PIN: Intel's Dynamic Binary Instrumentation Engine

Mainak Chaudhuri
Dept. of Computer Science and Engineering,
IIT Kanpur



1

How to do instrumentation

- · Where to insert code
 - For cache study, insert new code for every load and store instruction
 - For branch predictor study, insert new code for every branch instruction
- · What code to insert
 - For cache study, new code should model a cache or a cache hierarchy
 - Input to this code is the intercepted load/store
 - For branch predictor study, new code should model a branch predictor
 - Input to this code is the intercepted branch³



- What is instrumentation?
 - A technique for inserting extra code into an application to observe/study/change its behavior
 - Primary purpose is program analysis and performance profiling
 - There are many other use cases such as architectural studies by intercepting function calls, or specific type of instructions, or all instructions
 - Branch predictor analysis, cache analysis, etc.
 - Any microarchitecture feature can be analyzed once the right set of instructions or API calls is captured

How to do instrumentation

- · How to insert new code
 - Source code instrumentation
 - · Directly changes the source code
 - Drawbacks: need access to source code, cannot instrument external libraries, need recompilation
 - Static binary instrumentation
 - Directly changes the binary after it is compiled
 - · Positive: does not need access to source code
 - Drawbacks: branch targets need careful manipulation, cannot instrument dynamically loaded libraries and objects
 - Dynamic binary instrumentation
 - · Inject code in binary at run-time (a la JIT comp.)
 - Positives: source code not needed, can instrument anything that runs as part of the binary's AS







semi stotice kindel done outel per inster

Instrumentation coole - runs once feer instruction ordering running Analysis coole - runs of times the instruction is run

Pin

- Does dynamic binary instrumentation
 - Takes two inputs: the application binary and an instrumentation program written in C/C++
 - The instrumentation program describes where in the binary to insert new code and what code to insert (writing this program is the main task)
 - The instrumentation program is known as a pin tool and is compiled as a shared object
 - As the application runs, the pintool is used to dynamically and incrementally instrument part of the binary that is about to execute
 - The already instrumented parts of the binary are held in a software cache to avoid re-instrumenting those parts in future provided they are not evicted from the cache (what runs is the instrumented binary)
 - Concepts similar to just-in-time compilation are used to affect dynamic instrumentation

• A Pin tool has two parts

- Instrumentation routine and analysis routine
- Instrumentation routine is called by Pin whenever a portion of the binary needs to be instrumented
 - Inserts calls to the analysis routine at the appropriate locations of the binary and generates the new binary that will be executed
 - Where to insert the calls is specified in the instrumentation routine
 - Caches generated binary for future reuse
 - Ideally, instrumentation routine is invoked exactly once for each executed static instruction; may require multiple invocations if JIT code cache is small



- Analysis routine defines what the inserted code will do

Using Pin

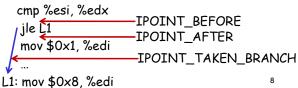
- Two possible ways to use Pin
 - Launch a binary and instrument it on-the-fly pin -t mypintool.so -- application binary Example: pin -t inscount.so -- /bin/ls
 - Executes Is and counts the number of executed instructions
 - Attach to an already running application, instrument it for a while, and detach pin-pid 1234-t mypintool.so
 - Detaching requires using the PIN_Detach function in the instrumentation routine so that the application returns to executing original binary



· Useful for instrumenting already running servers

Writing Pin tools

- Example demo: counting instructions
- Instrumentation points
 - IPOINT_BEFORE
 - IPOINT_AFTER (fall through)
 - IPOINT_TAKEN_BRANCH (taken edge)
 - Need to be careful about IPOINT_AFTER and IPOINT_TAKEN_BRANCH



load x, \$1 and writing to same add \$1,\$1,1 mem operand store \$1, x

Writing Pin tools

- Each argument to analysis routine is passed as a tuple: type, value
 - The value field is unnecessary in some cases depending on the type
 - The type is one of Pin's internal data types
 - Examples:
 - To pass an integer value: IARG_UINT32, value
 - To pass instruction pointer: IARG_INST_PTR or IARG_ADDRINT, Ins_Address(ins)
 - To pass the value of a certain register:
 IARG_REG_VALUE, register name (e.g., REG_AX)
 - To pass the target address of a branch:
 IARG BRANCH TARGET ADDR



- Passing arguments to analysis routine
 - More examples: virtual adds
 - To pass the effective address of a memory operand: IARG_MEMORYOP_EA, mem_operand_id
 - Intel x86 instructions can have multiple memory operands; mem_operand_id starts from 0 and identifies which memory operand in the instruction is being referred to
 - · Refer to the online Pin manual for more
 - Use case: IP trace and memory address trace demos



10

Instrumentation granularity

- Instrumentation can be done at different grains
 - For each instruction
 - For each basic block
 - A basic block is a sequence of instructions terminating at a control flow changing instruction
 - A basic block has a unique single entry point and a unique single exit point
 - For each trace
 - A trace is a sequence of basic blocks terminating at an unconditional control flow changing instruction (single entry, multiple exits)



- For each routine or function

Instrumentation granularity

- Instrumenting at a coarser grain can speed up the instrumented code
 - Less instrumentation overhead
 - For example, the instructions can be counted per basic block and then added to the total instruction count on each visit to a basic block
 - Another example is extracting statistics about various functions that are executed



12

Debugging Pin tool

- · A bit involved because the actual executable is Pin and not the Pin tool
 - Step 1: invoke gdb for debugging Pin gdb pin
 - Step 2: in another window, launch Pin with the Pin tool and the "-pause_tool" option pin -pause_tool 10 -t mypintool.so -- executable Pin will say this: "pausing to attach to pid nnnnn"
 - Step 3: go back to the gdb prompt in the other window and do the following (gdb) attach nnnnn (where nnnnn is the pid) (gdb) break main 13 (gdb) cont

