Binary Translation: A Simulator of Cross-ISA CPU Environment for Application

陈禹诚 2021215221

Binary translation is a CPU simulation technique that enables software run on other machines that deliver near-native code performance. Binary translation was born from emulation in late 1980s technology to provide the transition from the legacy CISC machine to the new RISC machine. Such technology was developed by a hardware manufacturer interested in marketing a new RISC platform. From the middle 1990, binary translation technology was used for translating competitive applications to the desired hardware platform.

By far, binary translation technique has been widely used in all kinds of computer system. For instance, virtual machines, debuggers, dynamic optimizer, ISA simulator, etc. They virtualize the guest binary execution environment by interposing a layer of software between program and the real CPU.

A Brief Introduction towards Binary Translation Methods

In this section, we will introduce three types of binary translation: (1) ISA-simulator (2) DBT (3) SBT and investigate its corresponding examples.

ISA Simulator

ISA is the interface between software and hardware (e.g. x86, arm, RISC-V). ISA-Simulator is a software that emulator the behavior of CPU of that ISA (e.g. Spike). Generally ISA-Simulator loads a cross-compiled ELF file of target ISA and run it on the host ISA. For example, Spike run a RISC-V target binary on a personal x86 computer. Spike acts as a functional model of a RISC-V CPU, after it loads the ELF, it will instruction fetch each instruction, and then decode and execution, finally write back to its memory simulator. Spike is an instruction basis simulator. In contrast, cycle accurate simulator provide cycle by cycle simulation that could be used to evaluate the hardware performance, but generally slower than instruction basis simulator. Gem5 is one of the state of the art example. Gem5 provides a highly configurable simulation framework, multiple ISAs, and diverse CPU models. GEMS complements these features with a detailed and flexible memory system, including support for multiple cache coherence protocols and interconnect models. Currently, gem5 supports most commercial ISAs (ARM, ALPHA, MIPS, Power, SPARC, and x86), including booting Linux on three of them (ARM, ALPHA, and x86). Gem5 allow user to provide ISA or cache architecture in their DSL and generate the

corresponding simulator of it. All gem5 simulator documentation and information is available at the website http://www.gem5.org. The website includes instructions on how to check out, build, and run the gem5 simulator, as well as how to download supplemental support files like OS binaries and disk images.

Dynamic Binary Translation

Dynamic binary translation is a CPU host simulation technique that translate the binary on the runtime, which is quite similar to JIT(Just in time) compilation technique. Generally, dynamic binary translator will translate code basic block on the fly. A code basic block is a segment of code that start with the entry or jump target and end with a branch instruction. When entering the basic block, dynamic binary translator will check if the basic block has been translated or not. If it's not translated, dynamic binary translator will do translation for that block using the host CPU ISA assembly and cache the translated block in the code cache. Comparing with the isa-simulator(interpreter like binary translation), the advantage of dynamic binary translation is that code of translated block can be reused, which greatly speed up the execution speed. A very famous example for dynamic binary translation is QEMU. QEMU is a generic and open source machine emulator and virtualizer. It use hand coded translation rule to turn target CPU instruction into its micro operation code. Micro operation code is much less than the original target CPU code(could consider this as an IR). Then every micro operation will be mapped to a C code and compile to an object file. The object file is host executable for the micro operation that corresponding to the target assembly. Finally QEMU dyngen will generate a dynamic code generator which can take in micro operation and output host code block. The dynamic code generator is the exactly the dynamic binary translator we talk about.

Static Binary Translation

Static binary translation, on the other hand, will do the translation process before the execution. In the past, dynamic binary translation (DBT) has been widely used for migration of applications since some challenging binary translation problems are avoided, such as: code detection, indirect branches. Static Binary Translation (SBT) is generally considered as a less general solution, and it was not actively investigated. However, SBT has the advantage of carrying out a more active optimization which can create more compact codes and higher code quality. Generally, SBT can translate applications with less storage, processor cycles and power, and boot faster. A famous static binary translation tool is SecondWrite. SecondWrite lift executable to LLVM general IR, which make it possible for application of binary rewriting, symbolic execution and source code recover. SecondWrite use speculative disassembly method to settle code discovery problem and use binary characterization method to solve indirect control transfer instruction(indirect-CTI) problem. Of course, for PIC indirect

CTI problem, self modifying code problem, obfuscated code problem, SecondWrite can't process.

Indirect CTI problem is one of the most challenging problem in static binary translation. Most of the work use a runtime hub to handle the mapping of source register space to the destination register space. The mapping table of the hub is constructed by overestimating. The worst case of overestimating is that every source address - destination address is stored as a key-value in the mapping table. By analyzing the compiler pattern the mapping table size could be decrease. For example we can identify the next instruction behind the return instruction is possibly a target address, we will map this in the mapping table. Other method like binary characterization will scan the ELF and identify every value that lie in the code segment and consider them as source of indirect branch. When PIC compilation flag is turn on, this method will be ineffective.

ISA Simulator vs DBT vs SBT

From ISA Simulator to dynamic binary translator to static binary translator, the speed of execution target binary is increasing and the architecture detail information of target information exposed is decreasing. ISA simulator, dynamic binary translator, static binary translator are correspond to interpreter, just in time compiler and traditional ahead of time compiler respectively. ISA Simulator has the heaviest runtime system to simulate the target CPU on the host, while dynamic binary translator alleviate runtime overhead by caching the translated block, and the static binary move the runtime overhead to the compile time with aggressive compilation optimization. So from the aspect of time consuming, the static binary translation will be a good choice. Meanwhile from the aspect of space consuming, the ISA simulator and static binary translator has relatively fix space overhead, in another word, the ISA simulator and static binary translator need no extra space to make them execute faster. For dynamic binary translator, if sufficient space is provided, code could be cached to the external storage and the performance will approximate to the static binary translator if every execution path is covered and cached.

Research Frontier of Binary Translation

In this section, we will present some of the hot topic in binary translation field.

Learning Base Binary Translation

Wenwen Wang et.al 2018 use a mapping table to learn binary translation rules from available source programs using compilers' debugging facilities. Basically, they compile a same source code to x86 assembly and arm assembly. Use this mapping

relationship to train the mapping table. They use symbolic execution to make sure the translated code is semantically equivalent to the source code. When the rule can't cover the binary translation emulation will be introduced.

Jiang et.al 2020 parameterize the assembly instruction, which expand the scope of learning rule, which increase the dynamic coverage from 69.7% to 95.5%.

Post Optimization

Zhang X. et.al 2015 propose a fast cross-ISA binary translator with post-optimization. When dynamic binary translator put a code block into instruction cache, their translator will build a host specific data dependence graph and perform several passes on it. After passes, assembly printer will be called to print the assembly according to the optimized graph.

Binary Layout Agnostic Recompilation

David Williams Kings et.al propose egalito system to providing binary rewriting infrastructure. This work use boundary check to distinguish code and data. This static binary translator provide very high accuracy recovery of binary lifting and rewriting on binary with flag -fpie on. When it comes to stripped binary, it use coarse grain heuristic to search possible CFG edges to decouple relevant function.

Multicores Machine Binary Translation

Speed, portability, and correctness have traditionally been the main requirements for dynamic binary translation systems. With the increasing availability of multi-core machines as guests and emulation hosts, scalability become more important on dynamic binary translator designation. However it has been an elusive goal for two reasons: contention over common data structures, such as the translation cache, is difficult to avoid without compromising performance and ISA disparities between the guest and the host can compromise correctness. e.g. mismatches in the memory consistency model and the semantics of atomic operations.

Emilio G. Cota et.al address these challenges in a simple and memory efficient way, by showing a multi-threaded DBT-based simulator extended in an architecture-independent manner. They explored the trade-offs existence when simulating atomic operations across ISA, and propose a new method of correct and scalable simulation.

Peephole Superoptimizer

Sorav bansal et.al proposed a new solution to perform binary translation. The code generated by this solution is comparable to or better than the existing binary translator which has much less engineering effort. Instead of manually coding translation instruction set to another, their method automatically use super optimization technology to learn translation rules. They have implemented a PowerPC to x86 binary translator. Translate and report the results of small and large compute-intensive benchmarks. Compared with native compiler, their translated code achieved 67% median performance in large benchmarks and on some small benchmarks stress testing is even better than native compilers.

Heuristic Free Binary Translator

Anil Altinay et.al propose BinRec, a new method of binary recompilation without heuristics method. It can improve dynamics trace the binary file to the intermediate representation (IR) at the compiler level and lower the IR back to the "recovered" binary file. IR based designation enables BinRec to apply rich program conversions, for example, compiler-based optimized delivery, source code recovery.

Atomic Instruction Translation

Ziyi Zhao et.al based on current dynamic binary translator couldn’t ensure atomic security on ABA problem (To emulate strong atomic primitive LL/SC of RISC machine on a CISC machine). It’s always easy to emulate CAS primitive by LL/SC, but hard to use CAS to emulate LL/SC without additional hardware support. This work propose hash table based store test and page protection base store test to resolve the problem of emulating LL/SC on a CISC machine.

Full System Simulator

System level dynamic binary translation provides the ability to boot the operating system and execute its programs compiled for the target instruction set architecture different from the host. Due to its performance-critical nature, the system-level DBT framework is usually Hand-coded and highly optimized, whether for their guests or host architecture. Although this leads to good performance, to support the engineering cost of the new system is very high.

Tom Spink et.al developed a new retargetable DBT hypervisor, which includes specific guest modules generated from high-level guests machine specifications. Their system not only simplifies the retargeting of DBT, but also offers performance levels higher than existing DBT solutions created manually.

Conclusion

In this article, we introduced some basic concept of cross ISA CPU simulator: binary translator and three kinds of it: ISA Simulator, dynamic binary translator and static binary translator. We present some state of the art academic research of different topic. From learning based translator to full system simulator. We believe in the future, binary translation technique will be widely applied in a large number of aspects, from virtual machine to debugger, in computer science.

Comment on Course:

The lecture given by Chen Weijian professor of simulator is interesting. I am also a researcher of cross isa CPU simulator. Compared to my field, his topic is more focus on some event or nature phenomenon, the simulator is driven by stochastic generator, while my research on CPU simulator put more attention on accurate on functional unit and performance. It's pleasure and interesting to listen to his lecture and get inspired in designing my simulator. Thanks.