Real-Time Embedded Software Group

RiTHM: A Tool for Enabling Time-triggered Runtime Verification for C Programs

User's Guide

Electrical and Computer Engineering Department,
University of Waterloo, January 2013

Table of Contents

List of Tables

List of Figures

About This Guide

This guide is intended for software developers that wish to use RiTHM to verify their systems at runtime. RiTHM is a tool chain that automates the process of runtime verification of projects written in C by leveraging time-triggered monitoring to observe the system's runtime behavior, and the GPU many-core technology to verify the runtime behavior. The rest of the guide provides (1) an overview of RiTHM, (2) the installation procedure, (3) RiTHM's directory structure, (4) RiTHM's system requirements and limitations, and (5) how to use RiTHM along with an example.

CHAPTER 1

RiTHM Overview

In a computing system, correctness refers to the assertion that the system satisfies its specification at all times. Achieving system correctness is a major problem for todays large software systems. Verification and testing are arguably the two most common approaches to ensure program correctness. However, verification may suffer from the state explosion problem, and testing may not be able to cover all possible execution scenarios of the system. These limitations argue for *runtime verification* where it inspects a program's runtime behavior to verify a set of properties at run time. Runtime verification frameworks mainly consist of two major components, the *monitor* which extracts information from the program at run time, and the *verification engine* which verifies a set of properties at run time with respect to the information provided by the monitor.

Most monitoring approaches in runtime verification are *event-triggered*. In these approaches, the occurrence of an event of interest triggers the monitor to extract information and subsequently call the verification engine for property evaluation. This technique leads to defects such as unpredictable monitoring overhead and potentially bursts of monitoring invocation at run time. Such defects can cause unpredictable behavior at run time; especially in real-time embedded safety/mission-critical systems, where it can result in catastrophic consequences. To tackle these drawbacks, we propose **RiTHM**: Runtime Time-triggered Heterogeneous Monitoring.

RiTHM is a runtime verification framework which uses *time-triggered* monitoring to observe the system's runtime behavior. The time-triggered monitor runs in parallel with the program and extracts (i.e., samples) the program state at fixed time intervals (i.e., the sampling period) and subsequently call the verify to evaluate a set of LTL_3 properties with respect to the sampled program state. Our studies show that the time-triggered monitor of RiTHM results in observing bounded monitoring overhead and predictable monitoring invocation at run time, a feature required for runtime verification of real-time embedded safety/mission-critical systems.

1.1 Time-triggered Monitor

The time-triggered monitor is a separate thread which runs in parallel with the program and samples the program with respect to a pre-defined fixed sampling period. An issue surrounding time-triggered monitoring is *sound state reconstruction* (i.e., sampling of all states vital to the verification of the properties). Hence, RiTHM leverages static analysis of the program to determine the *longest sampling period* which ensures sound state reconstruction. To further decrease the runtime overhead of monitoring, RiTHM devises a technique to add minimal instrumentation into the program to further increase the sampling period of the time-triggered monitor while ensuring sound state reconstruction. RiTHM intends to add the instrumentation such that minimal additional memory is required.

1.2 Verification Engine

When a time-triggered monitor reads the state of a program, it passes them to the verification engine to evaluate a set of $\textsc{Ltl}_3$ properties. The verification engine evaluates the set of properties in a parallel fashion and makes use of the GPU many-core technology. The verification engine uses two parallel monitoring algorithms that effectively exploit the many-core platform available in the GPU. The verification engine further reduces monitoring overhead, monitoring interference, and power consumption due to leveraging the GPU technology.

CHAPTER 2

Getting Started

This chapter will guide you through the procedure to download RiTHM's source files and build the tool from source. At the moment, RiTHM is targeted for both 32- and 64-bit Ubuntu 12.04 LTS and 32-bit Ubuntu 11.10. Most of the build and execution infrastructure should be portable to any *nix system, but will require additional efforts to build RiTHM's dependencies. The support or other *nix distributions is under construction and will be available in the future.

2.1 Pre-requisites

The following is a list of things that you will require to build and develop RiTHM:

Operating System

- Any computer running Ubuntu 12.04 LTS or 32-bit Ubuntu 11.10.
- `sudo` access with your user account in Ubuntu.

get-apt Packages

- `ia32-libs`: If the user is running Ubuntu 64-bit, `i132-libs` package must be installed. This package allows the user to skip compilation processes specific to 64-bit machines and run 32-bit executables on 64-bit platforms.
- `realpath`: This package is used by some of the invocation scripts to resolve path parsing issues and path dependencies.

- `qmake`: If the user wishes to build the GUI from the committed source file.

OpenCL Packages

RiTHM allows the user to choose whether the verification engine performs its verification on CPU or on GPU. Thus, the OpenCL supporting packages must be perviously installed.

- Systems with AMD/ATI GPU: AMD GPU OpenCL SDK can be downloaded from here:

  `http://developer.amd.com/tools/hc/AMDAPPSDK/downloads/Pages/default.aspx`

- System with NVIDIA GPU: NVIDI OpenCL SDK can be downloaded from here:

  `http://www.nvidia.com/Download/index.aspx?lang=en-us`

Remark: If your system uses AMD GPU/APU card, but AMD GPU OpenCL SDK is not installed, it will be installed automatically when running ./build_deps.sh.

General Applications

- Text editor of your choice.
- An integrated development environment (IDE), if you are not as comfortable with the command line.

Optional Dependencies

If you intend to acquire RiTHM from Bitbucket (see Section 2.2), you will need the following:

- `git`: If `git` has not been installed on your machine, you can install it on Ubuntu by typing the following command in the terminal:

```
sudo apt-get install git
```

- SSH keys, if you wish to connect to Bitbucket via SSH
- Bitbucket account (this is optional; you can choose to download the source and work locally).

Other Dependencies

The following packages are packaged along with RiTHM's source code and will be installed automatically when running ./build_deps.sh:

- `lp_solve`.
- `libconfig`.
- `open CSV` (java lib).
- `apache commons` (java lib).
- `LLVM`.
- `clang`.

2.2 Installation

Step 1. Make sure you have installed all the pre-requisites for OpenCL Packages and General Applications.

Step 2. Acquire RiTHM. You can do so by using one of the following methods:

- Acquire it directly from the Real-time Embedded System's group webpage. Download the tar ball of the tool provided by the following link:

  https://uwaterloo.ca/embedded-software-group/projects/rithm

- Acquire it from Bitbucket. To work with the git repository on Bitbucket, you can either retrieve the tool sing HTTPS or SSH. . If you plan on using HTTPS to communicate with the Bitbucket repository, change to the directory that you would like to make the clone in and then enter the following command in the terminal:

  ```
  git clone
  https://<username>@bitbucket.org/embedded_software_group/rvtool.git
  [name of local directory]
  ```

  where `<username>` is your Bitbucket username, and an optional argument `[name of local directory]`, which designates the name of the folder the cloned repository will be in. The `git clone` command is slightly different for SSH:

  ```
  git clone git@bitbucket.org:embedded_software_group/rvtool.git
  [name of local directory]
  ```

Step 3. After cloning the repository to your local machine, it is time to start building the tool from source. First, change the directory in your terminal to the root repository directory. Here, there are two files you will need to invoke to build the tool:

- `build-deps.sh`
- `Makefile`

`build-deps.sh` contains the necessary commands required to pull in all of the tool's external dependencies and build them as necessary. This will also establish the subdirectory named 'build', which will contain all compiled objects and executables. From the terminal, run the following command:

```
sudo ./build-deps.sh
```

`sudo` access is required only to install missing packages using `apt-get`. The following packages may be installed using `apt-get`:

- `realpath`: converts any relative directory/file paths into absolute paths.
- `subversion`: version control required to pull LLVM and Clang source.
- `ia32-libs`: needed for machines running 64-bit Ubuntu for compatibility with other libraries.
- `AMD APP SDK`: (Note: this is installed in `/opt/AMDAPP/` as opposed to the 'build' directory).
- `Qt and QMake`.
- `lp_solve`.
- `libconfig`.
- `open CSV` (java lib).
- `apache commons` (java lib).
- `LLVM`.
- `clang`.

This script will likely take one or more hours to finish, because the LLVM framework and Clang takes a long time to compile and build for the first time.

Step 4. When the `build-deps.sh`, run `make` form the root repository directory in the terminal (no need in sudo) to build the tool. When `make` finishes running, you can change to the 'build' directory and run RiTHM by either using command-line (calling `run.sh`) or the GUI (initiating `rvtool`). For more details about running RiTHM, please refer to Section 4.

CHAPTER 3

Repository Structure

CHAPTER 4

Running RiTHM

RiTHM can be initiated via both from a GUI environment (recommended) and from command-line. The GUI environment allows the specification of all the required parameters to run RiTHM. Then the GUI wraps the parameters and invokes the appropriate shell script that, in turn, the script sequentially invokes the shell scripts with their corresponding command-line parameters.

4.1 Running RiTHM via GUI