Marmot – MPI Correctness Checker Exercise

VI-HPS Tuning Workshop
Wed, March 5, 9:00 – Fri, March 7, 12:30

Bettina Krammer, Valentin Himmler
High Performance Computing Center Stuttgart (HLRS)
Introduction – Building and installing Marmot

Marmot can be built and installed using either
• Autotools (autoconf, automake, …) or
• CMake
Introduction – Building and installing Marmot

Building with the autotools

Building Marmot with the autotools is straightforward. The command triplet

`./configure`  
`make`  
`make install`

is sufficient. However, Marmot needs at least to know where to find the MPI libraries and headers, so you usually have to call configure with at least one option:

`--with-mpi-dir=MPI_DIR`
Introduction – Building and installing Marmot

Building with CMake – Screenshot of the CMake ncurses interface
Introduction – Marmot at the RWTH Compute Cluster

Marmot is installed on the RWTH cluster. You can load Marmot with the following command:

```
module load VIHPS marmot
```

Now you should be able to use the Marmot libraries and binaries. Let's do some examples…
Working with Marmot

Copying the examples

At first you may create a Marmot directory and copy some examples:

```bash
mkdir $HOME/marmot
cp -r /rwthfs/rz/SW/UTIL/icc-10.1/intel_mpi/marmot/marmot_intel_mpi_64/examples $HOME/marmot
```

Let’s see what we got:
```
cd $HOME/marmot
ls
```
Working with Marmot – Examples 1

basic.c / basic.f

This is a very primitive program that just calls _MPI_Init_ and _MPI_Finalize_. We can now compile two versions of the program: One with Marmot and one without:

```
mpicc –o basic basic.c
marmotcc –o basic_m basic.c
```

If we now execute the version without Marmot…

```
mpiexec –np 2 ./basic
```

… nothing happens, as expected.
Now if we execute the Marmot version of the program (remember we have to add one process for Marmot’s debug server):

```
mpiexec -np 3 ./basic_m
```

We find the file `MarmotLog.txt` in the working directory. Let’s have a look at it. In this case, Marmot did not detect any errors (the program obviously has none). Now that was not too exciting, so we’ll step to the next example.
Working with Marmot – Examples 1

deadlock.c / deadlock.f

As the name suggests, this program deadlocks.
We compile the program and execute the non-Marmot version with two processes. The program deadlocks and we have to abort it (with CTRL+C or kill).

Now we execute the Marmot version with three processes. Naturally the program still deadlocks, but we get the warning:

**WARNING: all clients are pending! (Details see the LogFile)**

We still have to kill the processes manually, but now we have a log file.
Working with Marmot – Examples 1

deadlock.c / deadlock.f

The log file lists the MPI calls that lead to the deadlock. Here, both processes issue an *MPI_Recv* call, causing them to wait for each other.

```c
8: Error global message with Text: WARNING: all clients are pending!
Last calls (max. 10) on node 0:
  timestamp 2: MPI_Init(*argc, ***argv)
deadlock.c line: 51
  timestamp 4: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)
deadlock.c line: 52
  timestamp 6: MPI_Comm_size(comm = MPI_COMM_WORLD, *size)
deadlock.c line: 53
  timestamp 8: MPI_Recv(*buf, count = 1, datatype = MPI_INT, source = 1, tag = 17, comm = MPI_COMM_WORLD, *status)
deadlock.c line: 64

Last calls (max. 10) on node 1:
  timestamp 1: MPI_Init(*argc, ***argv)
deadlock.c line: 51
  timestamp 3: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)
deadlock.c line: 52
  timestamp 5: MPI_Comm_size(comm = MPI_COMM_WORLD, *size)
deadlock.c line: 53
  timestamp 7: MPI_Recv(*buf, count = 1, datatype = MPI_INT, source = 0, tag = 18, comm = MPI_COMM_WORLD, *status)
deadlock.c line: 70
```
Working with Marmot – Examples 1

request-reuse.c

In this example we reuse a request we didn’t free before.

Excerpt from the log file:

10: Error from rank 0(Thread: 0) with Text: ERROR: MPI_Irecv Request is still in use !

Argument: request

Information for Resource of type MPI_Request:
created at request-reuse.c line: 54
not yet freed.
Working with Marmot – Examples 1

pending-msg.c

Here we call MPI_Finalize when there is still a non-received message pending.

Excerpt from the log file:

11: Warning from rank 0(Thread: 0) with Text: WARNING; MPI_Finalize: There are still pending messages!
Working with Marmot – Examples 1

cg-tutorial-marmot-exercise.c

This program leads to a lot of different warnings and errors. You will notice that the log file has grown very big and a lot of messages appear several times (e.g. if the corresponding MPI call is inside a loop).

To cope with that, we can switch to other output formats and prevent multiple logging of the same error. These and other settings can be altered by means of environment variables.
Working with Marmot – Environment variables

MARMOT_DEBUG_MODE
controls which types of events are being reported.

Possible values for MARMOT_DEBUG_MODE:
• 0: Errors
• 1: Errors and warnings
• 2: Errors, warnings and remarks (default)
Working with Marmot – Environment variables

MARMOT_LOGFILE_TYPE
sets the format of Marmot’s log file.

Possible values for MARMOT_LOGFILE_TYPE
• 0: ASCII logging (default)
• 1: HTML logging
• 2: Cube logging
Working with Marmot – Environment variables

**MARMOT_LOG_FILTER_COUNT**

sets the maximum number of messages of the same type that are logged for each source location

Possible values for MARMOT_LOG_FILTER_COUNT

- INT (default: 50)
Working with Marmot – Environment variables

MARMOT_TRACE_CALLS
controls whether a traceback in case of a deadlock is possible

Possible values for MARMOT_TRACE_CALLS

- 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 (default)
- -1: calls are not traced, traceback in case of a deadlock is NOT possible
Working with Marmot – Environment variables

MARMOT_MAX_PEND_COUNT
sets the number of calls to be traced back in case of a deadlock

Possible values for MARMOT_MAX_PEND_COUNT
• INT (default 10)
Working with Marmot – Examples 2

deadlock.c / deadlock.f

We’re looking at the deadlock program once again, but now we want HTML output.

export MARMOT_LOGFILE_TYPE=1
mpiexec –np 3 ./deadlock_m

Next slide: Browser view (MarmotLog.html)
### Working with Marmot – Examples 2

<table>
<thead>
<tr>
<th>Global</th>
<th>Process ID</th>
<th>Event</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>Warning</td>
<td>Text: Debug server runs on same node as process 0 (zam431)</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Warning</td>
<td>Text: Debug server runs on same node as process 1 (zam431)</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Warning</td>
<td>Text: Processes 0 and 1 both run on zam431</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Global</th>
<th>Process ID</th>
<th>Event</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0</td>
<td>Error</td>
<td>Text: WARNING: all clients are pending!</td>
</tr>
</tbody>
</table>

Last calls (max. 10) on node 0:
- Timestamp 1: MPI_Init(argc, **arg**)
- Deadlock.c line: 51
- Timestamp 3: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)
- Deadlock.c line: 52
- Timestamp 5: MPI_Comm_size(comm = MPI_COMM_WORLD, **size**)
- Deadlock.c line: 53
- Timestamp 7: MPI_Recv(buf, count = 1, datatype = MPI_INT, source = 1, tag = 17, comm = MPI_COMM_WORLD, **status**)
- Deadlock.c line: 64

Last calls (max. 10) on node 1:
- Timestamp 2: MPI_Init(argc, **arg**)
- Deadlock.c line: 51
- Timestamp 4: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)
- Deadlock.c line: 52
- Timestamp 6: MPI_Comm_size(comm = MPI_COMM_WORLD, **size**)
- Deadlock.c line: 53
- Timestamp 8: MPI_Recv(buf, count = 1, datatype = MPI_INT, source = 0, tag = 18, comm = MPI_COMM_WORLD, **status**)
- Deadlock.c line: 70

---

Bettina Krammer, Valentin Himmler

Marmot – MPI Correctness Checker

VI HPS Tunig Workshop, March 5-7 2008, Aachen
Working with Marmot – Examples 2

cg-tutorial-marmot-exercise.c

Let’s say we’re only interested in severe problems (i.e. errors) and we don’t want the same error to be reported more than once

```bash
export MARMOT_DEBUG_MODE=0
export MARMOT_LOG_FILTER_COUNT=1
mpiexec –np 7 ./cg-tutorial-marmot-exercise_m
```

Next slide: Browser view (MarmotLog.html)
## Working with Marmot – Examples 2

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Code</th>
<th>Error Message</th>
<th>Source File</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>36 0 0</td>
<td>Error</td>
<td>This message will not be repeated on this process as it has exceeded the MARMOT_LOG_FILTER_COUNT limit.</td>
<td>cg-tutorial-marmot-exercise.c</td>
<td>228</td>
</tr>
<tr>
<td>36 1 0</td>
<td>Error</td>
<td>This message will not be repeated on this process as it has exceeded the MARMOT_LOG_FILTER_COUNT limit.</td>
<td>cg-tutorial-marmot-exercise.c</td>
<td>228</td>
</tr>
<tr>
<td>36 2 0</td>
<td>Error</td>
<td>This message will not be repeated on this process as it has exceeded the MARMOT_LOG_FILTER_COUNT limit.</td>
<td>cg-tutorial-marmot-exercise.c</td>
<td>228</td>
</tr>
<tr>
<td>36 3 0</td>
<td>Error</td>
<td>This message will not be repeated on this process as it has exceeded the MARMOT_LOG_FILTER_COUNT limit.</td>
<td>cg-tutorial-marmot-exercise.c</td>
<td>228</td>
</tr>
</tbody>
</table>
Working with Marmot

You can play around with your own applications, of course.

Thank you