X86-ARM Binary Hardware Interpreter Research Paper

Adam Gorman
CDA3101 - 8/6/2012

There are currently two very popular and successful instruction set architectures (ISA). The first is the CISC x86 ISA, which is primarily known through Intel and AMD for desktop and laptop devices. The second is the RISC ARM ISA, which has been gaining much popularity in the mobile computing devices, such as smart phones and tablets. With this separation of ISAs, because program binaries can currently only run on the ISA that they were compiled for, thus an application compiled on the ARM won't work on the x86 and vice versa. The XAM or X86-to-ARM binary interpreter hardware is attempting to bridge the gap between the two architectures.

The different architectures are comprised of different instruction sets. The CISC x86 instruction set architecture uses a Complex Instruction Set Computer in which a single instruction can execute several low level operations. The opposing instruction set uses a RISC or Reduced Instruction Set Architecture Computing. The RISC principle attempts to use highly optimized and efficient instructions with significantly less complexity of the CISC. The ARM ISA uses a RISC approach which is better on mobile platforms where optimization is crucial due to more restricted and limited resources such as processor speed and battery concerns. The XAM that is attempting to bridge the gap between the two utilizes a MISA or Multiple Instruction Set Architecture. Many college courses, such as FSU's CDA3100/CDA3101 Computer Architecture course, primarily teach students the RISC design principles using MIPS, which is similar to ARM processors.

While it is possible to compile to different binaries, the hope of a binary hardware interpreter is that a single binary can be executed across both of the different plat forms. There are currently two ideas of thought on how to tackle this problem. The first is through "interpretation" and the second is through "translation". Interpretation basically maps each instruction of one ISA to an equivalent instruction on the target ISA at runtime without caching of any kind. Whereas the translation approach can be dynamic or static attempts to optimize more and reuse pieces of translated code. Although there might be similar instructions among both architectures, they are not identical and may be transformed and executed in different ways.

While software translation may offer a lot of flexibility during this translation or hybrid of the two architectures, there are a lot of possible pitfalls. Some example pitfalls are increased memory usage or execution time of sensitive code. If a translation or crossover is to occur, then it needs to execute the code the same among both platforms consistently without issues. Thus, those who proposed the XAM have decided to pursue a more hardware orientated approach by adding a special decoder that translates and executes it within the processor pipeline.

The XAM hardware interpreter will go within ARMv5 processor architecture. Depending upon the current execution mode selected, it will fetch the instructions and send them through a specific ARM I-decode block. In the x86 execution mode, it will go through a variable length x86-decoder and extract the instructions and forward the m to the XAM. The XAM decoder contains several translation tables which transforms the given x86 instruction into an equivalent ARM instructions.

In this particular hybrid, the XAM does not modify any of the original ARM architecture. There are no additional registers, new hardware or custom instructions. The only major change is the addition of the ARM to x86 decoder and XAM blocks. It is also going to utilize a flat memory model which is not a limitation for more recent x86 code. Some sacrifices have to be made and in this particular instance, the sacrifices are going to be made slightly on the x86 side of things. The ARM is a more optimized architecture and a little more simplified so it makes sense. Most importantly, the ARM processor has had high efficency performance concerns built into its design for some time now.

It was decided in this particular crossover that the x86 registers will be mapped to the ARM registers. This direction of the mappings, as well as the particular register to register mappings that was chosen, were taken into concern of performance compatibility involved with system calls. One of the first major differences in the instructions is involved with addition. Traditionally, x86 instructions has permitted use of processing data from memory locations. However, the ARM instruction required it to be loaded from memory prior to adding and thus had to utilize 2 instructions instead of the x86's one. Another major difference is with function call and returns from subroutines. The x86 ISA has pushed the call address onto the stack and later popped from it whereas the ARM ISA has had a special Link register.

There are a variety of benchmarks and potential interpretations of particular metrics. In the simulation results, it was run on the same hardware but with and without the XAM hardware enabled. There was an original ARM compiled binary for when it was off, and the X86 compile binary for when the XAM hardware was enabled. In many of the simulations, there was less than 1 % difference in instruction count, and on the extreme a 7% variance. There was about a 1%-5% difference in performance for execution time. It typically favored the native ARM compiled binary over the hardware translated binary.

One of the particularly interesting discoveries during the simulation was the instruction count translation. The x86 instructions were usually able to be mapped to an ARM instruction 1-to-1 75% of the time and about 1 to 2 20% of the time. There were a few with 1 to 3 or more mapped instructions. Both systems used about the same amount of energy regardless of which binary they were running, even when the additional XAM hardware was enabled. However, all the performance metrics could probably be optimized quite a bit if future ARM versions were slightly modified to accommodate XAM hardware a little better. In this particular paper, it was assumed that no modifications were to be done to the ARM ISA. However, I think that in future versions of both architectures, there will most likely be less variance in instructions as they will probably incorporate some small changes that would make it easier for the architecture to run binaries from both systems. Especially if you consider that the end user shouldn't need to be concerned with what architecture they are running.

The main purpose of this research paper is to prove the concept that it is possible to translate x86 binaries to ARM instructions at run time by using a few additional pieces of hardware. This is critically important as the future of software and hardware developers will want code to be more portable amongst different systems. There is currently a major divide between Windows, Mac, and Unix systems and that of smartphones. Currently, the mobile market is primarily using ARM architecture, which is making it difficult for people who might want a mobile application run on their x86 computers. The work in the paper was only a first step in the research and development of a binary crossover, but the future looks to be promising with further optimizations still yet to be made. Future versions of windows are supposed to be adding support for ARM processors. Hopefully, within a few years, it won't be a concern or barrier to move between the different Instruction Set Architectures.

Reference Article

X86-ARM Binary Hardware Interpreter, Karaki, H., H. Akkary, and S. Shahidzadeh, Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference, December 2011
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6122235