Pin
Building Customized Program Analysis Tools with Dynamic Instrumentation
CK Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Kim Hazelwood
Intel
Vijay Janapa Reddi
University of Colorado
http://rogue.colorado.edu/Pin
Slides modified by me from original PLDI’05 talk:
www.ckluk.org/ck/talks/pin_pldi_2005.ppt

Instrumentation
• Existing binary-level instrumentation systems:
  – Static:
    • ATOM, EEL, Etch, Morph
  – Dynamic:
    • Dyninst, Vulcan, DTrace, Valgrind, Strata, DynamoRIO
  • Most existing frameworks are not limited to inserting extra code, but allow modification to program code as the program executes

Pin is a new dynamic binary instrumentation system

Advantages of Pin Instrumentation
1. Easy-to-use Instrumentation API
   – Instrumentation code written in C/C++/asm
   – ATOM-like API, based on procedure calls

A Pintool for Tracing Memory Writes

Aside: Trace

• A straight-line sequence of instructions that terminates at one of the conditions
  – Unconditional control transfer (e.g., call)
  – A predefined number of conditional control transfers
  – A predefined number of instructions have been fetched in the trace
Dynamic Instrumentation

Original code

Code cache

Pin fetches trace starting block 1 and start instrumentation

Pin

Dynamic Instrumentation

Original code

Code cache

Pin transfers control into code cache (block 1)

Pin

Dynamic Instrumentation

Original code

Code cache

Pin fetches and instrument a new trace

Pin

Pin’s Software Architecture

Address space

Pin

Instrumentation APIs

Virtual Machine (VM)

JIT Compiler

Emulation Unit

Code Cache

Operating System

Hardware

Pin tool

3 programs (Pin, Pintool, App) in same address space:

- User-level only
- Through which Pintool communicates with Pin
- Dynamically compile and instrument
- Handle instrs that can’t be directly executed (e.g., syscalls)

Code cache:

- Stores compiled code

- Coordinated by VM

Register Re-allocation

- Instrumented code needs extra registers. E.g.:
  - Virtual registers available to the tool
  - A virtual stack pointer pointing to the instrumentation stack
  - Many more …

- Approaches to get extra registers:
  1. Ad-hoc (e.g., DynamoRIO, Strata, Dyninst)
     - Whenever you need a register, split one and fill it afterward
  2. Re-allocate all registers during compilation
     a. Local allocation (e.g., Valgrind)
        - Allocate registers independently within each trace
     b. Global allocation (Pin)
        - Allocate registers across traces (can be inter-procedural)

Pin Internal Details

- Loading of Pin, Pintool, & Application
- An Improved Trace Linking Technique
- Register Re-allocation
- Instrumentation Optimizations
- Multithreading Support

Register Re-allocation

- Instrumented code needs extra registers. E.g.:
  - Virtual registers available to the tool
  - A virtual stack pointer pointing to the instrumentation stack
  - Many more …

- Approaches to get extra registers:
  1. Ad-hoc (e.g., DynamoRIO, Strata, Dyninst)
     - Whenever you need a register, split one and fill it afterward
  2. Re-allocate all registers during compilation
     a. Local allocation (e.g., Valgrind)
        - Allocate registers independently within each trace
     b. Global allocation (Pin)
        - Allocate registers across traces (can be inter-procedural)
Valgrind’s Register Re-allocation

Original Code
mov 1, %eax
mov 2, %ebx
cmp %ecx, %edx
jz t
add 1, %eax
sub 2, %ebx
t:

Trace 1
mov 1, %eax
mov 2, %ebx
jmp 1

Trace 2
mov %eax, SPILL
mov %esi, SPILL
mov %esp, SPILL
mov %edi, SPILL
mov %eax, %edx
cmp %edx, 0
je <target2>
add %ecx, %edx
cmp %edx, 0
je <target2>
t:

Original Code
mov 1, %eax
mov 2, %esi
cmp %ecx, %edx
mov %esi, SPILL
mov %eax, SPILL
mov %esp, SPILL
mov %edi, SPILL
add 1, %eax
sub 2, %esi
t:

Trace 1
mov 1, %eax
mov 2, %esi
jmp 1

Trace 2
mov %eax, %edx
mov %esi, SPILL
mov %esp, SPILL
mov %edi, SPILL
add 1, %eax
sub 2, %esi
t:

Simple but inefficient
• All modified registers are spilled at a trace’s end
• Refill registers at a trace’s beginning

Pin’s Register Re-allocation

Scenario (1): Compiling a new trace at a trace exit

Original Code
mov 1, %eax
mov 2, %esi
cmp %ecx, %edx
mov %eax, SPILL
eax
mov %esi, SPILL
mov %esp, SPILL
mov %edi, SPILL
add 1, %eax
sub 2, %esi
t:

Compile Trace 2 using the binding at Trace 1’s exit:

Virtual
Physical
%edx
%edx
%ecx
%ecx
%esi
%ebx
%eax
%eax

No spilling/filling needed across traces

Pin’s Register Re-allocation

Scenario (2): Targeting an already generated trace at a trace exit

Original Code
mov 1, %eax
mov 2, %esi
cmp %ecx, %edx
mov %esi, SPILL
mov SPILL ebx
mov SPILL ebx, %edi
add 1, %eax
sub 2, %esi
t:

Trace 1 (being compiled)
mov 1, %eax
mov 2, %esi
cmp %ecx, %edx
mov %esi, SPILL
mov %esp, SPILL
mov %edi, SPILL
add 1, %eax
sub 2, %esi
t:

Compile Trace 2 in code cache:

Virtual
Physical
%edx
%edx
%ecx
%ecx
%esi
%ebx
%eax
%eax

Minimal spilling/filling code

Instrumentation Optimizations

1. Inline instrumentation code into the application
2. Avoid saving/restoring eflags with liveness analysis
3. Schedule inlined instrumentation code

Example: Instruction Counting

Original code
pushf
push %edx
push %ecx
push %eax
movl 0x3, %eax
call docount
pop %eax
pop %ecx
pop %edx
popf
ret

Original code
push %edu
push %ecx
push %eax
movl 0x3, %eax
call docount
pop %eax
pop %ecx
pop %edu
popf
ret

Instrument without applying any optimization (bailout)

Original Code
pushf
push %edx
push %ecx
push %eax
movl 0x3, %eax
BBL_InsertCall(bbl, IPOINT_BEFORE, docount(), IARG_UINT32, BBL_NumIns(bbl), IARG_END)
pop %edx
pop %ecx
pop %eax
popf

Inlining

Original Code
push %edu
push %ecx
push %eax
movl 0x3, %eax
BBL_InsertCall(bbl, IPOINT_BEFORE, docount(), IARG_UINT32, BBL_NumIns(bbl), IARG_END)
pop %edx
pop %ecx
pop %edu
popf

33 extra instructions executed altogether

3 extra instructions executed
Comparison among Dynamic Instrumentation Tools

- **Valgrind** is a popular instrumentation tool on Linux.
- **DynamoRIO** is the performance leader in dynamic optimization.
- **Pin** automatically provides efficient instrumentation.
**Pin Source Code Organization**

- Pin source organized into generic, architecture-dependent, OS-dependent modules:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>#source files</th>
<th>#source lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic</td>
<td>87 (48%)</td>
<td>53595 (47%)</td>
</tr>
<tr>
<td>x86 (32-bit + 64-bit)</td>
<td>34 (19%)</td>
<td>22794 (20%)</td>
</tr>
<tr>
<td>Itanium</td>
<td>34 (19%)</td>
<td>20474 (18%)</td>
</tr>
<tr>
<td>ARM</td>
<td>27 (14%)</td>
<td>17933 (15%)</td>
</tr>
<tr>
<td>TOTAL</td>
<td>182 (100%)</td>
<td>114796 (100%)</td>
</tr>
</tbody>
</table>

⇒ ~50% code shared among architectures

**Pin Applications**

- Sample tools in the Pin distribution:
  - Cache simulators, branch predictors, address tracer, syscall tracer, edge profiler, stride profiler

- Some tools developed and used inside Intel:
  - Opcodemix (analyze code generated by compilers)
  - PinPoints (find representative regions in programs to simulate)
  - A tool for detecting memory bugs

- Some companies are writing their own Pintools:
  - A major database vendor, a major search engine provider

- Some universities using Pin in teaching and research:
  - U. of Colorado, MIT, Harvard, Princeton, U. of Minnesota, Northeastern, Tufts, University of Rochester, …
Conclusions

- **Pin**
  - A dynamic instrumentation system for building your own program analysis tools
  - Easy to use, robust, transparent, efficient
  - Tool source compatible on IA32, EM64T, Itanium, ARM
  - Works on large applications
    - database, search engine, web browsers, ...
    - Available on Linux; Windows version coming soon
- Downloadable from [http://rogue.colorado.edu/Pin](http://rogue.colorado.edu/Pin)
  - User manual, many example tools, tutorials
  - 3300 downloads since 2004 July
  - Note: Link above broken. Now hosted at Intel

15 years later

- Pin is widely cited!
  - 3860 citations as of April 6th 2019
    - vs. 2280 for Valgrind, 580 for DynamoRIO
  - Hundreds of papers on different Pintools
- Abstraction
  - Architecture independent
  - Ease of writing Pintools --- framework abstracts away low-level details

Questions?

- No well-known Pintool such as DRMemory (DynamoRIO) or MemCheck (Valgrind)
- "Lightweight" Pin (i.e., JIT compilation without instrumentation) works well
- Instruction counting works well too
  - Unclear whether optimizations translate to code with more complex instrumentation
- Less flexible than Valgrind and DynamoRIO