MLRTEMS: Embedded Standard-ML

Nathan Burgers

May 3, 2013

1 Summary

Functional programming is often ignored in embedded and real-time systems because of its perceived runtime overhead. This project aims to provide a bridge between the MLton optimizing Standard-ML compiler, and the RTEMS real-time systems executive.

2 Motivation

C is currently the language of choice for embedded systems, and is the target language for RTEMS. Historically, C has proven itself to be insecure due to problems with dangling pointers, lack of true type safety, unbounded arrays, and ease of accidentally leaking memory. Embedded systems frequently demand high code stability and may be long-running processes where memory leaks are a serious issue.

In contrast, Standard-ML provides true type safety, immutable data types, mutable references, bounded arrays, and garbage collection, among other features. I argue that embedded systems stand to benefit from functional programming concepts. Many features of embedded systems map well to functional ideas such as stream processing and functional reactive programming. Standard-ML is the language of choice because it has extensive specification and several mature implementations that emit C as their target language.

3 Project Overview

MLton translates Standard-ML code to optimized C and RTEMS is provided as a series of C libraries that are conditionally compiled to meet the needs of the system. A basic RTEMS embedded system using Standard-ML requires bridging the two technologies using MLton's foreign function interface, and an associated toolchain to build MLton, RTEMS, and the embedded software product.

The primary focus of this project will be on tooling and will proceed in three phases:

1. Provide a build automation suite to make the process of contributing to the MLton and RTEMS projects simpler. RTEMS is notorious for being difficult to compile, and the barrier of entry for contributing to either of these projects is currently high.

2. Develop a set of benchmarks suitable for a real-time system alongside MLton's benchmark suite and run them against real hardware, locating problem areas. I have access to LEON3 development boards.

3. Target the areas of the MLton compiler that pose problems for real-time applications. It is expected that garbage collection will be the primary factor here. Methods of improving garbage collection speed can range from providing different functional data structure primitives, to altering the garbage collection scheme to reference counting or memory regions, etc.

4 Success Criteria

1. An automated build suite can update and compile the MLton and RTEMS sources for producing binaries on RTEMS's supported hardware. Ideally supporting all target architectures.

2. Get a suitable benchmark suite running on embedded hardware, and the benchmarks online.

3. A proof-of-concept solution for real-time Standard-ML. Most likely this will involve forking MLton's memory management scheme.

5 Roadmap

I would like to work out the roadmap for this project closely with a mentor given discussion about where to take the project once MLton can build its own benchmarks on RTEMS with real hardware.

Prior to GSoC Prepare the project build and test environment. Get a centralized Github repository up with the build tools in their current state. Work with mentors to gain a better understanding of MLton's build process and where its incompatibilities with RTEMS may lie. Read through the MLton's code emitting source.

First week Fork the MLton repository, and continue work scraping the RTEMS source tree to make a build generator. I intend to use Clojure to generate Ant build scripts that download, patch, and compile RTEMS for different architectures, primarily focusing on LEON3. The only dependency for the build tools will be Ant (amongst utilities like cURL and tar.) RTEMS requires special patched versions of some Autotools, GCC, GDB, and arithmetic libraries like MPFR. I intend to have the system set up a sandbox where these binaries don't interfere with a user's installed versions of GCC, etc.

Week 2 I expect to run into issues automating RTEMS compilation for different architectures, so I'm allotting some buffer time here.

Week 3 Provide bare-bones, possibly automated, bridge between RTEMS's IO libraries and MLton's FFI to run MLton's benchmarks. I expect RTEMS standard libraries will behave differently than how MLton expects, and this may take some time to get working.

Weeks 4 to 5 Get code compiled with MLton running under RTEMS on a virutal x86 environment such as QEMU. After this I want to run MLton's benchmarks on the system for a baseline and get those up on Github Pages.

Weeks 6 to 7 Get MLton successfully cross-compiling its benchmarks with RTEMS on real hardware. I currently have access to LEON3 development boards through my undergraduate studies at University at Buffalo. I expect MLton's will have some compatibility problems with RTEMS's C compiler since it may rely on non-standard features of the

GCC. MLton will need to be forked to solve these. I would want the benchmarks to be comprehensive to see exactly where the performance problems lie, and get these up online.

Week 8 Work with mentors outlining the direction the compiler should take to provide real-time guarantees, and select the most pressing issue if there are multiple. I expect memory management will be the biggest hurdle for real-time.

Weeks 8 to 9 Benchmark a reference implementation of our problem domain in straight C under RTEMS on the LEON3 board. C-based languages like Cyclone provide memory regions, and there are reference counting libraries for straight C. If the problem domain is memory management, then having a reference for the kinds of performance characteristics we are looking at will be useful.

Weeks 10 to 12 Get a rough implementation of the selected approach to the problem domain up. I don't expect to create a revolutionary solution for real-time functional programming completed in this time, but a working proof of concept is expected. This would preferably be a Standard-ML library, but may take the form of an extension to MLton if the need arises.

Week 13 Spend the remaining time doing code cleanup, polishing, and finalizing the web presence of the project. By this time I expect to have the benchmark suite for MLton, the custom benchmarks for RTEMS, and the build tools in a presentable state put up on Github Pages.

6 Involvement Beyond GSoC

I am currently working on the problem of real-time Standard-ML under professor Lukasz Ziarek at University at Buffalo, and have deep interest in functional programming and compilers. I intend to stay with this project for at least the next two years, and intend to get Standard-ML into a production-ready state for soft real-time robotics. To get a working system the project will need to provide a strict scheduling system with appropriate threading and continuation primitives, alongside a garbage collector or other memory-management scheme that does not interfere with process timing. We'll need to develop the proper abstractions between RTEMS's system primitives and Standard-ML. In the long term, adding real-time guarantees to Standard-ML may require a language extension, such as manual reference counts or region annotations.

7 Qualifications

My resume can be downloaded in pdf form from `resume.nateb.me`

- Currently performing undergraduate research with Standard-ML under professor Lukasz Ziarek at University at Buffalo.
- Over six years of general programming experience.
- Over one year of experience developing tools at Refulgent Software LLC.
- Currently working on an automated build suite for RTEMS by scraping its latest versions, dependencies, and tool patches from the RTEMS Git repository.