

High-Level Optimizations for Low-Level Software

John Regehr — School of Computing, University of Utah

Embedded software needs to meet stringent requirements such as high reliability, minimal use of resources, and short development time. The problem is that these requirements are not only individually difficult, but are also in tension. We are working on VIPA (vertically integrated program analysis), a partial solution to this problem. VIPA is a new framework that supports high-level optimizations and analyses that are more aggressive and more domain specific than those traditionally performed by compilers. Its distinguishing features are:

- Integration of tools operating at multiple levels of abstraction: the model level, source code level, and binary level.
- Extraction of as much information as possible from each tool, as opposed to the common practice of using program analyses separately to produce binary results, e.g., “the network acceptor thread has the potential for stack overflow.”

For example, in previous work [1] we not only demonstrated how to show stack safety of embedded software, but also how to use the call chain to the worst-case stack depth as input to a whole-program inliner. The stack tool operates on binaries while the inliner works on source code. This integrated analysis and optimization reduced the overall stack consumption of a number of embedded systems, on average, by 32% when compared to a smart inlining policy whose goal was to improve performance without causing code bloat. We are working on more sophisticated feedback loops, such as one that integrates stack depth bounding with an optimization that maps a number of logically concurrent activities onto a smaller number of OS-supported threads. Eliminating the potential for preemption will make it possible to eliminate some locks, further reducing computations’ memory and CPU requirements.

A number of factors motivate VIPA.

First, since information is both gained and lost at each level of abstraction in a system, analysis at multiple levels is necessary. For example, accurate estimates of worst-case execution time can only be determined by looking at a binary program; exceptions and concurrency are best analyzed at the source level; and, real-time deadlines and mutually exclusive modes are only apparent at the model level.

Second, each embedded system has its own set of goals,

with different prioritizations among energy usage, development time, reliability, footprint, etc. By connecting a number of independent tools, and by not forcing developers into any particular design methodology, VIPA will support highly diverse combinations of goals.

Third, the field of program analysis is rapidly advancing, and tools for finding race conditions, deadlocks, protocol errors, worst-case execution times, worst-case memory usage, etc. are becoming ever more useful. There is a large and growing asymmetry between the resources available on a typical workstation and the resources on a typical embedded system — this asymmetry can be exploited in order to perform advanced analyses and optimizations.

This research faces a number of challenges. First, how will the tools interact with each other? We are developing several exchange formats for program analysis results; one is centered around the callgraphs for a system, another around its concurrency structure: what are the concurrent activities and how do they interact? A number of existing analysis and optimization tools developed at Utah and elsewhere will be adapted to use these exchange formats. Second, what are the most useful feedback loops between analysis and optimization tools, and in what circumstances is each loop useful? We have already designed and tested a feedback loop that reduces stack memory consumption, and we hope to test a number of other loops in the near future. Third, can users be presented with coherent error messages, at the right level of abstraction, in the presence of a number of interacting analysis tools? We believe this to be an important challenge but have not yet made much progress towards solving it. Of course, machines often have trouble answering “why” questions because they usually do not keep track of the reasons that previous decisions were made, and because the correct answer can depend on the frame of mind of a human.

We believe that these challenges can be surmounted. If so, VIPA will help developers create systems at higher levels of abstraction, since fewer optimizations and customizations will have to be performed manually.

References

- [1] J. Regehr, A. Reid, and K. Webb. Eliminating stack overflow by abstract interpretation. In *Proc. of the 3rd International Conf. on Embedded Software (EMSOFT)*, Philadelphia, PA, Oct. 2003.