Emulation: Interpretation

Witawas Srisa-an

CSCE496/896: Embedded Systems Design and
Credits

• Most of the material for this lecture is from “Virtual Machines: Versatile Platforms for Systems and Processes”, Smith and Nair, 2005
Emulation

“the process of implementing the interface and functionality of one system or subsystem on a system or subsystem having a different interface and functionality...”
Instruction Set Emulation

- An essential function of a virtual machine
  - the source instruction set (e.g. a program compiled for IA32)
  - the target instruction set (e.g. but runs on PPC)
Forms of Emulation

- Interpretation
- done in software on instruction at a time
Forms of Emulation

- Binary translation
  - amortized the fetch and analysis costs by
    - translating a block of source instructions to target instructions
    - saving the translated code for repeated use
Forms of Emulation

- Translation cost
  - High
  - Low

- Execution cost
  - Low
  - High

- Binary translation
- Interpretation
Interpretation

- Resources
  - each instruction is viewed as an element in an array
  - basic resources similar to a real processor
while (!halt && !interrupt) {
    inst = code[PC];
    opcode = extract(inst, 31, 6);
    switch (opcode) {
        case LUI: LUI(inst);
        case LW: LW(inst);
        ...
    }
    PC = PC + 4;
}

LUI(inst) {
    RT = extract(inst, 25, 5);
    IMM = extract(inst, 15, 16);
    if (RT == 0) {
        interrupt = 1;
        return;
    }
    reg[RT] = IMM << 16;
    PC = PC + 4;
}

LW(inst) {
    ...
}
Interpretation

- Decode and dispatch
  - one source instruction can mean many target instructions
  - many branch instructions which may not be pipeline-friendly
Interpretation

Decode and dispatch (MIPS instruction set)

```c
while (!halt && !interrupt) {
    inst = code[PC];
    opcode = extract(inst, 31, 6);
    switch (opcode) {
        case LUI: LUI(inst);
        case LW: LW(inst);
        ...
    }
}
```

```c
LUI(inst) {
    RT = extract(inst, 25, 5);
    IMM = extract(inst, 15, 16);
    if (RT == 0) {
        interrupt = 1;
        return;
    }
    reg[RT] = IMM << 16;
    PC = PC + 4;
}
```

```c
LW(inst) {
    ...
}
```
Interpretation

- Threaded interpretation

```
LUI:
    RT = extract(inst, 25, 5);
    IMM = extract(inst, 15, 16);
    if (RT == 0) {
        interrupt = 1;
        return;
    }
    reg[RT] = IMM << 16;
    PC = PC + 4;
    if (halt || interrupt) goto exit;
    inst = code[PC];
    opcode = extract(inst, 31, 6);
    routine = dispatch[opcode];
    goto *routine;
...
```
Interpretation

Decode and dispatch

Threaded interpretation

From Virtual Machines by Smith and Nair, 2005
Interpretation

- Threaded Interpretation

- Now, the dispatch table is a bottleneck

- The same instruction can be interpreted multiple times and each time, many extractions are needed

- why not save the extracted information?
Interpretation

- Predecoding
  - decode an instruction and store the info in an intermediate form
  - reuse this intermediate form each time the instruction is emulated
Interpretation

Predecoding (MIPS instruction set)

Source Code

Intermediate Code
Interpretation

Predecoding example (MIPS instruction set)

0x1000: LW     R1, 8(R2)
0x1004: ADD    R3, R3, R1
0x1008: SW     R3, 0(R4)
...

135
1 2 8
032
3 1 03
142
3 4 00

0x10000: LW
0x10008: ADD
0x10010: SW
Interpretation

Predecoding example (MIPS instruction set)

```c
struct instruction {
    unsigned int op;
    unsigned char dest;
    unsigned char src1;
    unsigned short src2;
} code [CODE_SIZE]

LUI:
    RT = code[TPC].dest;
    IMM = code[TPC].src2;
    if (RT == 0) {
        interrupt = 1;
        return;
    }
    reg[RT] = IMM << 16;
    SPC = SPC + 4;
    TPC = TPC + 1;
    if (halt || interrupt) goto exit;
    inst = code[PC];
    opcode = code[TPC].op;
    routine = dispatch[opcode];
    goto *routine;
...
```
Interpretation

- Threaded Interpretation
  - The dispatch table is a bottleneck
    - The same instruction can be interpreted multiple times and each time, many extractions are needed
      - why not save the extracted information?
Interpretation

- Direct threaded interpretation
  - instead of encoding opcode, why not encode the actual address of the interpreter routine?
**Interpretation**

Direct threaded example (MIPS instruction set)

0x1000: LW     R1, 8(R2)
0x1004: ADD    R3, R3, R1
0x1008: SW     R3, 0(R4)

...
Interpretation

Direct threaded example (MIPS instruction set)

```c
struct instruction {
    unsigned int addr;
    unsigned char dest;
    unsigned char src1;
    unsigned short src2;
} code [CODE_SIZE]
```

LUI:
```c
RT = code[TPC].dest;
IMM = code[TPC].src2;
if (RT == 0) {
    interrupt = 1;
    return;
}
reg[RT] = IMM << 16;
SPC = SPC + 4;
TPC = TPC + 1;
if (halt || interrupt) goto exit;
inst = code[PC];
routine = code[TPC].addr;
goto *routine;
...
```
Summary

- We have discussed the basic interpretation techniques
  - decode and dispatch causes many branches
  - threaded interpretation eliminates the dispatch loop but creates a dispatch table
  - predecoding eliminates redundant extractions
  - direct threaded eliminates redundant extractions and the dispatch table