Benchmarking ISA Reliability to Intermittent Errors

Melina Demertzi, Bardia Zandian, Ricardo Rojas, Murali Annavaram
University of Southern California
{*demertzi, bzandian, ricardar, annavara*}*@usc.edu*

***Abstract*—Silicon scaling has led to accelerated wearout, which results in increased number of intermittent errors. Of the various factors that can lead to intermittent errors, device utilization is a major contributor to wearout. Every device on a chip is activated in response to either control signals or data movement resulting from instruction execution. This research proposes a systematic methodology for benchmarking the vulnerability of instruction set architecture (ISA) toward intermittent errors. By following each instruction during its execution through the processor pipeline, we quantify how many devices each instruction activates during its execution. We propose Vulnerability to Intermittent Failures (VIF) as a metric to quantify the stress imposed on circuits by an instruction. We show how VIF varies from instruction to instruction and how different inputs can affect VIF.**

I. Motivation

Prior studies have shown that almost all the phenomena causing intermittent failures, such as Time Dependent Dielectric Breakdown (TDDB) and Electromigration (EM) are caused by wearout of the gates due to extensive use. Gate usage is manifested in the form of gate toggles [4], [1]. Within a processor, gates are toggled by instruction execution. However, each instruction may stress different number of gates to achieve its functionality. Intuitively, a memory access instruction accesses the cache hierarchy and memory management unit. An ALU instruction may stress the ALUs within the execution stage of the pipeline. Hence, instructions within an ISA stress different sets of gates and, as a result, each instruction might create distinct effects on the overall system susceptibility to intermittent errors. Benchmarking the susceptibility of each instruction within an ISA to intermittent failures provides us with the ability to understand how software execution can cause wearout to the underlying hardware.

In this work, we achieve the benchmarking goal by measuring the number of gates sensitized by each instruction within an ISA. We use the term VIF (Vulnerability to Intermittent Failures) to measure the number of toggled gates by an instruction as a first order proxy for stress generated by that instruction. More precisely, VIF is a percentage that is defined as the number of toggled gates over the total number of gates in the circuit. VIF alone cannot measure the FIT (failures in a billion hours) of intermittent errors; it simply measures the stress factor. We then use various device-level models to translate the stress factor into FIT. Most models combine the VIF with operating conditions, such as temperature, to measure the timing degradation of a gate. For instance, VIF combined with temperature can be used for estimating timing degradation due EM using Black's equation [2].

In this study, we measure the VIF of SPARC V9 ISA. We follow a single instruction in the ISA as it traverses the pipeline of a SPARC T1 processor core [3] and compute the number of gates toggled during the execution. We use the OpenSPARC implementation of SPARC T1 processor for measuring the VIF. OpenSPARC RTL code is divided into 5 units: Instruction Fetch Unit (IFU), Execution Unit (EXU), Floating Point Frontend (FFU), Load/Store Unit (LSU), and Trap Logic Unit (TLU). For our study, we monitor four OpenSPARC RTL modules: IFU, EXU, FFU, and LSU. In other words, we track how each instruction moves through these four logic blocks and track how many gates each instruction sensitizes within each block.

Instruction selection within an ISA: We choose a wide and representative range of instructions so as to be able to cover the vast majority of the instructions likely to be executed during a typical program execution. We also include several implementation flavors of the same instruction in our experiments. These are listed on the x-axis of Figure 1.

Emulation Setup: We use a multi-step cross-layer experimental infrastructure to compute the number of gates toggled in the T1 processor by each benchmarked instruction when executed in isolation. The first input to our setup is the T1 RTL code. We then use the Synopsys gate-level dc compiler to generate the gate-level circuit implementation of each one of the four modules in the T1 processor using the IBM 90nm library. In the second step, the same SPARC T1 is augmented with information generated from the ChipScope Pro Integrated Logic Analyzer (ILA) and then it is mapped on a Xilinx ML509 FPGA evaluation platform. ILA allows us to probe the input signals to the four modules of interest within T1, while the one instruction we are benchmarking is executing on the FPGA platform. These inputs are then saved as value change dump (VCD) files. Each VCD file shows the list of input signals that would be entering the module when the instruction is about to execute in that module. In the last step of our experimental methodology, the synthesized modules from the first step are simulated using the VCD files as inputs. During gate-level simulation, a Switching Activity Interchange Format (SAIF) file, which counts the exact number of gate toggles in the circuit, is

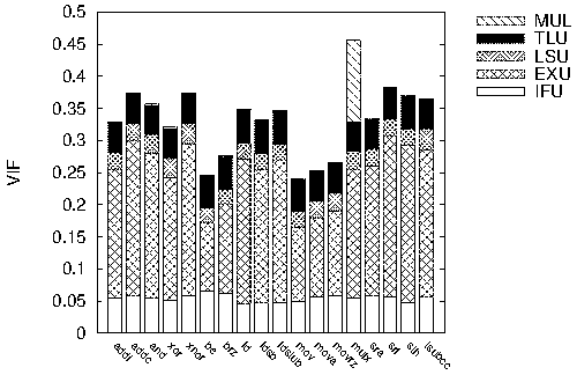

Figure 1. Stacked view of the toggled gates within the pipeline

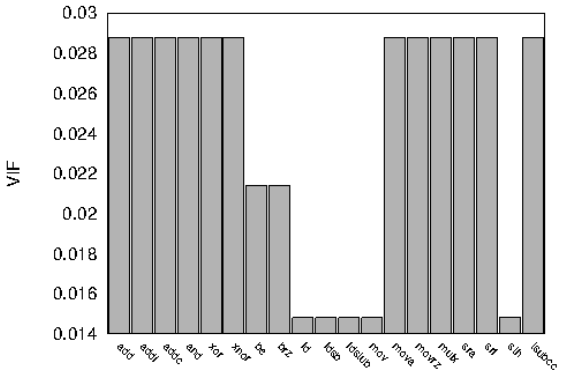

Figure 2. Percentage of toggled gates in ifu_ifqdp sub-module

generated.

II. Experimental Results

Figure 1 presents the number of gates toggled by each individual instruction when it traverses the pipeline as the percentage of gates in each pipeline stage over the total number of gates in the synthesized T1 processor. Some instructions, such as be and mov, toggle as few as 22% of the total gates in the T1 processor, while shift instructions such as sra and srl toggle about 40% of the total gates. We also observe that instructions within the same group, such as $addi$, $addc$, produce similar stress on the processor since they have similar gate toggle count distributions. The take-away point from this chart is that even using a simple in-order processor, there is nearly $2\times$ variation in the gates toggled by different instructions.

Sub-Module Variations: The variation of gate toggle count behavior becomes more prominent when we look at a finer granularity within each module. For example, IFU is stressed equally by all instructions. But within the IFU, the sub-module ifu_ifqdp, which implements the instruction fetch queue datapath, exhibits significant variations in the gate toggle count depending on the executed instruction. The VIF variation in the ifu_ifqdp module can be seen in Figure 2. We also observe that control instructions, such as branches, have lower toggle counts than the rest of the instructions. On the other hand, there are sub-modules that exhibit uniform gate toggle counts independent of instruction, such as ifu_swl which manages the processor threads and schedules the next thread for execution. There are modules within the processor, such as the exu_alu within EXU and the lsu_dcdp within the load store unit, which exhibit high utilization patterns regardless of the instruction being executed. These modules are prime candidates for being the first modules where wearout failures will exhibit themselves and should be considered for selective protection.

Input Dependence: Another factor that affects VIF is the input operand values for an instruction. For instance, an add instruction with simple inputs may not propagate any carry, while a different set of inputs may propagate the carry bit from the least to the most significant bit. In our simple in-order pipeline, we note that inputs cause gate toggle variations primarily in the execution stage of the pipeline. The impact of input operands on other pipeline stages, such as fetch and decode, are quite insignificant. Moreover, when we average the VIF values across the entire pipeline, we observe that VIF variations due to input values are reduced, as can be seen in Figure 3.

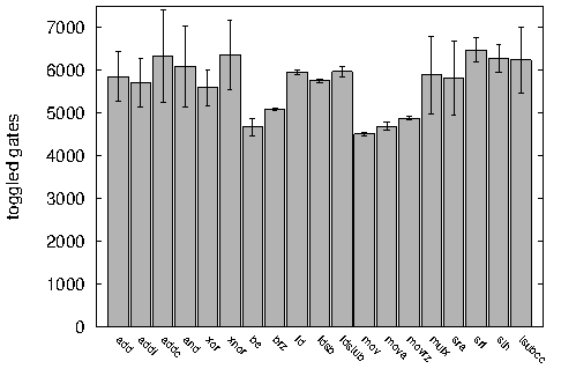

Figure 3. Effect of input values on toggled gates

References

[1] C. Basaran and S. Li. Electromigration time to failure. In *CGMS'10: Proceedings of Conference on Grand Challenges in Modeling & Simulation*, pages 72–79, 2010.

[2] J. Black. Electromigration - a brief survey and some recent results. In *IEEE Transactions on Electron Devices*, pages 338–347, 1967.

[3] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25:21–29, 2005.

[4] R. Vattikonda, W. Wang, and Y. Cao. Modeling and minimization of PMOS NBTI effect for robust nanometer design. In *DAC'06: Proceedings of the Design Automation Conference*, 2006.