Appears in the International Conference on Parallel Architectures and Compilation Techniques (PACT)
Brasov, Romania, September, 2007

Reducing the Impact of Process Variability with Prefetching and Criticality-Based Resource Allocation

Bogdan F. Romanescu, Michael E. Bauer, Daniel J. Sorin, and Sule Ozev
Department of Electrical and Computer Engineering
Duke University
{bfr2, meb26, sorin, sule}@ee.duke.edu

1 Introduction

A major problem facing the computer and semiconductor industries is the increasing amount of CMOS process variability [1, 3]. Variability in low-level circuit parameters, such as transistor gate length and gate oxide thickness, complicates system design by introducing uncertainty about how a fabricated system will perform. Although a circuit or chip is designed to run at a nominal clock frequency, the fabricated implementation may vary far from this expected performance.

We have developed architectural techniques for mitigating the impact of process variability. We make the following three contributions:

- We mitigate the impact of having some L1I and L1D cache frames that are slower than others, by using small L0I and L0D caches and prefetching.
- We reduce the impact of slow functional units by giving critical instructions priority for the fast functional units.
- We alleviate the impact of having slow access latencies for some registers in the register file, by renaming the destinations of critical instructions to fast registers.

All three of our approaches enable us to aggressively clock the processor with only minor degradation in IPC (instructions per cycle), thus achieving an overall performance improvement. Our work combines several ideas—prefetching, L0 caching, and criticality—that were previously developed for other purposes. *Our contributions are using and combining these ideas to overcome the effects of process variability.*

2 Variability in L1I & L1D Cache Latency

We use small L0I and L0D caches (e.g., 2-16 entries) to mitigate the effects of process variability. We deconfigure slow L1 frames and use the fast L0 frames for holding the data. By combining the fast L0 with prefetching, we overcome the loss of the slow L1 frames. We guarantee that the L0 caches have only fast frames, despite variability, by deconfiguring slow L0 frames.

3 Variability in Functional Unit Latency

We allow a functional unit that is substantially slower than other identical functional units to take an additional cycle. To do so, we must address two issues. First, we must avoid putting slow functional units on the critical path of a program's execution. Second, the instruction scheduler must accommodate variable latencies. Our scheme is *criticality-based functional unit allocation (CFUA)*. We borrow the previously developed instruction criticality predictor from Fields et al. [2] to identify critical instructions.

4 Variability in Register File Latency

Our approach, *criticality-based register allocation (CRA)*, steers critical instructions to the fast (1-cycle) registers. When a critical instruction reaches the Rename stage of the pipeline, its destination register is renamed to a free fast register, if one is available. CRA does not ensure that critical instructions will *read* from fast registers, because we do not control which registers are read. However, by having a critical instruction, *A*, write to a fast register, we enable instructions dependent on *A* to read *A*'s output from a fast register or the operand bypass network.

References

[1] S. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6):10–16, Nov/Dec 2005.

[2] B. Fields et al. Focusing Processor Policies via Critical-Path Prediction. In *Proc. 28th Int'l Symp. on Computer Architecture*, pages 74–85, July 2001.

[3] International Technology Roadmap for Semiconductors, 2003.