ARM to x86 Binary Translator

Jimmy Wei, Bharadwaj Vellore
jw2553,vrb2102@columbia.edu

February 19, 2008

1 Project Description

The objective of our project is to develop a binary translator that will enable executables compiled for the ARM architecture and the Linux operating system to run on x86-Linux machines. This is expected to be useful as an emulation environment for testing ARM applications on the more prevalent x86 systems. The application is implemented as an on-the-fly binary translator from the ARM to x86 instruction sets. System calls in the input binary are translated to system calls made to the x86 host.

The objective of the implementation is to not just achieve correctness in the translation but also achieve speed. The implementation is targeted to be able to out-perform standard ARM emulators available for the x86 platforms.

2 Initial Design

2.1 High Level Block View

A block view of the binary translator is as follows.

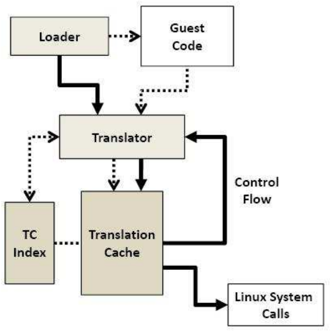

Figure 1: Block Diagram of Binary Translator

The translator block creates a mapping from the given executable into the host instruction set one basic block at a time. The translation cache maintains a record of the basic blocks that have already been translated. Translated code is placed in a translation cache out of which it is executed.

In the interest of speed, the translation is performed once per basic block and basic blocks are then chained. It is also envisaged that the ability to handle self-modifying code is a useful feature. This can be achieved by invalidating translations corresponding to basic blocks whose code is written to.

One of the challenges of the binary translation is to make a mapping from registers in the ARM architecture to registers in the x86 architecture. Given ARM is a load-store architecture, all data-related instructions address only registers. The efficiency of such an architecture is best translated into efficiency on the target machine if target instructions also operated on register operands. However, this is hard since the number of registers available in ARM is larger than the number available in x86 architecture. In the initial design, operands are planned to be placed on the x86 stack, as is conventional in x86 applications without optimization. In future, typical compiler strategies may be employed to create a register mapping.

The ARM v5 instruction set architecture is chosen for this implementation based on its popularity in the embedded systems domain.

2.2 Open Points

The following are identified as open points to be dealt with in the design:

- The execution architecture: It remains to be determined how control is handed to the binary translator.
- Global and static variable handling: Memory addresses in the object code need to be handled.
- IO, Interrupt and Exception handling: It remained to be ascertained how these are handled, if required.

3 Division of Labor

We plan to have weekly meetings and work collaboratively in an 'extreme programming' style to complete the implementation.

4 Milestones

In the first spin (25th Mar) of the project, a functional, but not necessarily fast, implementation is targeted.

The second (final, 29th Apr) milestone will target optimization via register usage colouring and smart register allocation.

Final Deliverable: A demo application ARM application that clearly demonstrates basic functionality as well as speed-up relative an ARM emulator (to be selected).