CMP SCI 635: Modern Computer Architecture

Professor Charles Weems

Measuring Experimental Error in Microprocessor Simulation

Name: Kunjal Panchal Date: 10th Sept, 2019

Student ID: 32126469 Paper: 2 – Methodology [Desikan01]

Strengths:

1. The microbenchmark suite used is really powerful. They cover all the possible scenarios that might be encountered in actual programs. [control-conditional (C-C), control-recursive (C-R), control-switch (C-S), and complex-control (C-O), execute-independent (E-I), execute-float-independent (E-F), execute-dependent-n (E-Dn), memory-independent (M-I), memory-dependent microbenchmark (M-D), memory-L2 (M-L2) and memory-memory (M-M), M-L2 are coded to miss in the L1 D-cache on every reference, and in M-M to miss in both the L1 and L2 caches. Finally, the memory-instruction-prefetch (M-IP) benchmark tests the efficacy of instruction prefetching by iterating over an enormous loop that flushes the L1 instruction cache with each iteration.]

2. Despite having the least available information on memory architecture of 21264, the researchers did an impressive job of finding out the most likely configuration by using 3 memory specific benchmarks and lots of variations in RAS cycles, CAS cycles, precharge latency and control latency.

3. The four recommendations in the end of *Conclusion* section: Reproducibility, Consistent Parameters, Common Baselines and Quantified Stability; can be made to serve as important guidelines for any future research as we often see most of the research becomes unreproducible and unprovable without sufficiently detailed hardware and software configurations.

4. Most of the improvements from *sim-initial* to *sim-alpha* are well-reasoned and working on those reasons show a fair amount of error reduction/deviation with only mean 2% error for micro benchmarks and 18% for macro benchmarks.

5. {Meta} Authors own up about shortcomings of memory system models which might save other researchers lot of trouble, who wants to reproduce some results on un/validated simulators and the same can provide as suggestion for future work for other researchers.

Weaknesses:

1. No clear explanation is given on how line predictions can be tuned for *sim-alpha* from two slightly different cases of code sequences of C-C [*C-Ca:Tru64 Unix 5.1 Compaq C V6.3-025 and C-Cb: Digital UNIX 4.0 DEC C V5.9-008*].

2. Author didn't explain how subtracting latency of a bypassed instruction from the execution time in simulator is different from actual 21264 core. There's no mention of how 21264 core implements bypassing of instructions.

3. Does it matter if individual Micro/Macro-validation error rates are really high or low but the mean error is kind of acceptable? Mean error shouldn't matter if we focus on one functionality of a program e.g. If a *sim-alpha* performs great for M-I instructions, is faster than *DS-10L*, but performs poorly for C-R instructions, then even if the good and bad performances evens each other out, most of the real-life applications will have more loops [C-R instructions] than M-I instructions. Averaging the error rates should mean nothing with regard to validating a simulator.

4. {Addendum to Weakness #3} There is no mention of all macro benchmarks being varied enough that their results can be averaged.

5. Is the line and way prediction justified? Is the overhead of way prediction miss, which can fetch upto four 64-byte instruction cache line worth it? How often will it occur?

Questions/Assertions:

1. Is making a good replica of a processor through simulator hard because of the lack of architectural details available for that processor? Because otherwise, we could just copy the physical architecture of the processor, on circuit level, instead of copying the logical abstractions about how one module should act and how it should interact with other modules of the design. That will prevent lot of *Specification Errors* and decrease the number of *Modelling Errors/ Coding bugs*. And if that can't be done because the processor designers don't want to divulge all the low-level circuitry details, then how simulator designers decide which might be the best guess for that module of hardware?

2. {Addendum to Q/A #1} Replicating on circuit level instead of logic level also will give a fairer estimate of signal delays, physical constraints and finite storage structures.

3. Why aren't instructions assigned to clusters in *Slot* stage of the pipeline? Wouldn't assigning instructions at *Slot* stage do away with one-cycle delay of bypassing instructions between clusters? And if the answer is that they are assigned according to their *inum*, which we only know at *Map* stage, why not assigned a cluster right after that, in the *Map* stage itself. {Here, I am assuming that the one cycle delay is because of the change of register files/execution units, which won't be needed to be done if cluster is already determined}

4. "*The events may be sampled at several intervals, from 1,000 cycles to 64K cycles. Larger sampling intervals dilate the execution time less, but introduce additional error when counting events. We chose a sampling interval of 40,000 cycles, which showed the best trade-off between sampling error and instrumentation dilation.*" How exactly the optimal trade-off be identified? Do we have to try many sampling intervals and observe errors in combination with instrumentation dilation?

5. How is having 2 multipliers and 2 adders less expensive (2 cycles) than using 1 adder/multiplier and 3 adders (4 cycles); in *Execute* microbenchmark section.

6. How will sending an older instruction to one cluster if a younger instruction needed to go to the other minimize delay if that older instruction has already been in pipeline, using resources?

7. "*...our implementation of the memory system beyond the L1 cache is less accurate than that of the processing core. While this inaccuracy does not affect the cache-resident microbenchmarks...*" Why/how does implementation of L2, L3 caches; main memory and secondary storage doesn't affect cache-resident benchmarks, while being inaccurate?