An Esterel Virtual Machine (EVM)

Aruchunan Vaseekaran, Tamara Blain
Dept. of Computer Science
Columbia University, New York, NY
av62@columbia.edu ansaar@speakeasy.net

October 29, 2002

Abstract

Esterel is a synchronous, imperative language designed to specify deterministic control systems. However compilers and run time environments for Esterel are not available on low cost microprocessors and microcontrollers. We address this shortcoming by designing and implementing an Esterel Virtual Machine and compiler that will operate on a variety of low footprint target platforms. We will demonstrate the feasibility of this approach by implementing ab EVM for the Lego Mindstorms Platform - a Programmable Robot Construction Kit based on the Hitachi H8 microcontroller. We will compare the efficiency of this approach to the traditional approach of generating native code. We will show that it requires less memory at the cost of execution speed.

1 Introduction

The Esterel programming language [1] is an imperative control flow language which includes semantics for concurrency and preemption based on a synchronous global clock. Esterel programs are deterministic and its semantics can be formally defined. As such Esterel is highly suitable for building deterministic control systems. Unfortunately, Esterel is not widely available on many low cost embedded controllers.

A method of making programs highly portable is to compile then for an abstract virtual machine and then build virtual machine interpreters for all platforms on which the programs are to run. Programs written for the Esterel programming language can also be made highly portable in this manner at the cost of execution speed.

In this project we will design an Esterel Virtual Machine (EVM) and modify an existing compiler to generate code for it. The EVM will be a small C program. It will have instructions for signal handling, multiple threads of execution and context switching between threads.

We will then use the EVM to run an Esterel program on a microcontroller platform. The platform we will target is the LEGO Mindstorms Robot construction kit [5]. This brain of this kit is called the RX brick and is based on a Hitachi H8 microcontroller with 36 of externel RAM and 16K of on-chip ROM [1]. This will demonstrate a real Esterel application based on an EVM approach.

Finally we will contrast the approach of generating EVM code to that of generating native machine code. We expect to find that the EVM approach is less efficient by 2 orders of magnitude in terms of execution speed, but that it is much more efficient in terms of total memory footprint.

The remainder of this paper consists of the following sections: Related Work, Design of the EVM, Compiler for the EVM, An EVM Application, Comparing the EVM Approach, and Conclusions and Future Work.

2 Related Work

The approach of compiling languages for virtual machines is not new. It was used at least as far back as the 1970's in Pascal Compilers [6]. The USCD Pascal Compiler generated code for an abstract machine called the P-Code Machine. The P-Code machine was stack based and supported a global heap. It understood the primitive types supported by Pascal such as integers and sets. It had no support for concurrency or locking.

A more recent example of the use of virtual machines is the Java Virtual Machine [8] (JVM). THe JVM was designed for use in networked environments to ensure the secure platform independant delivery of programs accross a variety of computers and devices. All Java programs are compiled into Java Byte code instructions for the Java Virtual Machine. At runtime, the Byte code instructions are executed by the virtual machine. This process is sometimes optimized using a Just-In-Time (JIT) compiler, which will compile the Byte codes into native instructions at run-time.

The JVM is stack based like the P-Code machine but also supports higher level abstractions such as multiple threads of execution, object creation, object de-allocation and monitors for protecting critical sections of code.

The Common Intermediate Layer (CIL) [7] in Microsoft's .NET environment is another example of a virtual machine. All languages in the .NET environment are compiled into CIL code. Like Java byte codes, CIL supports high level object abstractions. The fact that all languages are compiled into CIL, means that modules written in different languages can interoperate in the same program.

A very different example of a virtual machine is VMWARE [2], which is used to simulate real hardware on real-platform. This software enables one to run multiple virtual machines

[1] http://graphics.stanford.edu/ kekoa/rcx/#Hardware

[2] http://www.vmware.com

on a single real machine instance. So a x86 machine running VMWARE can have running within it a virtual x86 machine running Windows NT and another virtual x86 machine running SUN Solaris. This type of virtual machine reduces hardware costs and server room real-estate at the expense of performance.

Many different compilation strategies exist for Esterel. The original approach by Berry [2] was to compile programs into a single finite state machine. This method produces very fast code but suffers from state-space explosion as the program size increases. An improved approach also due to Berry mapped the Esterel program into a network of logic gates and then generated code that simulated the network. This hardware simulation approach scaled very well with program size but generated much slower code than the single finite state machine approach.

The EC compiler by Edwards [3, 4], transforms the program into a concurrent control-flow graph. This graph is then analyzed and transformed into a set of individual program threads which are statically scheduled. We prefer to visualize this as a set of finite state machines which are statically scheduled. The point at which execution of a finite state machine stops and at a which another finite state machine resumes execution can be determined at compile time for most Esterel programs. The EC approach generates smaller and nearly as efficient code as the hardware approach but cannot compile all classes of legal Esterel programs.

3 Project Plan

- Design EVM instruction codes
- Modify ESUIF Compiler to generate EVM codes
  - Create mappings for intermediate constructs such as: if, break, goto, try, resume, parallel, fork, join
  - Identify Static schedule
  - Generate per-thread (fsm) code with explicit thread context switches
- Create EVM Assembler
  - Use awk
- Implement EVM in C on Linux
- Verify EVM on Linux
- Port EVM to Lego Mindstorms RCX
- Verify and demonstrate simple Esterel program on Lego Mindstorms

References

[1] Gerard Berry. Esterel v5 language primer, 2000.

[2] Gerard Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics and implementation. *Scientific Computer Programming*, 19, November 1992.

[3] Stephen A. Edwards. An esterel compiler for large control-dominated systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(2), 2002.

[4] Stephen A. Edwards. ESUIF: An Open Esterel Compiler. *Electronic Notes in Theoretical Computer Science*, 65(5), 2002.

[5] Jonathan B. Knudsen. *The Unofficial Guide to LEGO MINDSTORMS Robots*. O'Reilly, 1999.

[6] Steven Pemberton and Martin Daniels. *Pascal Implementation: The P4 Compiler and Interpreter*. Ellis Horwood, 1982.

[7] Thuan Thai and Hoang Q. Lam. *.NET Framework*. O'Reilly, 2001.

[8] Frank Yellin Tim Lindholm. *The Java Virtual Machine Specification*. Addison-Wesley, 1996.