=GLOBUS_LIBDIR`	give the path for globus-libraries, default: `GLOBUSDIR`/lib
`--with-myrinet-dir=MYRINETDIR`	give the path for Myrinet directory, default: /opt/Myricom
`--with-myrinet-lib-dir=MYRINET_LIBDIR`	give the path for Myrinet-libraries, default: `MYRINETDIR`/lib
`--with-rpm-dir=DIR`	give RPM directory, default: pwd
`--with-marmot-bin-prefix=PREFIX`	give prefix for installed binaries, default: package name
`--with-marmot-lib-prefix=PREFIX`	give prefix for installed libraries, default: marmot
`--with-openmp-inc-dir=OPENMP_INCDIR`	give the path for the OpenMP include directory (only needed when `--enable-openmp` was set), default: ""
`--with-openmp-lib-dir=OPENMP_LIBDIR`	give the path for the OpenMP libraries (only needed when `--enable-openmp` was set and there is the need of an extra OpenMP library), default: ""
`--with-openmp-libs=OPENMP_LIBS`	give the name of the openmp libraries if necessary (only needed when `--enable-openmp` was set), default: ""
`--with-openmp-flag=OPENMP_FLAG`	give the name of the compiler flag used for enabling openmp (only needed when `--enable-openmp` was set), default: "-openmp"

2.4.6 Influential environment variables:

CXX	C++ compiler command
CXXFLAGS	C++ compiler flags
LDFLAGS	linker flags, e.g. -L¡lib dir¿ if you have libraries in a nonstandard directory ¡lib dir¿
CPPFLAGS	C/C++/Objective C preprocessor flags, e.g. -I¡include dir¿ if you have headers in a nonstandard directory ¡include dir¿
AR	Archiver to create libraries, useful for cross compilation
PRELINK	use -prelink on SX machines
RANLIB	ranlib to bless libraries
CC	C compiler command
CFLAGS	C compiler flags
CPP	C preprocessor
F77	Fortran 77 compiler command
FFLAGS	Fortran 77 compiler flags

2.5 Configure

Run `./configure` to create the Makefiles etc. automatically from the corresponding templates named *.in. Note that the default values may not be correct and that you may have to specify options for ./configure, for example to specify the paths of MPI and C/C++/Fortran compilers. Consult Appendix A to get an idea how to configure Marmot for different platforms.

2.6 DDT Plugin

A plugin for Allinea's parallel debugger DDT is under development. However, a first version is already available. With this plugin you will be able to run a debugging session with DDT and at the same time switch on the Marmot plugin to perform Marmot's checks.

2.6.1 Prerequisites

To use Marmot's DDT plugin you need

- DDT from Allinea [ALLINEA], along with a valid Licence, version 2.3.1 or above,
- Open MPI with shared libraries (support of other MPIs will follow),
- Marmot with shared libraries

2.6.2 Installing DDT

Installation instructions for DDT can be found in the DDT manual. After installing DDT you will find the following folder and files in the DDT folder:

```
$ls
```

```
bin doc examples help icons lib Licence plugins script templates wizard
```

To use plugins you need to create a plugin folder. Chnage into the DDT folder and issue

```
$mkdir plugins
```

2.6.3 Installing the Marmot Plugin

Installing the Marmot plugin is as easy as copying the appropriate XML file into DDT's plugin folder. An XML file for Open MPI can be found in the Marmot source tree (in SRC/TOOLS/MarmotDDTPlugin/OpenMPI):

```
$cp marmot_ddt_plugin_openmpi_alpha.xml <DDT_DIR>/plugins/
```

Now you may startup DDT to see whether the Marmot plugin has been recognized. Go to *Session* → *New Session*→ *Run...* and switch to the *Advanced* view. You should now see the Marmot plugin:

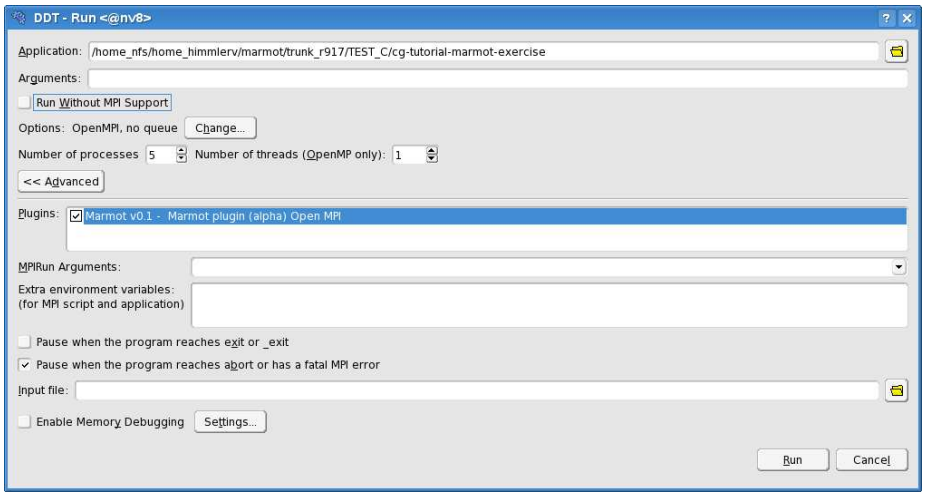

Figure 2: DDT plugin selection

3 Usage

3.1 Compilation

3.1.1 C and C++ programs

To compile a C/C++ application with Marmot, you can use *marmotcc*/*marmotcxx* which are wrapper scripts invoking *mpicc*/*mpicxx* from the underlying MPI library. Compilation of a C/C++ application should be as easy as typing

```
$marmotcc -o basic basic.c
```
or
```
$marmotcxx -o basic basic.cc
```

However, this might sometimes lead to problems, especially in the linking step. In this case, you will need to modify *marmotcc*/*marmotcxx* for your needs. To see what exactly *marmotcc*/*marmotcxx* does, you can type

```
$marmotcc --marmot-verbose -o basic basic.c
```

and should see something like

```
mpicc -I/usr/local/marmot/include -o basic basic.c -L/usr/local/marmot/lib
-lmarmot-profile -lmarmot-core -L/usr/local/mpi/openmpi-1.2.8/lib -lmpi
-lpthread -L/usr/lib -lxml2 -L/usr/local/cube/lib -lcube3 -L/usr/lib
-lstdc++
```

To see the invocation of the underlying compiler, type `$marmotcc -show`

```
-o basic basic.c
```

```
gcc -I/usr/local/mpi/openmpi-1.2.8/include -pthread -I/usr/local/marmot/include
-o basic basic.c -L/usr/local/marmot/lib -lmarmot-profile -lmarmot-core
-L/usr/local/mpi/openmpi-1.2.8/lib -lmpi -lpthread -L/usr/lib -lxml2
-L/usr/local/cube/lib -lcube3 -L/usr/lib -lstdc++ -L/usr/local/mpi/openmpi-1.2.8/lib
-lmpi -lopen-rte -lopen-pal -ldl -Wl,--export-dynamic -lnsl -lutil
-lm -ldl
```

As you can see, *marmotcc* passes the show option to *mpicc* and you get the compiler command and its command line options. Depending on the compiler and the MPI library, you might for example need to link some other libraries, too.

Another approach is to use the mpicc command or the "plain" compiler command itself. You could start with:

```
$mpicc -o basic basic.c -L/usr/local/lib -lmarmot-profile
-lmarmot-core -lstdc++
```

or you can invoke the compiler directly:

```
$gcc -o basic basic.c -L/usr/local/lib -lmarmot-profile
```

```
-lmarmot-core -lstdc++ -L/usr/local/mpi -lmpi
```

Either way you go, make sure you link the libraries in the correct order, i.e. the Marmot libraries have to be linked prior to the MPI libraries. You won't get an error message because of the wrong linking order but Marmot simply won't work.

Further, the compiler wrappers try to redirect the MPI header. If successful, your application will use Marmots provided MPI header, which in turn includes the MPI implementation provided header. As a result, Marmot can add source code information to MPI calls, thus providing more detailed output.

To get basic usage hints just type *marmotcc*/*marmotcxx* without any arguments.

3.1.2 Fortran programs

Compiling fortran programs basically works the same way as compiling C programs. One can use *marmotf77*:

```
$marmotf77 -o basic basic.f
```

or one can manually compile and link the application. This might look like this:

```
$gfortran -c basic.f -I/usr/local/mpich2/include
```

```
$gfortran basic.o -L/usr/local/lib -lmarmot-profile -lmarmot-fortran
-lmarmot-core -lstdc++ -L/usr/local/mpich2/lib -lmpich -lpthread -lrt
```

Be sure to link the Marmot libs prior to the MPI libs and don't forget to link *libmarmot-fortran*, too.

The Marmot compiler wrappers will attempt a source to source translation, which adds additional source code data to the MPI calls. For some Fortran applications this may fail, use the extra argument *–marmot-noinst* in these cases. You can also run *marmotf77*/*marmotf90* without any arguments to see all options.

3.1.3 Hybrid programs

A Marmot installation configured with OpenMP support works like a normal installation. You can use *marmotcc* (*marmotf77* resp.) to compile an link your programs. The compiler wrappers will automatically include the necessary MPI library and the OpenMP flag, include paths and libraries.

3.2 Running the application

3.2.1 Invocation

To run the application, one has to add an additional process working as debug server, i.e. one needs (n+1) instead of n processes:

```
$mpirun -np (n+1) foo
```

Marmot's output is written to a logfile (see Section 3.3).

Note on hybrid programs:

Running a program compiled with an OpenMP supporting version of Marmot works as usual. Bur you might have to set additional environmental variables in order to enable MPI/OpenMP interoperability on your system.

3.2.2 Influential environment variables

The following environment variables affect Marmot's behaviour at runtime:

MARMOT_DEBUG_MODE	0: errors, 1: errors and warnings, 2: errors, warnings and remarks are reported (default)
MARMOT_LOGFILE_TYPE	0: ASCII Logging (default), 1: HTML Logging, 2: CUBE Logging (when enabled via configure), 3: VAMPIR Logging (when enabled via configure) (not implemented yet)
MARMOT_LOG_FILTER_COUNT	Limits how often a sepecific problem is recoreded (default: 50)
MARMOT_LOG_FLUSH_TYPE	0: Flush on Error (default), 1: Immediate Flush
MARMOT_RESOURCE_TRACE_SELECTION	0: On (Full resource tracing if source-locations are present and quantitative tracing otherwise) (default), 1: Off (no additional resource tracing, some tracing is still done in order to detect certain errors)
MARMOT_INTERFACE_MODE	0: C interface, 1: Fortran interface, interface mode is set automatically
MARMOT_SERIALIZE	0: code is not serialized, 1: code is serialized (default)
MARMOT_TRACE_CALLS	1: calls are traced with output to stderr, traceback in case of a deadlock is possible (default),

	0: calls are traced without output to stderr, traceback in case of a deadlock is possible, -1: calls are not traced, traceback in case of a deadlock is NOT possible.
MARMOT_MAX_PEND_COUNT	maximum number of calls that are traced back (default 10)
MARMOT_MAX_TIMEOUT_DEADLOCK	maximum message time (default 10^6 μs)
MARMOT_MAX_TIMEOUT_SERIALIZE	maximum message time (default 10^3 μs)

3.3 Marmot's output

Marmot's output can be set to one of currently three modes. *ASCII* logging, *HTML* logging and *Cube* logging. The mode is set via the environment variable *MARMOT_LOGFILE_TYPE* (see Section 3.2.2). In the following, the different modes are compared by having a look at the output of a small program named *deadlock1.c* in the directory *TEST_C*. The source code can be seen in Figure 3.

As the name suggests, this program deadlocks. So let's have a look at Marmot's output, depending on the logging mode.

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    const int COUNT = 1;
    const int MSG_TAG_1 = 17;
    const int MSG_TAG_2 = 18;
    int rank = -1;
    int size = -1;
    int dummy = 0;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf(" I am rank %d of %d PEs\n", rank, size);
    if (size < 2)
    {
      fprintf(stderr, " This program needs at least 2 PEs!\n");
    }
    else
    {
      if (rank == 0)
      {
         MPI_Recv(&dummy, COUNT, MPI_INT, 1, MSG_TAG_1, \\
         MPI_COMM_WORLD, &status);
         MPI_Send(&dummy, COUNT, MPI_INT, 1, MSG_TAG_2, \\
         MPI_COMM_WORLD);
      }
      if (rank == 1)
      {
         MPI_Recv(&dummy, COUNT, MPI_INT, 0, MSG_TAG_2, \\
         MPI_COMM_WORLD, &status);
         MPI_Send(&dummy, COUNT, MPI_INT, 0, MSG_TAG_1, \\
         MPI_COMM_WORLD);
      }
    }
    MPI_Finalize();
    return 0;
}
```

Figure 3: deadlock1.c

3.3.1 ASCII logging

```
1: Note from rank 0 with  Text: performing
On Call: MPI_Init

2: Note from rank 1 with  Text: performing
On Call: MPI_Init

3: Note from rank 0 with  Text: performing
On Call: MPI_Comm_rank

4: Note from rank 1 with  Text: performing
On Call: MPI_Comm_rank

5: Note from rank 0 with  Text: performing
On Call: MPI_Comm_size

6: Note from rank 1 with  Text: performing
On Call: MPI_Comm_size

7: Note from rank 0 with  Text: performing
On Call: MPI_Recv

8: Note from rank 1 with  Text: performing
On Call: MPI_Recv

8: Error global message with  Text: WARNING: all clients are pending!
Last calls (max. 10) on node 0:
 timestamp 1: MPI_Init(*argc, ***argv)

 timestamp 3: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)

 timestamp 5: MPI_Comm_size(comm = MPI_COMM_WORLD, *size)

 timestamp 7: MPI_Recv(*buf, count = 1, datatype = MPI_INT, source = 1, tag = 17, comm = MPI_COMM_WORLD, *status)


Last calls (max. 10) on node 1:
 timestamp 2: MPI_Init(*argc, ***argv)

 timestamp 4: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)

 timestamp 6: MPI_Comm_size(comm = MPI_COMM_WORLD, *size)

 timestamp 8: MPI_Recv(*buf, count = 1, datatype = MPI_INT, source = 0, tag = 18, comm = MPI_COMM_WORLD, *status)
```

Figure 4: ASCII logging (excerpt)

The output file in ASCII logging mode is named *Marmot_EXE_YYYYMMDD_hhmmss.txt*, where EXE denotes the name of your executable, and YYYYMMDD_hhmmss is a timestamp.

3.3.2 HTML logging

3	0	Note	Text: performing Call: MPI_Comm_rank	Unknown	
4	1	Note	Text: performing Call: MPI_Comm_rank	Unknown	
5	0	Note	Text: performing Call: MPI_Comm_size	Unknown	
6	1	Note	Text: performing Call: MPI_Comm_size	Unknown	
7	1	Note	Text: performing Call: MPI_Recv	Unknown	
8	0	Note	Text: performing Call: MPI_Recv	Unknown	
8	Global	Error	Text: WARNING: all clients are pending! Last calls (max. 10) on node 0: timestamp 2: MPI_Init(*argc, ***argv) timestamp 3: MPI_Comm_rank(comm = MPI_COMM_NULL, *rank) timestamp 5: MPI_Comm_size(comm = MPI_COMM_NULL, *size) timestamp 8: MPI_Recv(*buf, count = 1, datatype = MPI_DATATYPE_NULL, source = 1, tag = 17, comm = MPI_COMM_NULL, *status) Last calls (max. 10) on node 1: timestamp 1: MPI_Init(*argc, ***argv) timestamp 4: MPI_Comm_rank(comm = MPI_COMM_NULL, *rank) timestamp 6: MPI_Comm_size(comm = MPI_COMM_NULL, *size) timestamp 7: MPI_Recv(*buf, count = 1, datatype = MPI_DATATYPE_NULL, source = 0, tag = 18, comm = MPI_COMM_NULL, *status)	Unknown	Infos see MPI-Standard

Figure 5: HTML logging (excerpt)

The output file in HTML logging mode is named *MarmotLog_EXE_YYYYMMDD_hhmmss.html*.

3.3.3 CUBE logging

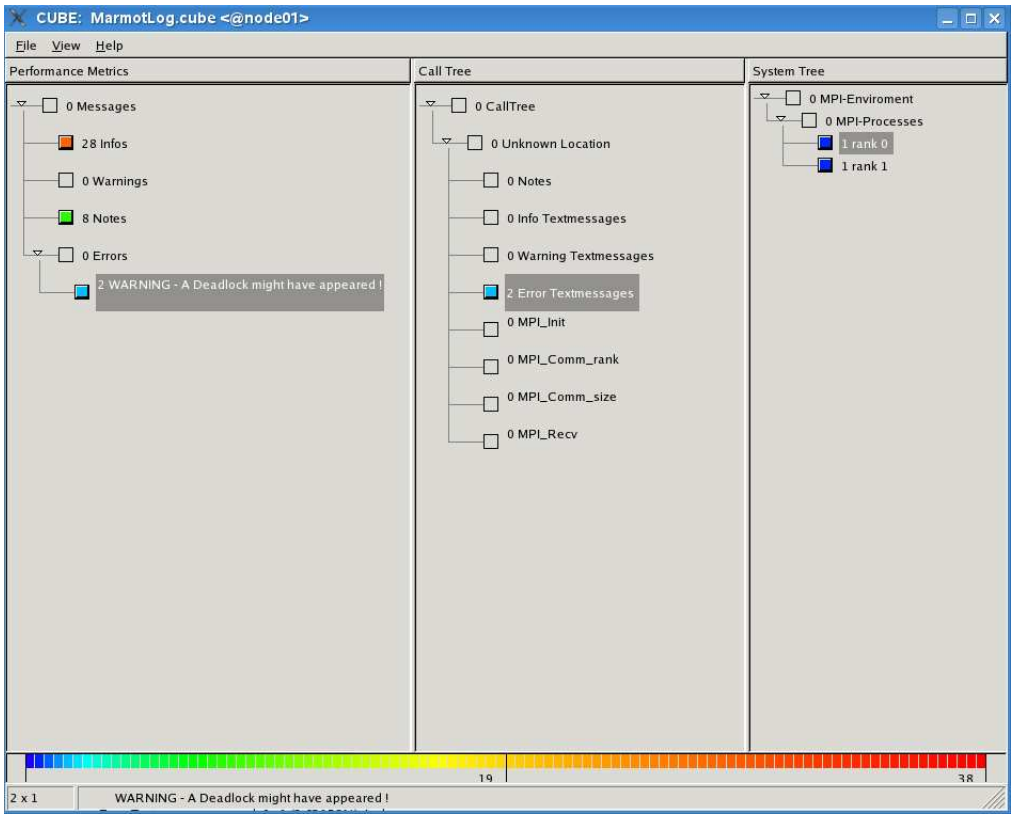

Figure 6: CUBE logging

The output file in CUBE logging mode is named *MarmotLog_EXE_YYYYMMDD_hhmmss.cube*. Further a folder named *MARMOT_HTML* is created, which contains detailed information. To view it with the Cube browser, type

`$cube MarmotLog_EXE_YYYYMMDD_hhmmss.cube` or `$cube3 MarmotLog_EXE_YYYYMMDD_hhmmss.cube` for the most recent version of the Cube browser.

3.3.4 Running DDT with Marmot's plugin

Install the Marmot plugin as described in section 2.6. Compile your program with the original MPI compiler wrappers (e.g. *mpicc*) and don't forget to include debug information in the executable (usually with *-g*):

```
$mpicc -g -o yourprogram yourprogram.c
```

Launch DDT and go to *Session* → *New Session* → *Run ...*. In the advanced view, select the Marmot plugin:

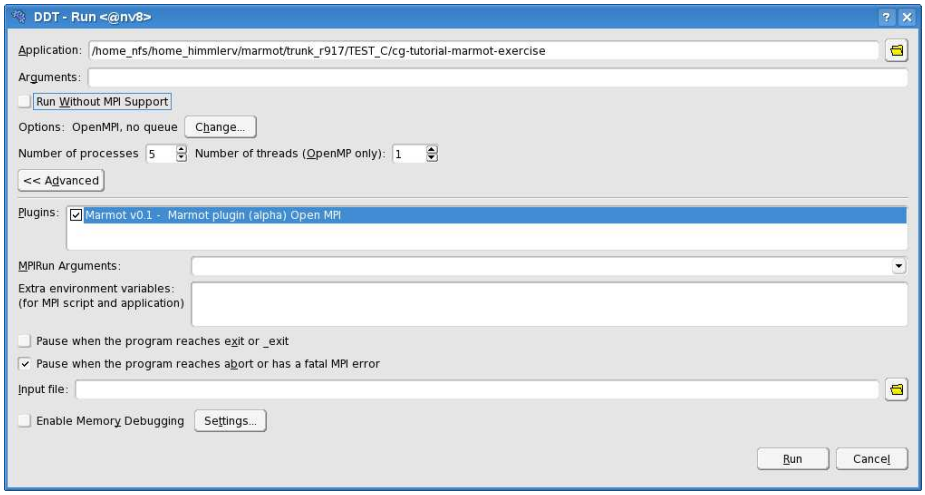

Figure 7: DDT plugin selection

Run the application with the original number of processes. DDT will automatically add one process for Marmot's debug server, which is not displayed. When Marmot detects an error DDT will pause the execution and pop up a window:

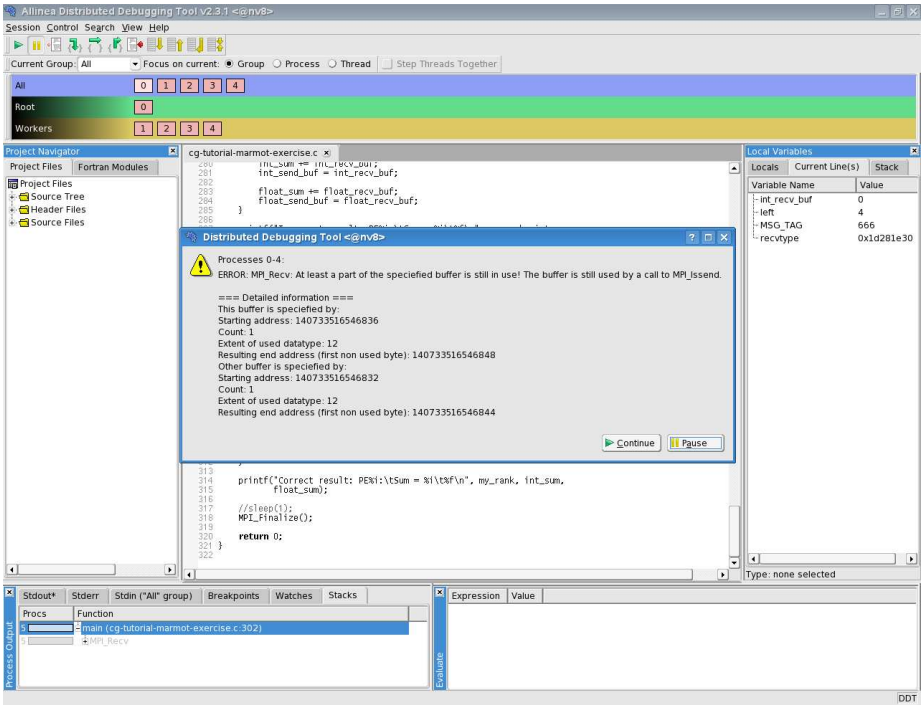

Figure 8: Marmot detects error from within DDT

When run from within DDT Marmot still creates its logfile for later analysis.

A Installation Examples

Marmot was (and is being) tested on a variety of different machines, including installations at HLRS and ZIH. As mentioned in Section 2.3 on page 7, it is usually necessary to configure Marmot with different options, depending on the specific hardware and software of the target platform. These adjustments have to be made because of

- the used compiler (e.g., GCC or Intel)
- the used MPI library (e.g., OpenMPI or IntelMPI) and the compiler it was built with
- the used interconnection (e.g. Myrinet)
- the paths where libraries and/or include files reside
- the desired features Marmot can be compiled with (e.g. CUBE support)

A.1 Overview

In the following, four installation processes are presented in more detail, to give the user an idea on how to configure and install Marmot, as well as running a simple test program.

1. "**Cacau**": A cluster of 204 Intel Xeon EM64T nodes, Node-Node interconnect Voltaire Infiniband, Intel Compiler, MPI library, OpenMPI, Batch system: Torque, Maui scheduler
2. "**A1**": A NEC SX-8 cluster with 72 SX-8 nodes, each having 8 CPUs, Node-Node interconnect IXS, NEC SX (cross)compiler, MPI library: NECmpi, Batch system: NQSII
3. "**Windows HPC Server 2008 cluster**": A 17 node cluster with Microsoft HPC pack v.2
4. "**bwGrid**": A cluster with 868 Quad-Core Intel Xeon processors (3472 cores). Infiniband interconnection. Open MPI, MVAPICH

Of course it is unlikley that you will build Marmot in exactly the same environment, so at least the used paths will differ from yours.

A.2 Configuration, installation and compilation

A.2.1 Cacau

Configuration and installation:

```
$./configure --with-mpi-dir=/opt/OpenMPI/1.2.2
--disable-tests --disable-doc
```

```
   --prefix=$INSTALL_DIR
   CXXFLAGS="-g -Wall -I/opt/OpenMPI/ 1.2.2/include/openmpi"
   CFLAGS="-g -Wall "
   FFLAGS="-g -Wall "
   CXX="icpc"
   CC="icc"
   CPP="icc -E"
   F77="ifort"
   $make
   $make install
```

Compilation of a test program:

```
   $cd TEST_C
   $marmotcc -o deadlock1 deadlock1.c --marmot-verbose
   mpicc -o deadlock1 deadlock1.c -L/usr/local/marmot/lib -lmarmot-profile
-lmarmot-core -L/opt/OpenMPI/1.2.2/lib -lmpi -lpthread -L/usr/lib -lstdc++
-I/usr/local/marmot/include
```

Running the test program:

```
   $mpirun -np 3 ./deadlock1
   I am rank 0 of 2 PEs
   I am rank 1 of 2 PEs
   WARNING: all clients are pending!  (Details see the LogFile)
```

A.2.2 A1

Configuration and installation:

```
   $cd MARMOT
   $export SX_BASE_CROSS=/SX/opt/crosskit/inst/
   $export PATH=/SX/usr/bin/:$PATH
   $export SX_BASE_CPLUS=/SX/opt/sxc++/inst
   $export SX_BASE_F90=/SX/opt/sxf90/inst
   $export SX_BASE_MPI=/SX/opt/mpisx/inst
   $export CC="sxcc"
   $export CXX="sxc++"
   $export F77="sxf90"
   $export CPP="sxcc -E"
   $export AR="sxar"
   $export CXXFLAGS="-K exceptions"
   $export PRELINK=1
   $export RANLIB="sxar -s"
   $./configure --with-mpi-dir=/opt/NECmpi --disable-tests --disable-doc
--prefix=$INSTALL_DIR --host=sx8-nec-superux
```

```
$make
$make install
```

Compilation of a test program:

```
$cd TEST_C
$sxc++ deadlock1.c -c
$sxmpic++ -K exceptions -mpiprof -o deadlock1 deadlock1.o -I../SRC/INCLUDE
-L../LIB -lmarmot-profile -lmarmot-core -f90lib
```

Running the test program:

```
$mpirun -np 3 ./deadlock1
I am rank 0 of 2 PEs
I am rank 1 of 2 PEs
WARNING: all clients are pending!  (Details see the LogFile)
```

A.2.3 Windows HPC Server 2008 cluster

Configuration and installation:

- Create a directory for the generated files
- Start *CMake* and select the root of the marmot sources and the path for the generated files
- Click on the "Configure" button
- Select a generator according to the VisualStudio version installed (Please note that if you intend to enable Fortran support it is advisable to select the NMake generator)
- Change the field "`USED_MPI_PACKAGE`" to `MPI` (Default find module is `MPICH`)
- Adjust the configuration options to your needs
- Click on the "Configure" button again and then "Generate"
- Open the generated solution file with VisualStudio and build the "`ALL_BUILD`" target and then the "`INSTALL`" target
- Change the build configuration from "Debug" to "Release" and reiterate
- All the necessary files can be found in the directory set in `CMAKE_INSTALL_PREFIX`. You may copy/move the contents of this directory anywhere you wish. Set the environment variable `MARMOT_HOME` to the directory you installed/copied marmot to. (It is not necessary to set `MARMOT_HOME` if you used the installer to set-up Marmot.)

Compilation of a test program:

For VisualStudio 2008 there is an example solution file in `$MARMOT_HOME\share\marmot\examples\marmot-vs-demo`. The project should compile this small example as longs as the `$MARMOT_HOME` environment variable is set and the Microsoft HPC SDK is installed. If you prefer to use *CMake* in your project there is an example `CMakeLists.txt` file in `$MARMOT_HOME\share\marmot\examples\mpihello` which you may use as a starting point.

Running the test program:

Open a command shell and change to the `Debug` or `Release` subdirectory (depending which configuration you built). Launch the test application with:

```
$mpiexec -n 3 sdk-demo.exe
```

An alternative way to launch applications and also to have a look at the marmot messages is the usage of the Marmot-Addin for VisualStudio.

- Either use the marmot installer or start `$MARMOT_HOME\bin\register_marmot_addin.bat` in order to register the Addin in VisualStudio.
- Open your project file and select a "Startup Project".
- Compile the subproject if it was not built yet.
- Click on the leftmost button in the Marmot-Addin toolbar (this will setup a commandline with mpiexec some environment variables and the correct absolute path to the executable. Note: If you edit this line manually you have to confirm any changes made by hitting the return-key.
- Click on the "Run" button in the Marmot-Addin toolbar to launch the application.
- You should see the marmot output in the build pane of VisualStudio (just like regular error or warning messages coming from the compiler).
- If you use VisualStudio 2008 or later you will get an extra tool window that displays the Marmot output.

A.2.4 bwGrid

Configuration and installation:

```
$cd MARMOT
$./configure --prefix=/opt/bwgrid/debugger/marmot
--with-mpi-dir=/opt/bwgrid/mpi/mvapich2/1.0.3-gcc --disable-tests
--enable-shared-libs F77=gfortran CXXFLAGS=-DMPICH_IGNORE_CXX_SEEK
```

```
$make
$make install
```

Compilation of a test program:

```
$cd TEST_C
$marmotcc --marmot-verbose -o pending-msg pending-msg.c
mpicc -I/opt/bwgrid/debugger/marmot/include -o pending-msg pending-msg.c
-L/opt/bwgrid/debugger/marmot/lib -lmarmot-profile -lmarmot-core
-L/opt/bwgrid/mpi/mvapich2/1.2p1-intel-10.1/lib -lmpich -lpthread -L/usr/lib
-lstdc++
```

Running the test program:

```
$mpirun -np 5 ./pending-msg
We call Finalize when there is still a non-received message pending
I am rank 0 of 4 PEs
I am rank 2 of 4 PEs
I am rank 1 of 4 PEs
I am rank 3 of 4 PEs
```

References

[SCALASCA] Scalable Performance Analysis of Large-Scale Applications, FZ Jülich, available at http://www.fz-juelich.de/jsc/scalasca/

[AUTOCONF] Autoconf - GNU Project - Free Software Foundation (FSF), http://www.gnu.org/software/autoconf/

[AUTOMAKE] GNU Automake - GNU Project - Free Software Foundation (FSF), http://www.gnu.org/software/automake/

[DOXYGEN] Doxygen, http://www.doxygen.org/

[CMAKE] CMake, http://www.cmake.org

[ALLINEA] Allinea Software, http://www.allinea.com