ΕΛΠ 370: Αρχιτεκτονική Υπολογιστών

Φροντιστήριο Αρ. 2

PIN (Dynamic instrumentation of programs)

Slides from **Robert Cohn (Intel)**

Object dump

You can generate the assembly code from the executable.
To do that you have to use object dump. There are various types:
objdumpLINUX-ALPHA
objdumpAIX-MIPS
objdumpLINUX-MIPS
objdump
Example:

objdumpLINUX-ALPHA –d mcf00.peak.ev6 > mcf.out

Run SPEC CPU 2006 - MCF Benchmark

exe/mcf_base.amd64-m64-gcc42-nn data/ref/input/inp.in

SPEC-CPU 2006 command lines:
http://www.cs.ucy.ac.cy/courses/EPL605/Spring2017Files/SPEC%20CPU2006%20command%20lines.pdf

objdump -d exe/mcf_base.amd64-m64-gcc42-nn data/ref/input/inp.in

What is Instrumentation?

A technique that inserts extra code into a program to collect runtime information

```
    counter++;
sub $0xff, %edx
    counter++;
cmp %esi, %edx
    counter++;
jle <L1>
    counter++;
mov $0x1, %edi
    counter++;
add $0x10, %eax
```

PIN is doing…

Binary instrumentation:

- Instrument executables directly

Instrument dynamically – at runtime

Advantages of Pin Instrumentation

Easy-to-use Instrumentation:

- Uses dynamic instrumentation
  - Do not need source code, recompilation, post-linking

Programmable Instrumentation:

- Provides rich APIs to write in C/C++ your own instrumentation tools (called Pintools)

Multiplatform:

- Supports x86, x86-64, Itanium
- Supports Linux, Windows

Robust:

- Instruments real-life applications: Database, web browsers, …
- Instruments multithreaded applications
- Supports signals

Efficient:

- Applies compiler optimizations on instrumentation code

Downloading, installing and running PIN

https://software.intel.com/en-us/articles/pin-a-binary-instrumentation-tool-downloads

```
tar -xvf pin-2.14-67254-gcc.4.4.7-linux.tar.gz
mv pin-2.14-67254-gcc.4.4.7-linux pin-2.14
cd pin-2.14/source/tools/ManualExamples
make (Do not compile all the tools)
pin-2.14/source/tools/ManualExamples> make
  inscount0.test
pin-2.14/source/tools/ManualExamples> ../../../pin -t
  obj-intel64/inscount0.so -- /bin/ls
/pin-2.14/source/tools/ManualExamples> cat inscount.out
Count 791522
./pin.sh  -t source/tools/ManualExamples/obj-
  intel64/inscount0.so -o myOutput.txt -- ls -lsa
```

Using Pin

Launch and instrument an application

```
$ pin –t pintool.so -- application
```

Instrumentation engine (provided in the kit)

Instrumentation tool (write your own, or use one provided in the kit)

Attach to and instrument an application

```
$ pin –mt 0 –t pintool.so –pid 1234
```

Pintool 1: Instruction Count

```
   counter++;
sub $0xff, %edx
   counter++;
cmp %esi, %edx
   counter++;
jle <L1>
   counter++;
mov $0x1, %edi
   counter++;
add $0x10, %eax
```

Pintool 1: Instruction Count Output

```
$ /bin/ls
  Makefile imageload.out itrace proccount
  imageload inscount0 atrace itrace.out

$ pin -t inscount0.so -- /bin/ls
  Makefile imageload.out itrace proccount
  imageload inscount0 atrace itrace.out

Count 422838
```

ManualExamples/inscount0.cpp

```
#include <iostream>
#include "pin.h"

UINT64 icount = 0;

void docount() { icount++; }

void Instruction(INS ins, void *v)
{
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
}

void Fini(INT32 code, void *v)
{ std::cerr << "Count " << icount << endl; }

int main(int argc, char * argv[])
{
    PIN_Init(argc, argv);
    INS_AddInstrumentFunction(Instruction, 0);
    PIN_AddFiniFunction(Fini, 0);
    PIN_StartProgram();
    return 0;
}
```

analysis routine

instrumentation routine

Pintool 2: Instruction Trace

```
        printip(ip);
sub  $0xff, %edx
        printip(ip);
cmp  %esi, %edx
        printip(ip);
jle  <L1>
        printip(ip);
mov  $0x1, %edi
        printip(ip);
add  $0x10, %eax
```

Need to pass ip argument to the analysis routine (printip())

Pintool 2: Instruction Trace Output

```
$ pin -t itrace.so -- /bin/ls
  Makefile imageload.out itrace proccount
  imageload inscount0 atrace itrace.out
```

```
$ head -4 itrace.out
    0x40001e90
    0x40001e91
    0x40001ee4
    0x40001ee5
```

ManualExamples/itrace.cpp

```
#include <stdio.h>
#include "pin.h"
FILE * trace;
void printip(void *ip) { fprintf(trace, "%p\n", ip); }

void Instruction(INS ins, void *v) {
   INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)printip,
                  IARG_INST_PTR, IARG_END);
}
void Fini(INT32 code, void *v) { fclose(trace); }
int main(int argc, char * argv[]) {
    trace = fopen("itrace.out", "w");
    PIN_Init(argc, argv);
    INS_AddInstrumentFunction(Instruction, 0);

    PIN_AddFiniFunction(Fini, 0);
    PIN_StartProgram();
    return 0;
}
```

argument to analysis routine

analysis routine

instrumentation routine

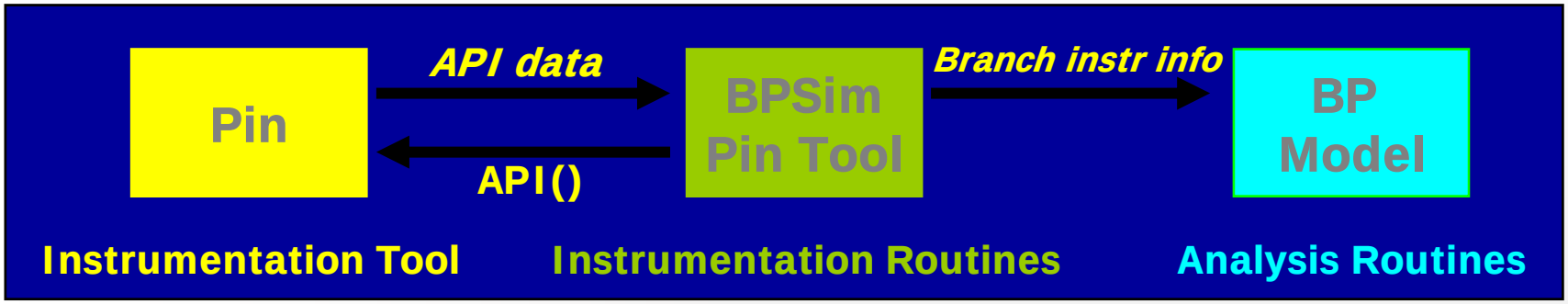
pin -t pinatrace.so -- /bin/ls

```
/*
 *  This file contains an ISA-portable PIN tool for tracing memory accesses.
 */

#include <stdio.h>
#include "pin.H"


FILE * trace;

// Print a memory read record
VOID RecordMemRead(VOID * ip, VOID * addr)
{
    fprintf(trace,"%p: R %p\n", ip, addr);
}

// Print a memory write record
VOID RecordMemWrite(VOID * ip, VOID * addr)
{
    fprintf(trace,"%p: W %p\n", ip, addr);
}
```

```
$ pin -t pinatrace.so -- /bin/ls
Makefile          atrace.o     imageload.o     inscount0.o   itrace.out
Makefile.example  atrace.out   imageload.out   itrace        proccount
atrace            imageload    inscount0       itrace.o      proccount.o
$ head pinatrace.out
0x40001ee0: R 0xbfffe798
0x40001efd: W 0xbfffe7d4
0x40001f09: W 0xbfffe7d8
0x40001f20: W 0xbfffe864
0x40001f20: W 0xbfffe868
0x40001f20: W 0xbfffe86c
0x40001f20: W 0xbfffe870
0x40001f20: W 0xbfffe874
0x40001f20: W 0xbfffe878
0x40001f20: W 0xbfffe87c
$
```

```
// Is called for every instruction and instruments reads and writes
VOID Instruction(INS ins, VOID *v)
{
    // Instruments memory accesses using a predicated call, i.e.
    // the instrumentation is called iff the instruction will actually be executed.
    //
    // The IA-64 architecture has explicitly predicated instructions.
    // On the IA-32 and Intel(R) 64 architectures conditional moves and REP
    // prefixed instructions appear as predicated instructions in Pin.
    UINT32 memOperands = INS_MemoryOperandCount(ins);

    // Iterate over each memory operand of the instruction.
    for (UINT32 memOp = 0; memOp < memOperands; memOp++)
    {
        if (INS_MemoryOperandIsRead(ins, memOp))
        {
            INS_InsertPredicatedCall(
                ins, IPOINT_BEFORE, (AFUNPTR)RecordMemRead,
                IARG_INST_PTR,
                IARG_MEMORYOP_EA, memOp,
                IARG_END);
        }
        // Note that in some architectures a single memory operand can be
        // both read and written (for instance incl (%eax) on IA-32)
        // In that case we instrument it once for read and once for write.
        if (INS_MemoryOperandIsWritten(ins, memOp))
        {
            INS_InsertPredicatedCall(
                ins, IPOINT_BEFORE, (AFUNPTR)RecordMemWrite,
                IARG_INST_PTR,
                IARG_MEMORYOP_EA, memOp,
                IARG_END);
        }
    }
}
```

Branch Predictor Model



BPSim Pin Tool

- Instruments all branches
- Uses API to set up call backs to analysis routines

Branch Predictor Model:

- Detailed branch predictor simulator

BP Implementation

```
BranchPredictor myBPU;

VOID ProcessBranch(ADDRINT PC, ADDRINT targetPC, bool BrTaken) {
  BP_Info pred = myBPU.GetPrediction( PC );
  if( pred.Taken != BrTaken ) {
    // Direction Mispredicted
  }
  if( pred.predTarget != targetPC ) {
    // Target Mispredicted
  }
  myBPU.Update( PC, BrTaken, targetPC);
}
```

ANALYSIS

```
VOID Instruction(INS ins, VOID *v)
{
  if( INS_IsDirectBranchOrCall(ins) || INS_HasFallThrough(ins) )
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR) ProcessBranch,
      ADDRINT, INS_Address(ins),
      IARG_UINT32, INS_DirectBranchOrCallTargetAddress(ins),
      IARG_BRANCH_TAKEN, IARG_END);
}
```

INSTRUMENT

```
int main() {
  PIN_Init();
  INS_AddInstrumentationFunction(Instruction, 0);
  PIN_StartProgram();
}
```

MAIN

DEMO

exe/mcf_base.amd64-m64-gcc42-nn data/ref/input/inp.in

objdump -d exe/mcf_base.amd64-m64-gcc42-nn data/ref/input/inp.in

cd /extraspace/research/cs05np1/pin-3.6/source/tools/ManualExamples

../../../pin -t obj-intel64/itrace.so -- /home/research/cs05np1/tutorials/gdb_tutorial 5 4

cat itrace.out

../../../pin -t obj-intel64/inscount0.so -- /home/research/cs05np1/tutorials/gdb_tutorial 5 4

cat inscount.out