Intel Processor Trace for Reconstructing Guest Control Flow in a Dynamic Binary Translator

University of St Andrews

Robbie Wallace Tom Spink (Supervisor)

Motivation

A familiar problem programmers face is trying to figure out what their code is doing. Some programmers may add print statements to their code, allowing them to see the control flow.

There are several problems with this approach.

- Adding the print statements changes what code is being run. Possibly affecting the result.
- What happens if the language we are dealing with doesn't have print statements?
- What if we only have access to the compiled binary of a program?

A better approach is to use some program that takes in the compiled binary of a program and returns a list of all basic blocks executed by a program and in what order. Where a basic block sequence of instructions with a single entry point, containing no branches, ending with a single exit branch.

One fundamental difficulty with this approach is that a single program can contain many basic blocks (1000s), each of which can be executed many times. If our program has to record that a block has been executed every time the control flow jumps to a block, it can add significant overhead to program execution.

Intel Processor Trace

Intel Processor Trace (Intel PT) is an extension to the Intel architecture that allows for collecting information about program execution.

- This approach is hardware-based; as such, it has minimal impact on the run-time performance of a program.
- However, as it is an Intel hardware feature, it cannot be used on programs written for other architectures, e.g. ARM
- Intel PT also generates a vast amount of data 1GB per second of run-time.

To collect this data, we can use the perf interface on Linux, which, when activated, will provide a stream of IntelPT data about a given process.

QEMU & Dynamic Binary Translation

Binary Translation is the process of translating object code written in a guest Instruction Set Architecture (ISA) to a target or host ISA. One Method of performing this is Static Binary Translation. However,

- Some instructions only exist in one ISA, so there does not exist a clean translation from one ISA to another.
- Many programs make use of dynamic jumps. These involve changing control flow to a location unknown at compile time.

To solve these problems, Dynamic Binary Translation (DBT) is used. This method involves using a run-time system to translate binary code as it is executed. This run-time system contains methods to deal with computed jumps or simulate instructions that don't exist on a given machine. QEMU is an open-source machine emulator and virtualizer that uses DBT.

Results

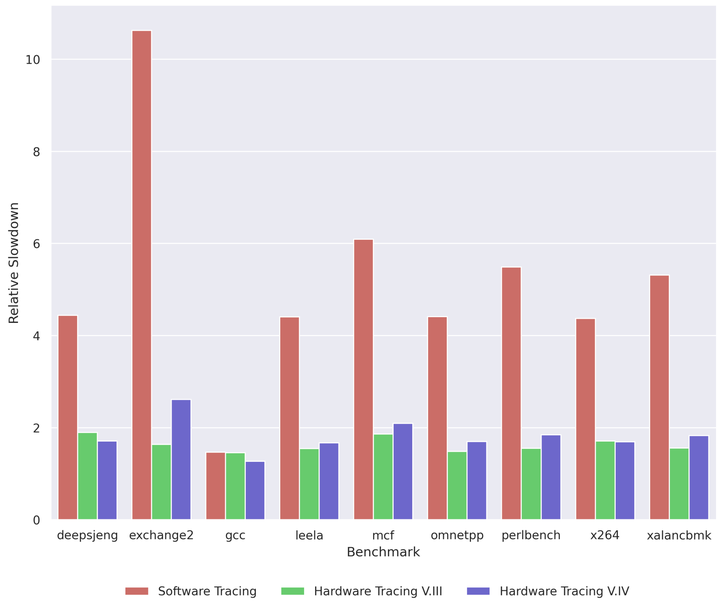

Figure 1. Final Qemu Runtime

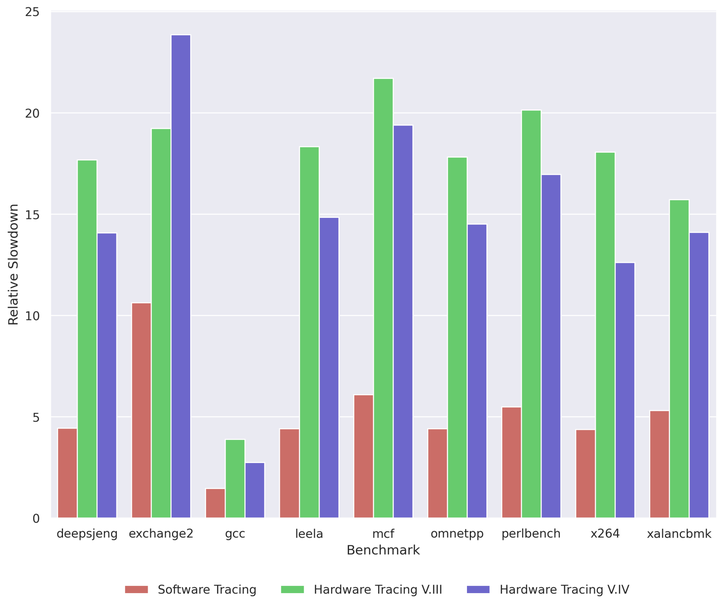

Figure 2. Final Qemu and Parsing runtime

Combining Intel PT and QEMU

The core component of this project was to combine Intel PT and QEMU. Doing so would allow us to use the Intel PT features to generate a control flow trace for a program written in any of the supported QEMU ISAs.

First, we need to collect Intel PT data in QEMU

- To do this, a second thread is added to QEMU. This thread can record Intel PT about the QEMU thread and store it in a data file.
- The Intel PT data generated by this will only trace the execution of the translated Binary Code; we want to trace the original pre-translated code.
- As such, we also need to collect mapping information so that we can map the translated trace to a trace representing the original program.

Second, we need to deal with Intel PTs parsing requirements

- Intel PT data is highly compressed and assumes the parser can access the executed source code. There is no source code, in this case; QEMU generates translated code at run-time
- QEMU needs to be modified to output key information during translation so our Intel PT parser can use it.

Generating the Final Trace

With the Intel PT data, the Source Code data, and the Mapping data, all being generated by QEMU, we can write a parser to transform all of that into a control flow trace.

This method was called 'Hardware Tracing VIII'. Looking at Figures 1 and 2, they show that the QEMU run-time is significantly faster than a traditional software-only approach to generating a control flow trace. However, when factoring in parsing time, it is significantly slower.

To deal with this, QEMU was modified to force Intel PT to generate unique markers at the start of each basic block. This would mean the parser no longer needs the Source Code data, hopefully speeding up parsing time.

This alternative method was called 'Hardware Tracing V.IV'. When looking at Figure 2, it shows that the speed-up gained from this new method is still not enough to make hardware completely outperform software.

Future Work

- Currently, the parser is a separate program from QEMU; bringing the parser into QEMU will make it a more usable and faster process.
- To speed up the parser significantly, multithreading will need to be used. Intel PT is separated into chunks, which can be parsed concurrently, and on a 6-core machine, this could lead to a 6-fold increase in total runtime.
- There are still further areas for performance increase both in the parser design and the modifications done to QEMU. These areas might only lead to slight performance boosts but are worth investing in after the first two future works elements have been completed.