Design and Implementation of an Eclipse plugin for MARMOT

Peng Deng

September 23, 2009

1 Overview

MARMOT[3] is a tool for analyzing and checking MPI programs. It surveys the MPI-calls made and automatically checks the correct usage of these calls and their arguments during runtime. It does not replace classical debuggers, but can be used in addition to them[1].

As one way to integrate the usage of MARMOT into normal MPI development work flow, we have been creating an plugin for a common IDE: Eclipse, which provides the functionality to configure/launch an executable and track the logging information within Eclipse interface. Moreover, with the Performance Tools Framework (PTFw) [5] provided by Parallel Tools Platform (PTP)[2], it is possible to further integrate the MARMOT compilation phase right inside a Eclipse project.

This document describes the design and implementation details of the plugin, as well as experience we received from some experimental steps we have taken to utilize PTFw. We will cover the issues we encountered during the integration and present some considerations for the development in future.

2 Standalone Eclipse Plugin

Generally, the standalone eclipse plugin, or "Marmoclipse" is developed without PTP in mind and tries to fulfill basic MARMOT use case inside Eclipse's user interface. The user can launch an executable file pre-compiled with MARMOT compiler and then the log information will be obtained by listening a certain TCP/IP port and tracking the data received. Some important parameters for the execution are configurable and the log information can be saved.

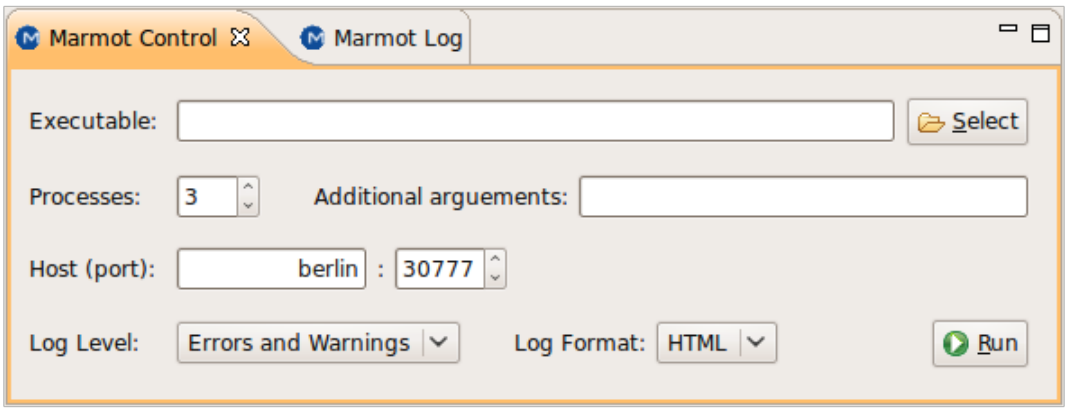

Figure 1: The user interface of Marmot Control View

2.1 User Interface

Marmoclipse provides 2 *views*, namely **Marmot Log View** and **Marmot Control View**, and 1 *perspective* — **Marmoclipse Perspective**. The functionalities of these three components are as following:

- **Marmot Control View** offers a configuration interface for the user to choose the executable to run and modify several MARMOT related runtime options. See figure 1.
- **Marmot Log View**, as shown in figure 2 on the next page displays the log information received by listening to the network port chosen by the user in the control view. If provides basic filters to hide/show different levels of log entries. It also let the user save the log entries into a file.
- **Marmotclipse Perspective** is meant to organize views and other UI components related to the plugin and MARMOT. Currently it simply arranges the 2 views mentioned above to the lower right area of the workbench.

2.2 Implementation

2.2.1 Packages and Source Code Organization

The code of the plugin is organized using 5 packages:

- `de.hlrs.marmoclipse` is a base package that holds a essential class for the plugin.

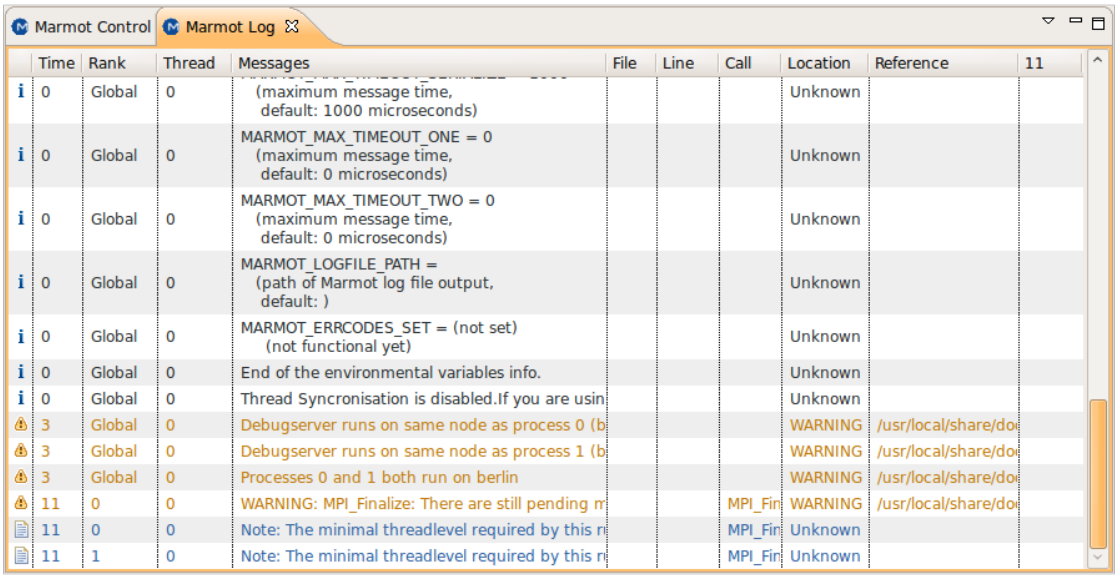

Figure 2: The Marmot Log View, displaying sample log entries.

- `de.hlrs.marmoclipse.log` contains MARMOT log model and a socket listener implementation to trace the log information.
- `de.hlrs.marmoclipse.log.ui` includes classes that create the table viewer which is used by the Marmot Log View.
- `de.hlrs.marmoclipse.views` and `de.hlrs.marmoclipse.perspective` hold the implementations of the 2 views and 1 perspective, respectively.

2.2.2 Important Classes

- `MarmotLog`

  This class models a MARMOT log entry, which contains 11 different fields. Object of this class is constructed from a received line of log which contains all the fields truncated with a special character (Unicode: `0xfffd`) as separator.

- `MarmotLogList`

  It models a list of `MarmotLog` objects and provides several helper methods for adding, removing log entry to the list and clearing the list. This class also contains a simple implementation of change listeners which will be used by `LogContentProvider` class.

- `LogSocketListener`

This class implements a socket listener so it is possible to receive the log information via a specific port. It extends class `Job` so that we can schedule the running of the listener into background and avoid it blocking the user interface.

- Table Viewer related classes

  Package `de.hlrs.marmoclipse.log.ui` provides a couple of classes which will support the use of `TableViewer` in `MarmotLogView`. `TableViewer` is a powerful yet flexible solution to integrate different UI components into a table representation, and it allows to display a domain model in a list, tree or table without converting the domain model beforehand.

  For our design, we need:

  - `LogContentProvider` to provide the log model to the table viewer;
  - `LogLabelProvider` to define how the data is displayed in the table viewer cell;
  - `LogEdtingSupport` to make the link displayed in the "Reference" column clickable (This in turn uses a customized cell editor class `ReferenceCellEditor`);
  - `LogViewerFilter` to support basic filtering functionality.

  The implementation of these classes are more or less following the routines, please refer to [4] for example and further information.

- `MarmotControlView`

  This class creates the user interface of the **Marmot Control View** by implementing all the UI components and their corresponding method. There is an in-line class `RunSelectionListener` triggered by the "Run" button when clicked. It does some sanity checks and assembly the command line from all the options chosen by the user and starts/schedules the socket listener before switching to the **Marmot Log View**.

- `MarmotLogView`

  This class creates the table viewer for displaying the log entries and several actions for filtering, clearing and saving the log information.

2.2.3 Notes

- Although it is possible to pre-define the path to save a log file in the execution command line, in our design we only let the user choose the path/file when the user decide to save a log. Therefore the log file is redirected into system's temporary folder when launching the executable, and moved to the destination when chosen to be saved or removed when discarded.

2.3 Test

Since Marmoclipse is not packaged as a production-ready plugin, following steps need to be follow to test it:

1. Check out the source code from MARMOT's SVN repository
2. Use Eclipse to open the "marmoclipse" project located at `SRC/TOOLS/marmoclipse/`. Note: "Plugin Development Environment (PDE)" should be installed with Eclipse.
3. Right-click at `plugin.xml` file in the "*Package Explorer*" on the left hand side and choose in the context menu "*Run as → Eclipse Application*";
4. A new eclipse instance should start. If the current perspective is not Marmoclipse, switch to Marmoclipse in the perspective menu ("*Window → Open perspective → Other ...*")
5. Now it is ready to test with any MPI binary executable pre-compiled using MARMOT's compiler.

3 Performance Tools Framework Integration

The **Parallel Tools Platform (PTP)** is a plugin for Eclipse which provides a highly integrated environment specifically designed for parallel application development. Since MARMOT is a tool for verification and analysis, it is natural and reasonable to utilize PTP as bridge to offer the MARMOT features as part of the parallel development environment.

The **Performance Tools Framework** (PTFw, a.k.a PTP External Tools Framework) is part of PTP project and aims to ease the integration of existing external tools in Eclipse. We have done some experiments to integrate MARMOT with Eclipse/PTP via this framework, however because it is still in an early stage in its development, we encountered some issues due to the lack of certain features. We will explain in this section some steps we have taken and discuss the possible solutions to work around the issue.

3.1 Integration Approach and Test

For PTFw, an XML file which specifies the steps to take for the *compilation*, *execution* and *analysis* phases in a work flow plays the most important role. By following certain syntax, one can define in the XML file the command and arguments needed. Moreover, UI components can be generated through the definitions which makes it possible to assign

values to arguments through user's input. For the analysis phase, an Eclipse plugin can also be used to handle (e.g. visualize) the produced data.

One of the advantages of using PTFw is we can easily include MARMOT compilation of the source code. This has to be done externally and manually with the standalone plugin mentioned in section 2 on page 1.

We have created an experimental version of such a XML file and included inside the project folder. It specifies the compile step and options needed to be given by the user for the execution step. You can add the XML file directly through PTP's preferences dialog (*Preferences* → *PTP* → *Performance Tools*), then right-click an MPI project created inside PTP and choose *Profile as* → *Profile Configurations ...*. You can create a new "Performance Analysis" profile in the opened dialog. At the *Performance Analysis* tab, "Marmoclipse" should be available from the *Select Tool* dropdown list. and a "Marmoclipse" sub-tab should be visible and selectable, where values for various arguments are supposed to be given.

3.2 Existing Problems

3.2.1 Socket Establishment

If performing a profiling/analysis, the source file can be seen compiled by marmotcc but then an error message like "`Marmot: Could not connect to localhost:30777`" will appear. This is because while running the MPI executable, we don't have a socket listener ready to establish a proper channel to receive the log messages. So here we need to take care of two things:

- We need to start a socket listener prior to the execution of the MPI binary
- We need to relay the network host and port arguments obtained from user's input to the socket listener.

And accordingly, we could have the following possibilities to work on solving the problem:

1. Using a script instead of `mpiexec` for the execution.

   This script will take all the parameters and pass the *host* and *port* values to an external network listening tool (e.g. netcat) and then start mpiexec (with all parameters). This is a quick fix but obviously we sacrifice the compatibility between different platforms since the scripting format and availability of network tool cannot be guaranteed, not to mention that we need to ship an extra script with the plugin. Also if we choose to use external tool to listen and receive the log information, we

cannot display the log simultaneously (as in the standalone plugin). Instead we have to parse the log messages dumped by the external tool.

2. After some hints provided by the author of this framework via the mailing list, some experiments show that it is possible to add a *analyze* step before *execute* in the XML file to launch some command. But during the experiments, it seemed not all commands can be effective.

   Nevertheless, this indicates that analysis can be triggered before execution therefore we may be able to activate our own plugin code before execution starts. This way we might be able to establish a socket listener like we do with the standalone plugin. The only problem here is how to pass the host and port information from the profile configuration page to the plugin code. It needs a better understanding and further investigation of the `org.eclipse.ptp.perf.dataManagers` extension point offered by PTFw.

   This approach is a recommended working direction as the existing code can be re-used and no extra external tool is used thus cross-platform compatibility can remain.

3. Start the listener separatedly.

   We can also work-around the issue by providing a socket listener independent from the PTFw. Then the user has to manually start the listener prior to starting the profiling/analysis. This can be implemented as a UI component directly integrated into Eclipse's interface (e.g. a toggle button). The disadvantage of this approach is it becomes impossible to let the listener listen to the host/port the user chooses. The host/port have to be a hard-coded value as a consequence.

3.2.2 Limited UI component choices

Currently there is limited number of UI component that can be generated from the XML definition. As a result, some option values have to be input with plain text and data validity of them cannot be assured. According to the documentation of the framework, more components may be added in future releases. When proper component is available, some of the current components should be replaced. (e.g. combo menu for "Log Level" and "Log Format", number spinner for "Port" and "Processes"). Meanwhile, there is a extension point `org.eclipse.ptp.configurationTabs` which can be used to add SWT widget.

References

[1] Marmot. `http://www.hlrs.de/organization/av/amt/research/marmot/`.

[2] PTP - Parallel Tools Platform. `http://www.eclipse.org/ptp/`.

[3] *MARMOT - tool for analysing and checking MPI programs*, May 2009.

[4] Laurent Gauthier and Mirasol Op'nWorks. Building and delivering a table editor with SWT/JFace, July 2003. `http://www.eclipse.org/articles/Article-Table-viewer/table_viewer.html`.

[5] Wyatt Spear and Beth Tibbitts. PTP/ETFw/PTP external tools framework, September 2009. `http://wiki.eclipse.org/PTP/PTFw/PTFw-Overview`.