9/11 Performance Evaluation Class Note

Calibration of Microprocessor Performance Models:

Author: Bryan Black , John Shen

Sources of Error:

Modeling Error – designers understand task correctly, but implement it wrong

Specification Error – designers is misinformed about the correct functionality

Abstraction Error – Some features are abstracted to keep modesl simple/fast. But the abstraction is wrong.

Background of paper: IBM asked CMU to develop performance simulators. CMU received specification from industries and tried to simulate more realistic. They validated their modeling by comparing their modeling result and result of running on real hardware. They found a lot of modeling error.

Specification Errors: FF multiply-add record instructions need a latency of six cycles and reserve all pipeline stages. (Documented as three cycles)

Complex instructions that modify the overflow bit

Modeling Errors: Mtfsb0 and mtfsb1 dispatching need to dispatch to the ff unit and not the complex integer unit

Instruction finish adds an extra cycle to latency and should be part of the last cycle of execution

Load instructions execute in a single cycle, but it should be two

Complex integer unit is not fully pipelined

We have to always assume there are errors in our modeling and work

Abstraction Error: Branch misprediction paths are not simulated

Data-dependent execution is not simulated

The memory hierarchy assumes L2 are always hit, never be modeled.

Validation by inspection: single step through small code sequences

Okay with trace-driven

Difficult with execution driven

Incremental observation with incremental inputs, like we can do 100 iterations first then 200 iterations

Sanity checks with an array of instrumentation counters
Simple code sequences to exercise boundary conditions of all resources in the model

Validation Phases: Alpha test – exercise instruction latency by executing each instruction one at a time
Beta test – more complicated test than alpha
Random test sequences – up to 100 instructions exercise interactions among instructions and the different components of the microarchitectures
ATPG – automatic test pattern generators as in digital circuit design and testing
Handwritten patterns – to test microarchitectural features for which tests are difficult to generate automatically

Shen's alpha, beta, gamma tests are generated by ATPG. ATPG extracts ISA information.

Infant model validation: Gamma has 29890 instruction counts, all fails in the test
Child model validation: if they allow one cycle error, alpha passed all. And Gamma passed more than 80%. Random sequence still showed huge amount of errors
In early models, misprediction path are just cycle penalties.
Shen also used his infant model and child model to compare with the hardware reference models in order to validate his modeling. They used cjpeg, grep, gperf, and other benchmarks.
In some cases, two errors may cancel each other, but actually there are two big errors existing in the model.

TPC benchmarks:
C (OLTP), H (DSS), E (OLTP), W is the one like amazon, DS(DSS), VMS (a combination of pervious benchmarks running on virtualized machines)
TPC-Queries:
Read and update Queries in OLTP
Read-only are more frequent in DSS
Written in SQL

TPC-C:

Modeling a business worldwide case, OLTP.

1992

metric: business throughput, number of orders processed per min, transactions

TPC-E:

For stock market

Metrics: tpse, $/tpse

Has 33 tables, more than common ones

SPECjbb benchmarks:

This is the benchmark written to shrink down the requirement of the database from TPC.

SPECweb benchmarks:

It does not exist now.

VolanoMark:

Try to model the chat room.

Embedded Benchmarks:

EEMBC Benchmarks – Motorola used it for automobiles embedded processor

BDTI Benchmarks – they are evaluating and scoring your processor

MediaBench : has JPEG, MPEG, and etc.

MiBench – originally is a copy of EEMBC

MorphMark – for embedded java

MIDP Mark – also for java

Andriod – current one for java application and system

Coremark – generic benchmark from EEMBC, targets on processor core, ignores memory and i/o systems