ALFPrinter: adding static WCET analysis to LLVM

Abstract:

For software that runs on an embedded processor that is to be used in a hard real-time system, it is important to know how long a particular task might run in the worst case. This is called the Worst Case Execution Time (WCET). There exist tools for computing the WCET that analyse the binary. They can compute the WCET without executing the code (static-analysis) using a complex mathematical model. It might be the case that not all possible paths in the code are executed when executing the code very often.
Adding new targets to these WCET tools is a bit of a problem. Typically, they have the decompilation and reconstruction of the CFG build-in. Adding a new target is then either under contract if the tool is commercial, or requires extensive knowledge of the tool if it is open-source. The WCET tool SWEET is a bit of an exception to this, as it has an interface language called ALF. ALF is developped in such a way that various Instruction Set Architectures (ISA) can be expressed in it.
Unfortunately, no compilers allow a mappings to ALF as of now. This is the problem that this project aims to solve, by adding an interface for outputting ALF code from the LLVM compiler. The main idea is to subclass MCStreamer (say we name it ALFPrinter) that allows transforming MCInsts from the MC layer to ALF code. The project will also include a proof-of-concept with a simple ISA.

1 Summary:

By adding an ALFPrinter interface in the MC layer, this project aims to make it easy for supporting new targets for the static analyisis tool SWEET. SWEET can compute the Worst Case Execution Time of your code by looking at the code statically using flow-analysis. This has applications for embedded processors that are to be applied in a hard real-time system.

2 The project:

A WCET tool is a tool that can determine the longest possible execution time of your code (Worst-Case Execution Time). This WCET is important in hard real-time applications

where multiple tasks are scheduled on the same (embedded) processor, where each task needs to finish by a specified deadline. Missing the deadline can have disastrous results (e.g. air-bag in a car does not open in time).

Static-analysis WCET tools decompile a binary and do analysis on the flow of the code (so they do not actually run the code). These tools typically manage the decompilation themselves, and adding a new target architecture is either not possible or is under contract and will be costly if the tool is commercial.

The open-source WCET static-analysis tool SWEET [1] has a novel approach to this problem. It has an interface language (called ALF [2]) that is designed to allow ISA code like AVR, ARM to be described in it. The idea is to make it easier to add a new target architecture, as you would only need to decompile and transform it to this ALF.

The goal of this project, is to add a new interface to LLVM in the MC layer (an MCStreamer), that allows printing to ALF just like printing is done to .o object files or .s assembler text. Thus making it very easy to decompile into MCInsts and output ALF. This would make it very easy to add WCET analysis to existing and upcoming architectures.

Apart from ALF code, SWEET also requires the cycle count of the basic blocks in order to determine a good WCET bound. This can either come from a cycle-accurate simulator or could be outputted from LLVM directly, if the cycle costs of the instructions are simple to determine. Perhaps there already exists data structures in LLVM that keep track of the cycle count of a particular basic block. This will have to be investigated.

The ALFBackend project [3], build upon LLVM 3.4, contains code to output to ALF which can possibly be re-used. ALFBackend implements an IR Pass, and allows outputting ALF code from IR code directly. This is different from what I aim to achieve. The issue is that the structure of IR code is a lot different from the code in the final binary due to optimizations in the back-end. The MC layer is supposed to closely resemble the output binary.

3 Benefits for LLVM:

- Add WCET analysis to LLVM by adding support for an external open-source tool.
- Saves implementing Abstract Interpretation and other complex WCET analysis stuff into LLVM.
- New targets would be added easily, the developer would only have to map MCInsts to ALF using ALFPrinter.
- Allows computing the WCET right away with compilation of your code.

4 Timeline:

Community bonding period:

Investigate what parts of ALFBackend can be re-used. Take a look at the MC layer and the MCStreamer API. Learn about the LLVM development model.

Week 1: Get more familiar with ALF & how memory is represented in ALF. Choose target test setup. Obtain a dev board with a simple embedded processor such as ARM cortex-M0/3, AVR. Preferably one that is supported by the major commercial WCET tool, aiT from AbsInt [4] to compare against this tool.
Week 2: Take another good look at the MC layer, and at an implementation of ASMPrinter.
Week 3: Subclass MCStreamer, reuse the classes from ALFBackend that model and print ALF.
Week 4: Attempt mapping some simple instructions (e.g. add, mul). See if I can get working output.
Week 5: Try to output the required basic-block costs from ALFPrinter. First hard-coded, then maybe from an existing datastructure in LLVM that allows me to derive cycle costs from a given basic-block.
Week 6: Map the other instructions to ALF. Possibly use the undefined ALF instruction if it is not possible.
Week 7: Buffer week for working on mapping.
Week 8: Buffer week for working on mapping.
Week 9: Attempt to execute some tests from the Mälardalen benchmark suite. Try to get a license for aiT and compare with this tool.
Week 10: Write documentation on how to add a new ALFPrinter target.
Week 11: Try to port ALFBackend to mainline, get all the tests working.
Week 12: Compare ALFBackend (IR->ALF), with ALFPrinter (MC->ALF), with aiT.

5 Deliverables:

- ALFPrinter interface in the MC layer
- ALFPrinter interface implemented for a simple target architecture

6 Nice to have:

- Test setup that compares ALFBackend WCET bounds and ALFPrinter WCET bounds
- Documentation on how to add a new target using ALFPrinter.
- Port ALFBackend to LLVM mainline

About me:

I'm an second-year Embedded Systems graduate student from the Eindhoven University of Technology in the Netherlands. I've been introduced to LLVM by the course "Parallelism, compilers and platforms" where it has been used extensively as a learning tool. I did a bachelor in computer science at the Avans University of Applied Sciences in Den Bosch in the Netherlands where I learned to program in a bunch of programming languages such as C, C++, Java, C#, Python.

In the last five to six weeks I have been looking into WCET tools (both commercial and open source) and into SWEET and ALF.

References:

[1] http://www.mrtc.mdh.se/projects/wcet/sweet/index.html
[2] http://www.mrtc.mdh.se/projects/all-times/documents/ALF/ALF-spec.pdf
[3] https://github.com/visq/ALF-llvm
[4] https://www.absint.com/ait/index.htm