



**U S E R   M A N U A L**  
**M A R M O T**

**WP2.2 - Code Debugging and Verification Tool (MARMOT)**

---

Document Filename:	<b>CG2.2-v0.1-UST001-MARMOTUserManual-v1.1.12.doc</b>
Work package:	<b>WP2</b>
Partner(s):	<b>USTUTT, CSIC</b>
Lead Partner:	<b>USTUTT</b>
Config ID:	<b>CG2.2-v0.1-UST001-MARMOTUserManual-v1.1.12</b>
Document classification:	<b>PUBLIC</b>

---

Abstract:

This is the user manual for the MPI analysis and checking tool MARMOT.

---

### Document Log

Version	Date	Summary of changes	Author
0.1	13/12/2004	Draft version	Bettina Krammer
	26/01/2005	Verified by the QE	Robert Pajak

---

## CONTENTS

<b>COPYRIGHT NOTICE .....</b>	<b>4</b>
<b>1. INTRODUCTION.....</b>	<b>5</b>
1.1. ABBREVIATIONS AND ACRONYMS .....	5
1.2. REFERENCES AND SOURCE CODE .....	5
<b>2. PRODUCT USAGE .....</b>	<b>7</b>
2.1. RUNNING THE PRODUCT .....	7
2.1.1. <i>Operating Requirements</i> .....	7
2.1.2. <i>Step-by-Step User Setup</i> .....	9
2.2. BASIC OPERATION .....	9
2.2.1. <i>Simple Job Submission</i> .....	9
2.2.2. <i>Job Submission with Migrating Desktop</i> .....	10
2.2.3. <i>Log Files</i> .....	15
2.3. ADVANCED FEATURES.....	17
2.4. KNOWN PROBLEMS.....	18
<b>3. INTERFACE REFERENCE GUIDE.....</b>	<b>19</b>
<b>4. TROUBLESHOOTING Q&amp;A .....</b>	<b>20</b>
4.1. HOW DOES MARMOT WORK?.....	20
4.2. WHAT ERRORS CAN MARMOT DETECT? .....	21
4.3. HOW CAN USERS MODIFY MARMOT? .....	24
4.4. WHERE CAN I FIND MORE INFORMATION ON MARMOT?.....	24
<b>5. CONTACT INFORMATION AND CREDITS.....</b>	<b>25</b>
<b>6. THE GPL LICENSE AGREEMENT .....</b>	<b>26</b>

---

## **COPYRIGHT NOTICE**

Copyright (c) 2005 by High Performance Computing Center Stuttgart (HLRS), affiliated with University of Stuttgart. All rights reserved.

Use of this product is subject to the terms and licenses stated in the GPL license agreement, see Section 6 for details.

This research is partly funded by the European Commission IST-2001-32243 Project “CrossGrid”.

## 1. INTRODUCTION

The Message Passing Interface (MPI) is a widely used standard for writing parallel programs. However, the MPI-1.2 standard, with its 129 calls, has a size and complexity that makes it possible to use the MPI API incorrectly. According to our experience there are several reasons for this:

- Developers do not only have to face all the problems that occur in serial programming. In addition, parallel applications get more and more complex and especially with the introduction of optimisations like the use of non-blocking communication also more error prone.
- MPI programs do not always behave deterministically. Deadlocks or race conditions may appear, depending on the platform environment or on the MPI implementation.
- The MPI standard leaves many decisions to the implementation, e.g. whether or not a standard communication is blocking. This implementation-defined behaviour may cause problems when porting an application from one platform to another, for example, when porting an application from a local platform to the CrossGrid testbed.

Debugging MPI programs has been addressed in various ways. The different solutions can be roughly grouped in four different approaches: classical debuggers, special MPI libraries and other tools that may perform a run-time or post-mortem analysis.

- Classical debuggers have been extended to address MPI programs. This is done by attaching the debugger to all processes of the MPI program. There are many parallel debuggers, among them the very well-known commercial debugger Totalview. The freely available debugger gdb has currently no support for MPI, however, it may be used as a back-end debugger in conjunction with a front-end that supports MPI, e.g. mpigdb. Another example of such an approach is the commercial debugger DDT by streamline computing, or the non-freely available p2d2.
- The second approach is to provide a debug version of the MPI library (e.g. mpich.) This version is not only used to catch internal errors in the MPI library, but it also detects some incorrect usage of MPI by the user, e.g. a type mismatch of sending and receiving messages.
- Another possibility is to develop tools dedicated to finding problems within MPI applications. At present, three different message-checking tools are under active development: MPI-CHECK, Umpire and MARMOT. MPI-CHECK is currently restricted to Fortran code and performs argument type checking or finds problems like deadlocks. Like MARMOT, Umpire uses the profiling interface. Unfortunately, Umpire is not freely available. These three tools all perform their analysis at runtime.
- The fourth approach is to collect all information on MPI calls in a trace file, which can be analysed by a separate tool after program execution. A disadvantage with this approach is that such a trace file may be very large. However, the main problem is guaranteeing that the trace file is written in the presence of MPI errors, because the behaviour after an MPI error is implementation defined.

### 1.1. ABBREVIATIONS AND ACRONYMS

MPI	Message Passing Interface
API	Application Programming Interface
MD	Migrating Desktop

### 1.2. REFERENCES AND SOURCE CODE

- MARMOT's source code can be found at

---

[http://savannah.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp2/wp2\\_2-verif/src/MARMOT/](http://savannah.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp2/wp2_2-verif/src/MARMOT/)

- MARMOT's RPMs can be found at

<http://gridportal.fzk.de/distribution/crossgrid/autobuilt/i386-rh7.3-gcc3.2.2/wp2/RPMS/>

The latest tag is currently v1.1.12.

- More information can be found at MARMOT's homepage

<http://www.hlr.de/organization/tsc/projects/marmot>

especially on the publications site

<http://www.hlr.de/organization/tsc/projects/marmot/pubs.html>

---

## 2. PRODUCT USAGE

- 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 log file.
- The tool can be configured via environment variables.
- Currently MPI-1.2 standard is supported.
- See also the README in MARMOT's distribution, and the installation guide and developer's guide, to be found in MARMOT's download area at [http://savannah.fzk.de/distribution/crossgrid/crossgrid/wp2/wp2\\_2-verif/](http://savannah.fzk.de/distribution/crossgrid/crossgrid/wp2/wp2_2-verif/)

### 2.1. RUNNING THE PRODUCT

- For CrossGrid, there are two distributions, namely cg-wp2.2-marmot for mpich-p4 and cg-wp2.2-marmot-g2 for mpich-g2. The rpms for CrossGrid install the MARMOT files to the /opt/cg directory. The subdirectories are named cg-wp2.2-marmot and cg-wp2.2-marmot-g2, respectively.
  - The MARMOT libraries libdmpi.a, libmpo.a, libfmpo.a will be installed in /opt/cg/lib/cg-wp2.2-marmot\*. These are the MARMOT libraries that have to be linked to an application to make use of the tool.
  - README, CONFIG-EXAMPLES, DEPENDENCIES, INSTALL, USERS\_GUIDE will be installed in /opt/cg/share/doc/cg-wp2.2-marmot\*. This is the most important documentation for MARMOT. (USERS\_GUIDE is not contained in tag v1\_1\_11)
  - basic, basic.jdl, basic-g2.jdl, cg-tutorial-marmot-exercise, cg-tutorial-marmot-exercise.jdl will be installed in /opt/cg/bin/cg-wp2.2-marmot\*/C. These are simple test programs written in C to test MARMOT's functionality. More C examples may be found in MARMOT's source code in the TEST\_C directory.
  - basic, basic.jdl, basic-g2.jdl will be /opt/cg/bin/cg-wp2.2-marmot\*/F. These are simple test programs written in Fortran to test MARMOT's functionality. More Fortran examples may be found in MARMOT's source code in the TEST\_F directory.
- See also the installation guide.

#### 2.1.1. Operating Requirements

MARMOT itself has some requirements as described in the following subsections. In addition to that, further requirements may be needed by the application to be checked.

##### 2.1.1.1. Hardware requirements

MARMOT does not require any special hardware. MARMOT has been tested on the following platforms:

- NEC SX5, SX6
- IBM Regatta

- Linux IA32/64 clusters
- Some old platforms, such as Cray T3e, Hitachi SR8000,...

### 2.1.1.2. Software requirements

- See also the DEPENDENCIES file in MARMOT's distribution and the installation guide.
- MPI implementation
  - 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. 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.
  - The MPI implementation itself may require some other software, for example globus libraries when using mpich-g2.
  - Some of MARMOT's source files include mpi.h, therefore mpi (i.e. at least mpi.h) is needed for the compilation of MARMOT to make the MARMOT libraries, which can then be linked to an application.
- C++ compiler:
  - 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 7.0 (or later) or the KAI C++ 4.0 compiler have been used successfully. MARMOT is also tested with xLC\_r for AIX Compiler, Version 6.
  - 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).
- 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.
- C compiler:
  - To support C applications, a C compiler is required (any C compiler should do).
- Other tools that users may need for installation
  - make (tested with GNU make 3.79.1 and later versions) for compilation.
- Other tools that users do not necessarily need
  - Doxygen (tested with version 1.2.14 and later versions) is used to automatically generate documentation.

MARMOT's distribution comes with all the required files that users just have to run "configure" and then "make".

### 2.1.1.3. Grid infrastructure requirements

MARMOT needs the usual requirements to run an MPI job in the testbed, also depending on the application and on how we submit the job, for example with Migrating Desktop.

## 2.1.2. Step-by-Step User Setup

Applications have to be relinked with MARMOT in order to make use of this tool.

- Applications written in C can be compiled accordingly to this example:

```
gcc -I/opt/cg/mpich/include -c basic.c
g++ -o basic basic.o -L../LIB -ldmpi -lmpo -L/opt/cg/mpich/lib -lmpich
(with gcc = /opt/gcc-3.2.2/bin/gcc-3.2.2, g++ = /opt/gcc-3.2.2/bin/g++-3.2.2)
```

It is also possible to use mpicc, in this case it is very important to link the proper version of the libstdc++ (i.e. same version as was used to compile the MARMOT libraries, i.e. specify the correct path for -lstdc++):

```
mpicc -c basic.c
mpicc -o basic basic.o -L../LIB -ldmpi -lmpo -L/opt/cg/mpich/lib -lmpich \
-L/opt/gcc-3.2.2/lib -lstdc++
```

- Applications written in Fortran can be compiled accordingly to this example:  
(with g77 = /opt/gcc-3.2.2/bin/g77-3.2.2, g++ = /opt/gcc-3.2.2/bin/g++-3.2.2):

```
g77 -I/opt/cg/mpich/include -g -O2 -c basic.f
g++ -I/opt/cg/mpich/include -g -O2 -o basic basic.o \
-L../LIB -ldmpi -lfmpo -lmpo \
-L/opt/cg/mpich/lib -lmpich \
-L/opt/gcc-3.2.2/lib/gcc-lib/i686-pc-linux-gnu/3.2.2 \
-L/opt/gcc-3.2.2/lib/gcc-lib/i686-pc-linux-gnu/3.2.2/../../../../ -lfrtbegin \
-lg2c -lm -lgcc_s
```

It is also possible to use mpif77, in this case it is very important to link the proper version of the libstdc++ (i.e. same version as was used to compile the MARMOT libraries, i.e. specify the correct path for -lstdc++):

```
mpif77 -c basic.f
mpif77 -o basic basic.o -L../LIB -ldmpi -lfmpo -lmpo \
-L/opt/cg/mpich/lib -lmpich \
-L/opt/gcc-3.2.2/lib -lstdc++
```

- MARMOT's source code contains two subdirectories TEST\_C and TEST\_F with simple MPI test programs written in C/Fortran. Some of these programs are correct, some of these programs are erroneous to evoke MARMOT's warnings. Users may play with these programs to get a taste of MARMOT.
- For the Crossgrid tutorial there is a special program TEST\_C/cg-tutorial-marmot-exercise.c.
- Any application from WP 1 might be used but will not evoke enough warnings from MARMOT to really demonstrate its functionality.
- After installation, the directories /opt/cg/bin/cg-wp2.2-marmot/C and /opt/cg/bin/cg-wp2.2-marmot/F and /opt/cg/bin/cg-wp2.2-marmot-g2/C and /opt/cg/bin/cg-wp2.2-marmot-g2/F contain some example binaries.
- See also the USERS\_GUIDE and CONFIG-EXAMPLES file in MARMOT's distribution.

## 2.2. BASIC OPERATION

### 2.2.1. Simple Job Submission

- To run the application with MARMOT, 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 sent to stderr.

- For the use on the testbed, the simple test program basic is installed in /opt/cg/bin/cg-wp2.2-marmot/C. It only performs MPI\_Init and MPI\_Finalize. Use for example a jdl file like the following basic.jdl:

```
Executable          = "basic";
JobType              = "MPICH";
NodeNumber           = 3;
VirtualOrganisation = "cg";
StdOutput            = "basic.out";
StdError             = "$HOME/basic.err";
InputSandbox         = {"basic"};
OutputSandbox        = {"basic.out", "$HOME/basic.err"};
```

to submit your job via Resource Broker

```
$ edg-job-submit basic.jdl
```

You can watch the output with something like

```
$ tail -f basic.err
```

on the CE where the job runs, and later on, you can get the output with edg-job-get-output. Same for mpich-g2 jobs.

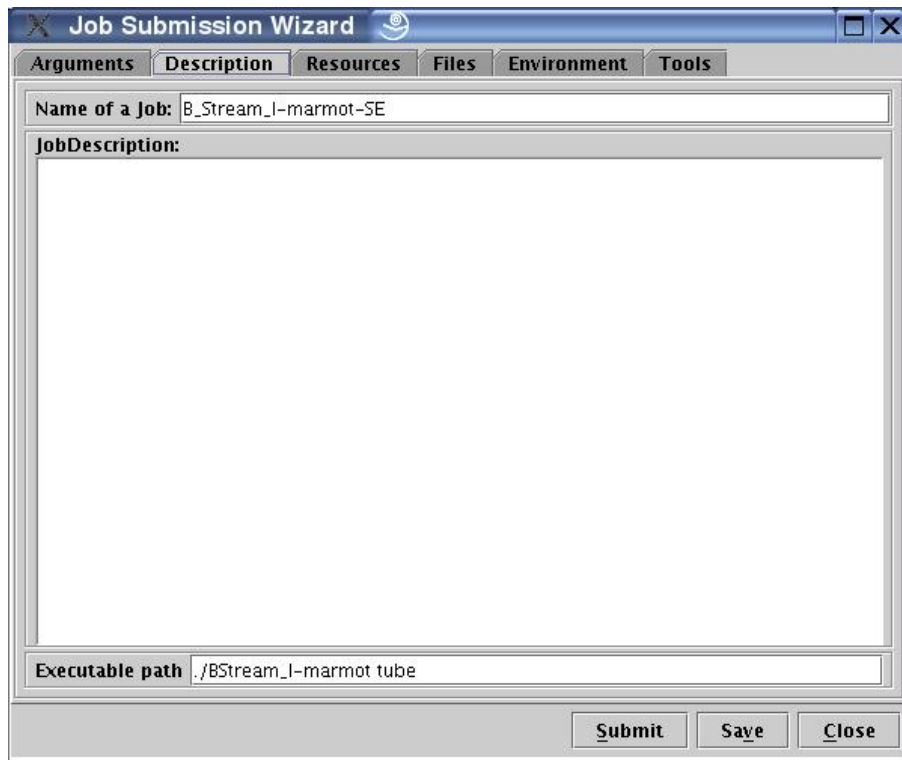
### 2.2.2. Job Submission with Migrating Desktop

It is also possible to submit jobs via Migrating Desktop. For example, for the bloodflow simulation from task 1.1, we used the following setup:

- After relinking the B\_Stream\_1 application with MARMOT, the resulting executable is called B\_Stream\_1-marmot. Together with the input files needed by the application, it is stored on a Storage Element. The Storage Element Settings within Migrating Desktop probably depend on the platform on which MD is running. We do not use Virtual Directories nor Home Directories on Computing Elements, because in the first case, MD has a problem, in the second case, the actual name of the home directory varies, depending on the testbed site, and may also change from time to time, as the accounts are pooled. Currently, Storage Elements seem to be the most reliable solution.
- We use the Job Wizard for a general command line job. If not indicated otherwise, we use the default entries, for example here, we simply leave the field empty:



- The next step is the job description, it is important to specify the argument “tube” needed by the application:



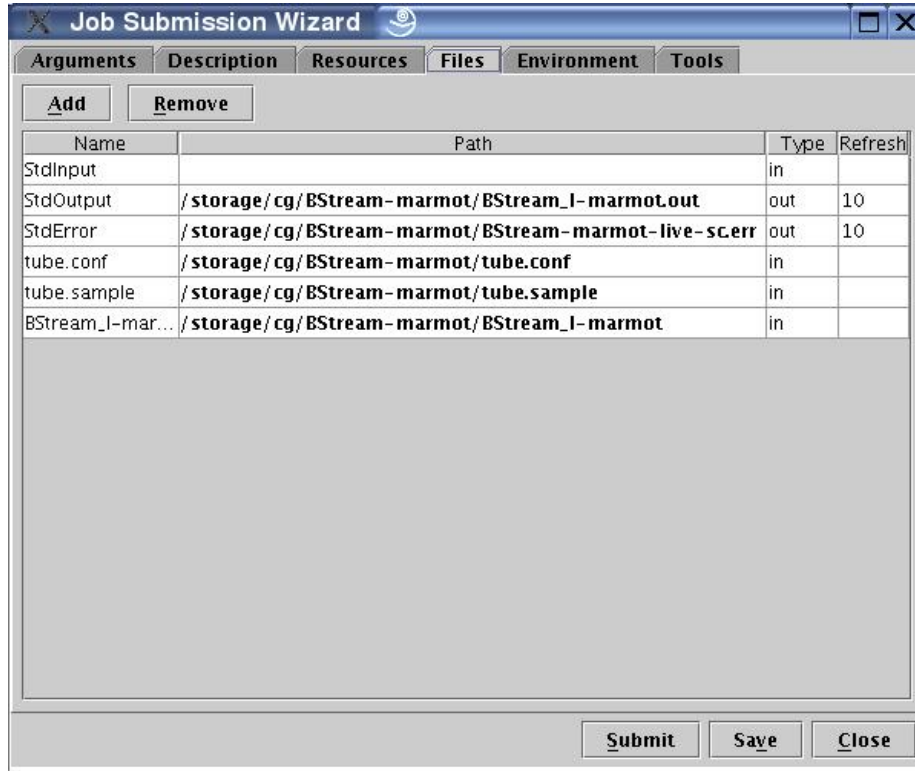
- The next step is to specify the job type and the number of nodes. Remember that MARMOT needs one additional process, i.e. the application itself will run on 3 nodes.

The screenshot shows the 'Job Submission Wizard' dialog box with the 'Resources' tab selected. The 'Type' sub-tab is active, showing 'Job Type' set to 'mpich'. Under 'Work Management', 'RB' and 'L&B' are both set to 'rb01.lip.pt'. A 'Node number' spinner is set to 4. At the bottom are 'Submit', 'Save', and 'Close' buttons.

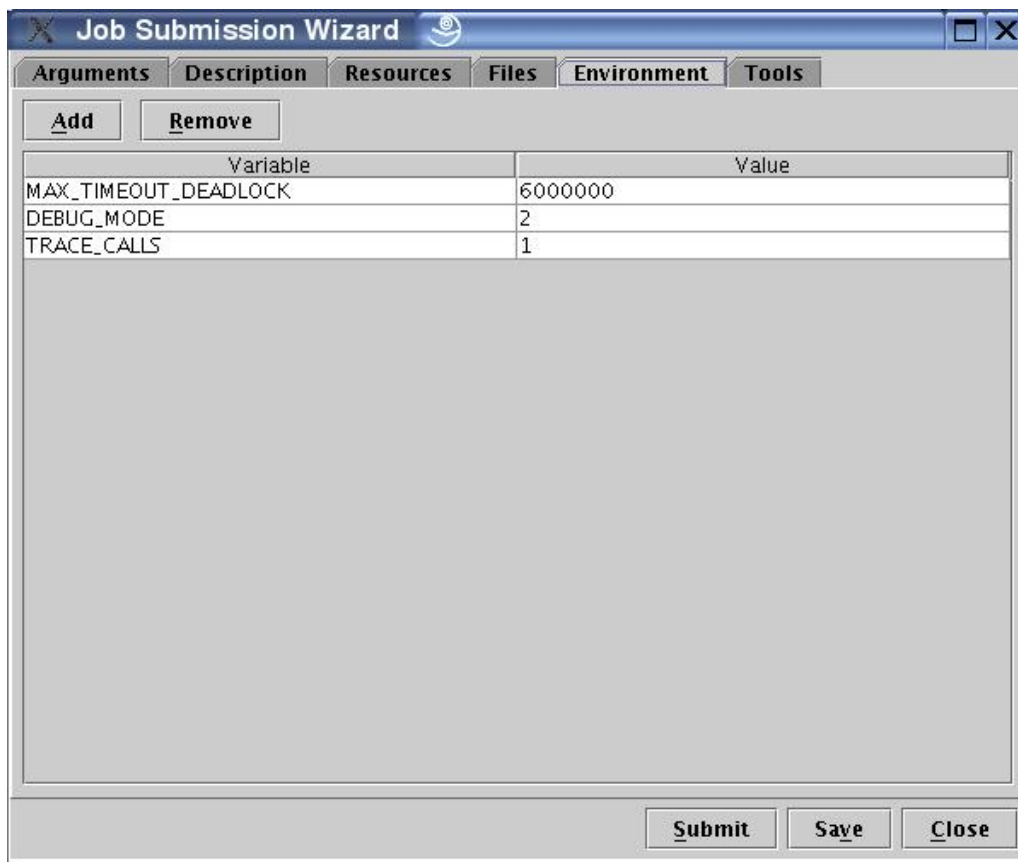
- Again, the correct executable path has to be specified. With this setup, the hostname may be “any”, i.e. the application may be executed on **any** testbed site.

This screenshot shows the 'Job Submission Wizard' dialog box with the 'Resources' tab selected. The 'Type' sub-tab is active. The 'Hostname' dropdown is set to '- any -'. The 'Application Executable' text field contains './BStream\_I-marmot tube'. The 'Application Software Runtime Environment' text field is empty. The 'Max Number of Submission Retries' section has a checkbox for 'Retry Count' which is unchecked, and a spinner set to 0. At the bottom are 'Submit', 'Save', and 'Close' buttons.

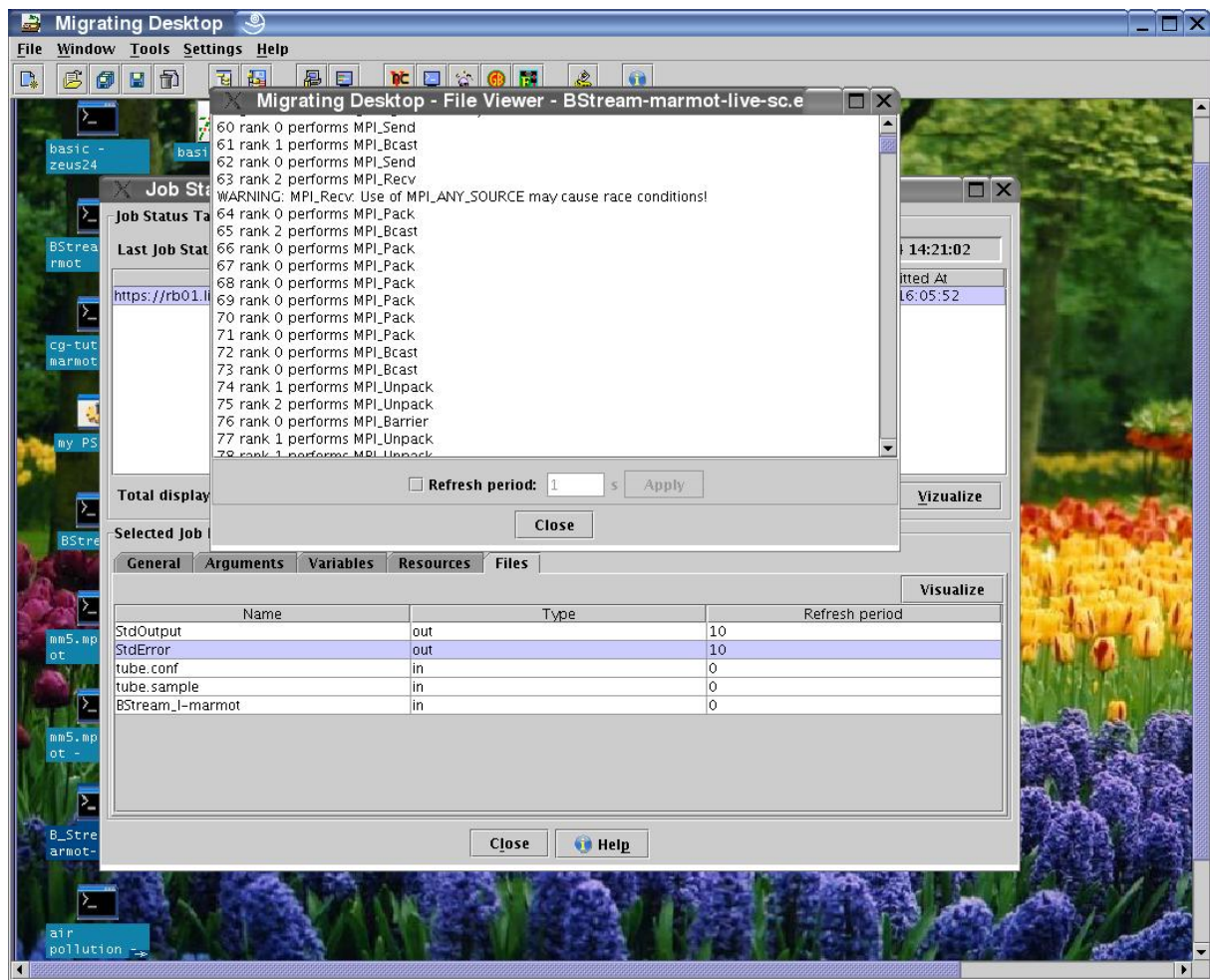
- It is very important to specify the input and output files correctly. In this example, we did not specify all of the output files. (The application produces some more output files, but in our case, we are only interested in MARMOT's output.) The files are located on a Storage Element. The refresh rate was set to 10 seconds. Lower values are **not** recommended for real applications, as this causes too much overhead for file transfer.



- It's also possible to specify environment variables (however, this never worked for me within Migrating Desktop, which is a problem of MD)



- Finally, the Job Monitor Dialog allows us to visualize the output at runtime. The output file is subsequently refreshed while the application is running.



### 2.2.3. Log Files

- MARMOT produces a human-readable log file.
- By default, MARMOT's logging is directed to stderr, users may redirect this output wherever they like.
- The log file will start with a header to give you an overview of the environment variables you set:

```

===== environmental variables =====
MARMOT_DEBUG_MODE=2
(0: errors,
 1: errors and warnings,
 2: errors, warnings and remarks are reported,
 default: 2)
MARMOT_INTERFACE_MODE=0
(0: C interface,
 1: Fortran interface,
 interface mode is set automatically)
MARMOT_SERIALIZE=0
(0: code is not serialized,
 1: code is serialized,
 default: 0)
MARMOT_TRACE_CALLS=1

```

```

(1: calls are traced with output to stderr,
    traceback in case of a deadlock is possible,
 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.
default: 1
Number of calls to be traced back in case of deadlock
can be set via MAX_PEND_COUNT.)
MARMOT_MAX_PEND_COUNT=10
(maximum number of calls that are traced back,
default: 10)
MARMOT_MAX_TIMEOUT_DEADLOCK=1000000
(maximum message time,
default: 1000000 microseconds)
MARMOT_MAX_TIMEOUT_SERIALIZE=1000
(maximum message time,
default: 1000 microseconds)
MARMOT_MAX_TIMEOUT_ONE=0
(maximum message time,
default: 0 microseconds)
MARMOT_MAX_TIMEOUT_TWO=0
(maximum message time,
default: 0 microseconds)
=====

```

- For example, the output of the cg-tutorial-marmot-exercise will contain warnings such as

```

18 rank 2 performs MPI_Type_struct
WARNING: MPI_Type_struct: blocklength[0]=0!
WARNING: MPI_Type_struct: datatype[0] is Fortran-Type!
WARNING: MPI_Type_struct: datatype[1] is optional!
WARNING: MPI_Type_struct: blocklength[0]=0!
WARNING: MPI_Type_struct: blocklength[0]=0!
19 rank 0 performs MPI_Type_struct
20 rank 1 performs MPI_Type_struct
21 rank 2 performs MPI_Type_struct
WARNING: MPI_Type_struct: datatype[0] is Fortran-Type!
WARNING: MPI_Type_struct: datatype[1] is optional!
WARNING: MPI_Type_struct: datatype[0] is Fortran-Type!
WARNING: MPI_Type_struct: datatype[1] is optional!
22 rank 0 performs MPI_Type_commit
23 rank 1 performs MPI_Type_commit
24 rank 0 performs MPI_Type_commit
25 rank 1 performs MPI_Type_commit
26 rank 2 performs MPI_Type_commit
NOTE: MPI_Type_commit: Datatype already committed!
NOTE: MPI_Type_commit: Datatype already committed!
27 rank 0 performs MPI_Address
28 rank 1 performs MPI_Address
29 rank 2 performs MPI_Type_commit
NOTE: MPI_Type_commit: Datatype already committed!
30 rank 0 performs MPI_Address
31 rank 1 performs MPI_Address
32 rank 2 performs MPI_Address

```

```

33 rank 0 performs MPI_Type_struct
34 rank 1 performs MPI_Type_struct
35 rank 2 performs MPI_Address
36 rank 0 performs MPI_Type_commit
37 rank 1 performs MPI_Type_commit
38 rank 2 performs MPI_Type_struct
39 rank 0 performs MPI_Issend
40 rank 1 performs MPI_Issend
41 rank 2 performs MPI_Type_commit
WARNING: MPI_Issend: count=0 !
WARNING: MPI_Issend: datatype is for reduction functions!
WARNING: MPI_Issend: count=0 !
WARNING: MPI_Issend: datatype is for reduction functions!
42 rank 0 performs MPI_Recv
WARNING: MPI_Recv: count = 0!
43 rank 1 performs MPI_Recv
44 rank 2 performs MPI_Issend

```

The warning for MPI\_Issend refers for example to

```

/* This will produce warnings:
 * the count is 0,
 * the types are for reduction functions.
 */
MPI_Issend(&int_send_buf, 0, MPI_LONG_INT, right, MSG_TAG,
          MPI_COMM_WORLD, &request);

```

where we had deliberately put some errors.

- More details can be found in MARMOT's tutorial description, see <http://www.eu-crossgrid.org/wp5-1-login/CGTutorial.htm> (password-protected).

### 2.3. ADVANCED FEATURES

- The default behaviour of MARMOT can be overridden by setting special environment variables.
- NOTE: In previous releases (<= v1.1.11), these variables were not prefixed with "MARMOT\_", i.e. later releases use MARMOT\_DEBUG\_MODE instead of DEBUG\_MODE.
- In general, it depends on your system how to set environment variables, e.g. if you simply have to use a commandline like in bash
 

```
export MARMOT_DEBUG_MODE=1
```

 or in other shells
 

```
setenv MARMOT_DEBUG_MODE 1
```

 or if you have to edit a script like .bashrc or others.
- See also the USERS\_GUIDE file in MARMOT's distribution.
- MARMOT\_DEBUG\_MODE
 

Set the environmental variable MARMOT\_DEBUG\_MODE (integer) for the debug mode:

  - MARMOT\_DEBUG\_MODE < 0: nothing is reported.
  - MARMOT\_DEBUG\_MODE = 0: only errors
  - MARMOT\_DEBUG\_MODE = 1: errors and warnings
  - MARMOT\_DEBUG\_MODE = 2: errors, warnings and remarks are reported.

The default value is 2.

- **MARMOT\_INTERFACE\_MODE**

NOTE: THE FOLLOWING HOLDS TRUE FOR OLD VERSIONS OF MARMOT: MEANWHILE, MARMOT\_INTERFACE\_MODE IS SET AUTOMATICALLY TO THE CORRECT VALUE.

Set the environmental variable MARMOT\_INTERFACE\_MODE for the interface mode (default 0). The values mean

- 0: C Interface
- 1: Fortran interface

It does not do any harm to run a Fortran program with C INTERFACE\_MODE=0, in this case, MARMOT just may issue incorrect warnings like "ERROR: datatype is Fortran-Type" if one uses e.g. MPI\_INTEGER (which is correct in a Fortran program but not in a C program).

- **MARMOT\_SERIALIZE**

Set the environmental variable MARMOT\_SERIALIZE for serializing the code or not.

- MARMOT\_SERIALIZE = 0: code is not serialized
- MARMOT\_SERIALIZE = 1: code is serialized

default 0

- **MARMOT\_MAX\_TIMEOUT\_DEADLOCK**

Set the environmental variable MARMOT\_MAX\_TIMEOUT\_DEADLOCK for the maximum time all calls are allowed to stay pending before a deadlock warning is issued, i.e. set a value in microseconds for MARMOT\_MAX\_TIMEOUT\_DEADLOCK (default 1000000 microseconds = 1s).

- **MARMOT\_MAX\_TIMEOUT\_SERIALIZE**

Set the environmental variable MARMOT\_MAX\_TIMEOUT\_SERIALIZE for the maximum message time in case of serialization (MARMOT\_SERIALIZE=1), i.e. set a value in microseconds for MARMOT\_MAX\_TIMEOUT\_SERIALIZE (default 1000 microseconds).

- **MARMOT\_TRACE\_CALLS**

Set the environmental variable MARMOT\_TRACE\_CALLS, whether calls shall be traced back or not (default 1). The values mean

- 1: calls are traced with output to stderr, traceback in case of a deadlock is possible
- 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.

The number of calls to be traced back in case of deadlock can be set via MARMOT\_MAX\_PEND\_COUNT.

- **MARMOT\_MAX\_PEND\_COUNT**

Set the environmental variable MARMOT\_MAX\_PEND\_COUNT for the maximum number of MPI calls that can be traced back, i.e. set an int value for MARMOT\_MAX\_PEND\_COUNT (default 10).

## 2.4. KNOWN PROBLEMS

- When relinking the application with MARMOT, care has to be taken that all required libraries are linked properly and in correct order. The order is important. If you get an error message such as “undefined references”, you will have to add the library defining the missing thing. You may get thousands of “undefined references”, which can be resolved by adding only one single library, or you may have to add some more libraries. This depends very much on the compilers, platforms, linkers, etc.
- The usual problems may occur when submitting jobs to the testbed, for example problems with the Resource Broker or Migrating Desktop. MARMOT itself is quite stable.

---

### 3. INTERFACE REFERENCE GUIDE

Currently MARMOT does not have a GUI of its own.

## 4. TROUBLESHOOTING Q&A

### 4.1. HOW DOES MARMOT WORK?

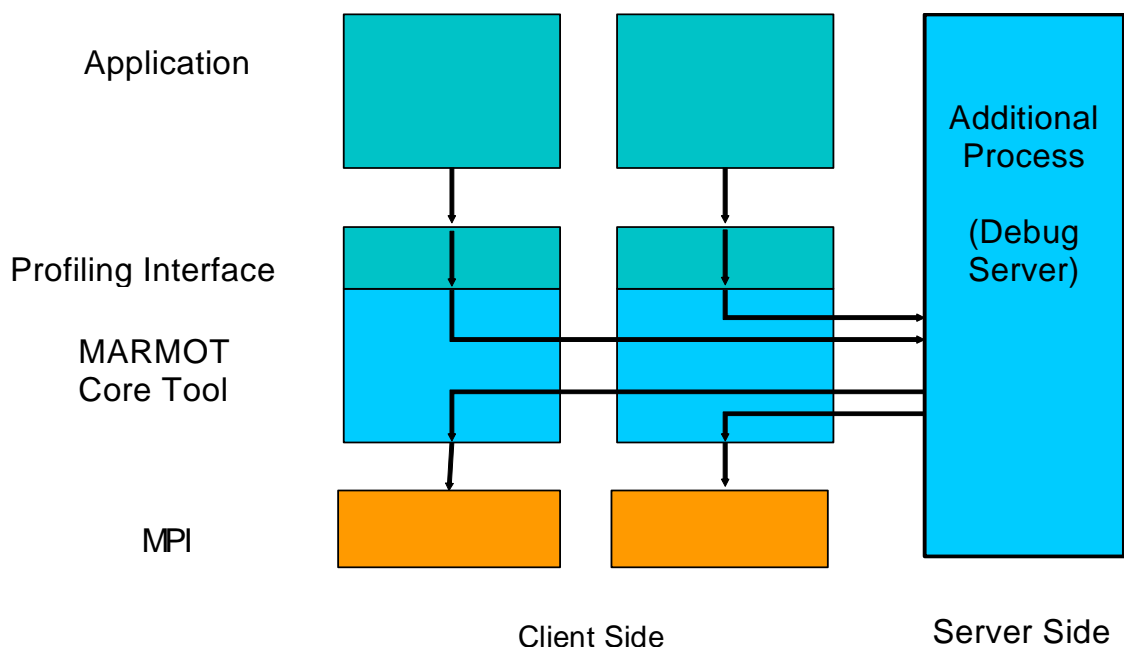
MARMOT does not change the MPI library or the application. We use the profiling interface that is defined in the MPI standard 1.2, but probably not all users know this profiling interface or have ever used it. The idea of the profiling interface is simple: any MPI call `MPI_Foo` can also be invoked by `PMPI_Foo`, they are exactly the same. This allows users to define their own MPI routines, and that's also what MARMOT does: to put it in a simple way, we redefine `MPI_Foo` by

```
{
  do the MARMOT checks;
  PMPI_Foo;
}
```

Thus we check the MPI calls made by the application and pass them to the underlying MPI library. Marmot is just an addition to the MPI library, not a replacement.

MARMOT also has its own book-keeping of resources such as datatypes, groups, communicators, etc. This enables the tool to check if these resources are used in a correct way, for example if they are constructed, used and freed properly.

MARMOT requires an additional process (our debug server), which is responsible for tasks that require a global view, e.g. deadlock detection. Local tasks such as the verification of resources are performed on the client side. The additional process is caught automatically by the tool, the user only has to run the application with one additional process. The figure below shows a graphical representation of the way MARMOT works.



## 4.2. WHAT ERRORS CAN MARMOT DETECT?

Currently there is no complete manual for all the warnings etc. MARMOT may find (we have 129 MPI calls and thus a lot of warnings.), but it is under construction. A very brief (however not yet complete) overview of MARMOT's error checks is given by this enum we use internally for MARMOT's implementation:

```
enum MARMOT_ERR_CODES
{
    MARMOT_SUCCESS = 0,          // successful return code
    MARMOT_ERR_UNDEFINED,       // undefined error
    MARMOT_ERR_MARMOT,          // implementation error in MARMOT

    // blocklength errors
    MARMOT_ERR_BLOCKLENGTH_NEG, // blocklength is negative

    // blocklength warnings
    MARMOT_WARN_BLOCKLENGTH_ZERO, // blocklength is 0

    // buffer errors
    // MARMOT_ERR_BUFFER,          // invalid buffer pointer
    // MARMOT_ERR_TRUNCATE,        // message truncated on receive
    MARMOT_ERR_BUFFER_ATTACHED,  // already a bufffer attached
    MARMOT_ERR_BUFFER_NOT_ATTACHED, //no buffer attached

    // color errors
    MARMOT_ERR_COLOR_NEG,       // color is negative

    // color warnings
    MARMOT_WARN_COLOR_UNDEFINED, // color is MPI_UNDEFINED

    // communicator errors
    MARMOT_ERR_COMM_NULL,       // communicator is MPI_COMM_NULL
    MARMOT_ERR_COMM_NOT_VALID,  // communicator is not valid
    MARMOT_ERR_COMM_NOT_CARTESIAN, // communicator is not cartesian
    MARMOT_ERR_COMM_NOT_GRAPH,  // communicator is not cartesian
    MARMOT_ERR_COMM_NOT_INTERCOMM, // communicator is not intercommunicator
    MARMOT_ERR_COMM_INTERCOMM,  // communicator is intercommunicator

    // communicator warnings
    MSG_WARN_COMM_NOT_WORLD,     // communicator is not MPI_COMM_WORLD
    MARMOT_WARN_COMM_NULL,       // communicator is MPI_COMM_NULL

    // coordinate errors
    MARMOT_ERR_COORD_NEG,       // coordinate is negative
    MARMOT_ERR_COORD_TOO_BIG,   // coordinate is out of range

    // count errors
    MARMOT_ERR_COUNT_NEG,       // count is negative
    MARMOT_ERR_COUNT_ZERO,     // count is 0
    // MARMOT_ERR_COUNT_ARRAY_NEG, //

    // count warnings
    MARMOT_WARN_COUNT_ZERO,     // count is 0
    MARMOT_WARN_COUNTS_NOT_EQUAL, // counts are not equal
    MARMOT_WARN_COUNT_UNDEFINED, // count is MPI_UNDEFINED

    // datatype errors
    MARMOT_ERR_TYPE_NULL,       // datatype is MPI_DATATYPE_NULL
    MARMOT_ERR_TYPE_NOT_VALID,  // datatype is not valid
    MARMOT_ERR_TYPE_NOT_COMMITTED, // datatype is created, not committed
}
```

```
MARMOT_ERR_TYPE_C, // datatype is C datatype
MARMOT_ERR_TYPE_F, // datatype is Fortran datatype
MARMOT_ERR_TYPE_REDUCTION_C, // datatype is for reduction function (C)
MARMOT_ERR_TYPE_REDUCTION_F, // datatype is for reduction function (F)
MARMOT_ERR_TYPE_OPTIONAL_C, // datatype is optional (C)
MARMOT_ERR_TYPE_OPTIONAL_F, // datatype is optional (F)
MARMOT_ERR_TYPE_CONSTR_DERIVED, // datatype is for constructing
// derived datatypes
MARMOT_ERR_TYPE_LB, // datatype is MPI_LB
MARMOT_ERR_TYPE_UB, // datatype is MPI_UB

// datatype warnings
MARMOT_WARN_TYPE_REDUCTION_C, //datatype is for reduction function (C)
MARMOT_WARN_TYPE_REDUCTION_F, //datatype is for reduction function (F)
MARMOT_WARN_TYPE_OPTIONAL_C, // datatype is optional (C)
MARMOT_WARN_TYPE_OPTIONAL_F, // datatype is optional (F)

// datatype notes
MARMOT_NOTE_TYPE_NULL, // datatype is MPI_TYPE_NULL
MARMOT_NOTE_TYPE_ALREADY_COMMITTED_C, // datatype is already
// committed (C)
MARMOT_NOTE_TYPE_ALREADY_COMMITTED_C_AND_F, // datatype is already
// committed (C and F)
MARMOT_NOTE_TYPE_ALREADY_COMMITTED_F, // datatype is already
// committed (F)

// dimension errors
MARMOT_ERR_DIM_NEG, // dimension is negative
MARMOT_ERR_DIM_TOO_BIG, // dimension is too big
MARMOT_ERR_NODES_NEQ_PRODUCT_DIMS, // nodes are not a multiple of dims

// dimension warnings
MARMOT_WARN_DIM_ZERO, // dimension is zero

// displacement errors
MARMOT_ERR_DISPLACEMENT_OVERRIDES_DATA, // displacement overrides data

// displacement notes
MARMOT_NOTE_DISPLACEMENT_READS_DATA, // displacement reads data /
// holes read from other nodes
MARMOT_NOTE_DISPLACEMENT_OVERRIDES_DATA, // displacement overrides data
MARMOT_NOTE_DISPLACEMENT_DATA_SENT_TWICE, // displacement sends data
// twice

// edge errors
MARMOT_ERR_EDGE_INVALID, // edge is invalid
MARMOT_ERR_EDGE_NEG, // edge is negative

// errhandler errors
MARMOT_ERR_ERRHANDLER_NOT_VALID, // errhandler is not valid

// errorcode errors
MARMOT_ERR_ERRORCODE_NOT_VALID, // errorcode is not valid

// group errors
// MARMOT_ERR_GROUP_NULL, // group is MPI_GROUP_NULL
MARMOT_ERR_GROUP_NOT_VALID, // group is not valid

// group warnings
MARMOT_WARN_GROUP_NULL, // group is MPI_GROUP_NULL

// index errors
MARMOT_ERR_INDEX_NEG, // index is negative
```

```
// index warnings
MARMOT_WARN_INDEX_LT_SIZE, // index is less than size of communicator

// leader errors
// Should we also distinguish between errors for remote
// and for local leaders?
MARMOT_ERR_LEADER_ANY_SOURCE, // leader is MPI_ANY_SOURCE
MARMOT_ERR_LEADER_NEG, // leader is negative
MARMOT_ERR_LEADER_OUT_OF_COMM, // leader is out of communicator

// neighbor errors
MARMOT_ERR_NEIGHBORS_NEG, // number of neighbors is negative

// operator errors
MARMOT_ERR_OP_NULL, // operator is MPO_OP_NULL
MARMOT_ERR_OP_NOT_VALID, // operator is not valid

// pending messages warnings
MARMOT_WARN_PENDING_MESSAGES_LEFT, // pending messages are left

// range errors
MARMOT_ERR_RANK_FIRST_NEG, // first rank is negative
MARMOT_ERR_RANK_LAST_NEG, // last rank is negative
MARMOT_ERR_RANK_FIRST_TOO_BIG, // first rank is too big
MARMOT_ERR_RANK_LAST_TOO_BIG, // last rank is too big
MARMOT_ERR_RANK_COMPUTED_NEG, // computed rank is negative
MARMOT_ERR_RANK_COMPUTED_TOO_BIG, // computed rank is too big
MARMOT_ERR_RANK_COMPUTED_TWICE, // rank is computed twice
MARMOT_ERR_STRIDE_NEG, // stride is negative
MARMOT_ERR_STRIDE_ZERO, // stride is zero
MARMOT_ERR_STRIDE_POS, // stride is positive

// rank errors (see also range errors)
// MARMOT_ERR_RANK, // invalid rank
MARMOT_ERR_RANK_NEG, // rank is negativ
MARMOT_ERR_RANK_PROC_NULL, // rank is MPI_PROC_NULL
MARMOT_ERR_RANK_ANY_SOURCE, // rank is MPI_ANY_SOURCE
MARMOT_ERR_RANK_TOO_BIG, // rank is too big
// MARMOT_ERR_DUP_RANK,
// MARMOT_ERR_RANK_ARRAY,
// MARMOT_ERR_LOCAL_RANK,
// MARMOT_ERR_REMOTE_RANK,

// rank warnings
MARMOT_WARN_RANK_ANY_SOURCE, // rank is MPI_ANY_SOURCE

// rank notes
MARMOT_NOTE_RANK_PROC_NULL, // rank is MPI_PROC_NULL

// request errors
MARMOT_ERR_REQUEST_COUNT_NEG, // count of requests is negative
MARMOT_ERR_REQUEST_NOT_VALID, // request is not valid
MARMOT_ERR_REQUEST_STILL_USED, // request is still in use
MARMOT_ERR_REQUEST_NOT_PERSISTENT, // request is not persistent
MARMOT_ERR_REQUEST_OUTCOUNT_UNDEFINED, // outcount is MPI_UNDEFINED

// request warnings
MARMOT_WARN_REQUEST_COUNT_ZERO, // count of requests is zero
MARMOT_WARN_REQUEST_NULL, // request is MPI_REQUEST_NULL
MARMOT_WARN_REQUEST_ACTIVE_NOT_PERSISTENT, // request is not persistent
// and active
MARMOT_WARN_REQUEST_ACTIVE_PERSISTENT, // request is persistent and
```

```

// still active
MARMOT_WARN_REQUEST_INACTIVE_PERSISTENT, // request is persistent
// and inactive
MARMOT_WARN_REQUEST_OUTCOUNT_UNDEFINED, // outcount is MPI_UNDEFINED
MARMOT_WARN_REQUESTS_LEFT, // requests are left

// size errors
MARMOT_ERR_SIZE_NEG, // size is negative

// size warnings
MARMOT_WARN_SIZE_ZERO, // size is zero

// tag errors
MARMOT_ERR_TAG_NEG, // tag is negative
MARMOT_ERR_TAG_TOO_BIG, // tag is too big ( > TAG_UB)
MARMOT_ERR_TAG_ANY_TAG, // tag is MPI_ANY_TAG

// tag warnings
MARMOT_WARN_TAG_TOO_BIG, // tag is too big ( > guaranteed range,
// but < TAG_UB)

// topology errors (see also dimension errors, coordinates errors)
MARMOT_ERR_TOPOLOGY_NEW_GT_COMM_OLD, // new topology greater than
// old communicator
MARMOT_ERR_TOPOLOGY_CART_NEW_GT_COMM_OLD, // new cartesian topology
// greater than
// old communicator

// topology warnings (see also dimension errors, coordinates errors)
MARMOT_WARN_TOPOLOGY_NEW_LT_COMM_OLD, // new topology smaller than
// old communicator
MARMOT_WARN_TOPOLOGY_CART_NEW_LT_COMM_OLD, // new cartesian topology
// smaller than
// old communicator

// misc
// MARMOT_ERR_PENDING, // Pending request

MARMOT_LAST_ERR_CODE
}; // end enum MARMOT_ERR_CODES
```

### 4.3. HOW CAN USERS MODIFY MARMOT?

todo

### 4.4. WHERE CAN I FIND MORE INFORMATION ON MARMOT?

MARMOT's homepage can be found at

<http://www.hlrs.de/organization/tsc/projects/marmot/>

A list of publications concerning MARMOT can be found at

<http://www.hlrs.de/organization/tsc/projects/marmot/pubs.html>

## 5. CONTACT INFORMATION AND CREDITS

If you have any questions, suggestions or bug reports, please contact the developers

Bettina Krammer, [krammer@hls.de](mailto:krammer@hls.de), Matthias Müller, [mueller@hls.de](mailto:mueller@hls.de)

HLRS

Allmandring 30

D-70565 Stuttgart

Phone: ++49 711 685 8038

Fax: ++49 711 678 7626

---

## 6. THE GPL LICENSE AGREEMENT

### GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

---

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for non-commercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

---

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.