Simulation and Instrumentation

01-1

Purpose and Overview

Instrumentation:
A facility used to determine what an *actual system* is doing.

Simulation:
A facility used to determine what a specified system *would be* doing.

What they do is different (what is vs. what could be) . . .

. . . but how they do it can be similar.

Techniques described here for one can sometimes be used for the other.

EE 7700-4 Lecture Transparency. Formatted 10:12, 21 August 1998 from lsli01.

01-1

References

01-2

See references web page for updated list and links.

Information on Shade and overview of instrumentation and simulation techniques.

B. Cmelik and D. Keppel, "Shade: A fast instruction-set simulator for execution profiling," 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.

Description of instrumentation by sampling.

Jennifer M. Anderson, Lance M. Berc, Jeffrey Dean, Sanjay Ghemawat, Monika R. Henzinger, Shun-Tak A. Leung, Richard L. Sites, Mark T. Vandevoorde, Carl A. Waldspurger and William E. Weihl, "Continuous profiling: where have all the cycles gone?," *ACM Transactions on Computer Systems*, vol. 15, no. 4, Nov. 1997, pp. 357-390.

Description of a simulator and some applications.

Mendel Rosenblum, Edouard Bugnion, Scott Devine, and Steve Herrod, "Using the SimOS Machine Simulator to Study Complex Computer Systems," *ACM Transactions on Modeling and Computer Simulation,* vol. 7, no. 1, Jan. 1997, pp. 78-103.

Instrumentation

01-3

Uses for Instrumentation

Answer question: On system X running program Y ...

... what is the CPI?

... what is the effectiveness of branch prediction?

Typical Procedure

Start with:

- System X.
- Program Y.
- An instrumentation system.

Set up instrumentation system to collect needed data, run, examine data.

Simulation

01-4

Uses for Simulation

Answer question: On system X running program Y ...

... what would the CPI be?
... what would the effectiveness of branch prediction be?

Typical Procedure

Start with:

- A description of system X.
- Program Y.
- A simulation system.

Set up simulator so that it models system X.
Run Y, examine data.

Definitions

01-5

Host:
The machine on which the simulator runs.

Target:
The machine being simulated.

Benchmark:
A program that runs on the simulated machine. (In this context not necessarily a standard or carefully chosen program, just whatever was chosen to run on the target system.)

Trace:
The data produced by an instrumentation system or a file holding such data.

Data Collected (Instrumentation and Simulation)

01-6

Most Detailed:

For any time, contents of any memory location, register (architecturally visible or not), cache line, bus, table, etc.

Number of executions of instruction i that stall pipeline.

Average Detail:

Number of cache hits; cold misses; "warm" misses.

Miss latency.

Less Detail:

Number of instructions executed.

Number of executions of a particular instruction.

Range of Capabilities

01-7

Overall:

Can it run any program?

Does program run as it would on a real system? (Or is it a rough approximation.)

Can benchmark contain self-modifying code?

Are I/O devices simulated in any detail?

Speed: How much slower does benchmark run? $1\times$? $10\times$? $1000\times$?

For instrumentation $\approx 1\times$ possible; for detailed simulation $> 1000\times$ typical.

Timing Data: Can one accurately determine time of events, including run time?
Many do not, not all that do do well.

Is complete system *including kernel* included?

Are exceptions and traps included?

Instrumentation and Simulation Techniques

01-8

Instrumentation and simulation overlap.

Instrumentation Only

- Tracing Hardware: Built in or special-purpose.
- Sampling: Interrupt program and see what it was doing.

Instrumentation and Simulation

- Augmentation: Have program collect its own data or simulate timing.
- Decode and dispatch: Interpret one instruction at a time.
- Static compilation: Translate each program instruction in advance.
- Dynamic compilation: Translate instructions just before needed.

Tracing Hardware

01-9

Hardware added to system to monitor events.

Examples:

- Logic analyzer.
- Special hardware.

If hardware doesn't interfere, can use timing of host system.

Typical data collected:
Addresses on bus, instructions being fetched.

Less cumbersome in 16-bit bus days.

Microcode

01-10

Have microcode write trace information to buffer for later saving.

Disadvantages

Not all machines have writable microcode.

Buffer area fills quickly.

Event Registers

01-11

Provided by manufacturer for tuning system.

Sometimes documented, sometimes secret.

Incremented when event occurs, might be read after benchmark runs.

Types of events determined by processor.
E.g., cache misses, branch miss-predictions, cycles.

Can also trigger an interrupt when reaches a certain value.

Uses:

Count specific events.

Interrupt system at specific events.

Interrupt randomly. (See sampling, below.)

Advantages
Nearly zero slowdown, nearly zero perturbation.

Disadvantages
Limited in what can be collected (on some systems, nothing).

Sampling (Continuous Profiling)

01-12

Interrupt system at random times, check what it's doing, let it proceed.

How system can be interrupted:

Use ordinarily present and frequently occuring interrupt (*e.g.*, page fault).

Induce interrupts: Jiggle mouse, added hardware.

Set event counter to count cycles and interrupt when threshold reached.

Can use timing of host system.

Timing between induced interrupts unperturbed.

Sampling, Continued

01-13

Procedure

At interrupt read time and saved program counter, write to buffer.

Occasionally flush buffer to *sample trace file*.

Use sample traces to deduce number of instruction executions.

In Alpha 21164 *et al* instruction frequency can be used to deduce stalls.

Stalls found by comparing frequency in sample trace to frequency in dynamic instruction sequence.

Sampling, Continued

01-14

Collected Data

Number of instruction executions.

Pipeline stalls.

Advantages

Very Fast

Pipeline stall data available.

Disadvantage

Only works in certain processors.

Augmentation

01-15

Augmentation:
The insertion of code into a benchmark program for instrumentation or simulation.

Modify or augment benchmark program. (Source or assembler.)

Inserted code collects needed data or simulates changed or new instruction.

Because of augmentation, timing must be simulated if needed.

Inserted instructions affect timing by ...
... the time they need to execute ...
... their effect on the cache and pipeline ...
... and the unavailability of any registers they require.

Augmentation Examples

01-16

Examples, instrumentation:

Right after each I_x instruction insert code to increment a count.

At the end of each basic block insert code to increment a cycle count ...
... by approximate or average time to execute the basic block.

At each indirect load count repeat addresses.

Examples, simulation of new instruction:

Have compiler and assembler emit opcode for new instruction.

Replace opcode with emulating instruction sequence; at basic block increment cycle count for unmodified code.

Example, simulation of cache:

Insert cycle counting code at basic blocks.

Replace loads and stores with cache simulator procedure call.

Cache simulator maintains state of cache ...
... updates cycle count based on instruction.

Augmentation Tradeoffs

01-17

Advantages

Fast (compared to schemes below), since target instructions execute on host.

Flexible, can simulate or instrument almost anything.

Disadvantages

Need assembler code for benchmarks and other parts of system.

OS and dynamic code omitted or must be emulated.

Decode and Dispatch

01-18

Simulator reads each instruction, decodes it, calls appropriate routine.

Simple DLX simulator:

```
/*  DLX Simulator */
ir = mem[pc];
switch( INSTR_TYPE(ir) ){
 case TYPE_R:
  rs1 = GET_RS1(ir);
  rs2 = GET_RS2(ir);
  rd  = GET_RD(ir);
  op = GET_TYPE_R_OPERATION(ir);
  switch( op ) {
   case ADD:
      regs[rd] = regs[rs1] + regs[rs2];
      break;
   case OR:
      regs[rd] = regs[rs1]  regs[rs2];
      break;
/* Et cetera. */
```

01-19

Host IR never sees benchmark source.

Speed depends upon level of simulation.

- Usually the slowest technique.

Can simulate to any level of detail.

Static Compilation

01-20

Translate (compile) each benchmark instruction or group of instructions ...
... into host instructions which update state and collect any needed data.

For example:

Arithmetic instruction might change array used for registers.

Instructions might increment a cycle counter.

Difficult without assembler code (since branch targets are not readily available).

Cannot handle self-modifying and dynamically linked code.

Faster than decode & dispatch.

Dynamic Compilation

01-21

Translate (compile) each benchmark instruction or group of instructions . . .
. . . at *simulation* time *just before* needed.

Advantages

Faster than decode & dispatch.

Can handle any code.

Example will be described.

Simulator Timing Detail

01-22

Varies Widely

Least: Correct execution, no timing data.

Can obtain instruction counts, cache hit ratio.

Instruction count can serve as rough estimate of time.

Moderate:

Approximate timing data on uninteresting parts of system ...

... accurate simulation on part being studied.

E.g., Approximate pipeline, simulate memory system accurately.

Detailed:

Simulate all possible delays, resource conflicts, etc.

Very time consuming for modern instruction pipelines.

Needed to simulate some pipeline modifications.

Example Simulator: Shade

01-23

To be used in some homework assignments.

Hosted on Sun SPARC machines, targets SPARC V8 and V9 and a MIPS architecture.

Simulates execution.

Calls user-supplied routines to collect trace (tracer), done frequently.

Calls user-supplied routines to analyze trace (analyzer), done less frequently.

Data Structures:

Array for memory of target system, vmem. (Yes, it can be big.)

Arrays for registers, etc.

Example Simulator: Shade, Continued

01-24

Translations:

Small sections of benchmark code translated into a host code fragment.

Translation changes data structures as real code would change target machine.

Translation also collects trace information.

Code at end of translation jumps to either main loop or next translation.

Overall Operation

Main loop either starts a translation or calls the analyzer routine.

To start translation:

Read PC, look for corresponding translation.

If not present, generate translation.

Jump to translation code.

Note user code called in two places:

Within translations to collect trace data.

By main loop to analyze collected data.

01-25

Shade Limits:

Possibly not well suited to collect timing data.

01-25