A Case for Instruction Subset Architectures (I_SA): Guaranteeing Functionality in High Defect Rate Technologies

Hiren D. Patel and Siddharth Garg
University of Waterloo
{hdpatel, s6garg}@uwaterloo.ca

1 Motivation

As we move towards the end of the technology roadmap, and potentially to newer technology flavors besides conventional CMOS, transistor defect rates are expected to increase significantly. Till recently, industry has dealt with defects (including both manufacturing defects and defects at run-time) using *conservative* approaches - discarding non-functioning chips at test time and extensive guardbanding. With increasing defect rates, however, these conservative approaches will become untenable, resulting in prohibitive yield and performance loss.

This is the challenge that Instruction Subset Architectures (I_SA) try to address — how do we design and architect processors, either single- or multi-core, that *degrade gracefully* with increasing defect rates. More specifically, how do we guarantee functional correctness, possibly at the expense of a performance penalty, in a system where each core has one or more faulty transistors. The challenge of graceful degradation has been extensively addressed for the memory sub-system — as illustrated in Figure 1, spare rows/columns and error-coding techniques can be used in caches to protect can against a relatively large number of transistor failures. In the worst-case, faulty bit cells reduce capacity and impact performance, but do not impact functionality. However, the same cannot be said for the core logic. For the most part, extant techniques to deal with faulty transistors in processing cores involve fully disabling cores. With increased defect rates, however, the likelihood that each core has at least one faulty transistor will grow to a point where disabling cores will result in unacceptable yield loss.

Any fault-tolerant design methodology depends on an assumed *fault model*. However, existing fault models focus on the transistor- or gate-level abstractions, which we find are inadequate to meet the challenges in designing gracefully degrading processors. Instead, in this paper, we make a **case for the use of instruction-set level fault models that make the impact of transistor faults explicit at higher levels of abstraction**. As a first step in this direction, we propose a simple yet powerful high-level fault model to aid in the design of gracefully degrading processors, which we call the *instruction subset fault (I_S) model*. The I_S fault model sets the stage for the design of *Instruction Subset Architectures* (I_SA) which represent, in our view, a broad set of open challenges spanning all stages of the design process, from circuit design to micro-architecture, compilers and operating systems.

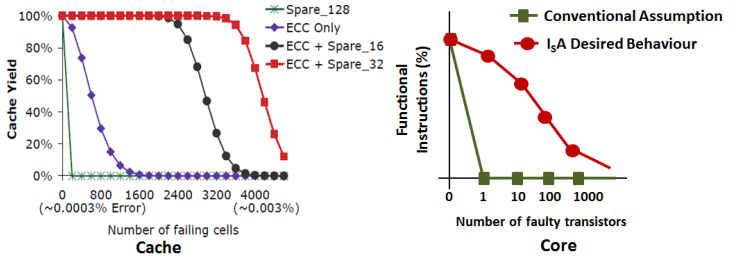

Figure 1: Cache yield degrades gracefully with increasing bit faults [1]. Analogously, the instruction subset (I_S) fault model identifies the percentage of functioning instructions in the ISA with increasing defects instead of assuming that every transistor in the core logic is a single point of failure.

2 Instruction Subset Fault Model

The I_S model is a simple, yet expressive fault model for processing cores. This fault model assumes that each core can reliably execute only a *subset* of instructions from the ISA that are designed to execute correctly. This subset of fault-free instructions can vary with time due to intermittent faults and, furthermore, each core in a multi-core system could potentially have a different subset of fault-free instructions because of the random nature of manufacturing defects.

The I_S model reflects our belief that in high-defect rate technologies, the contract between the software and hardware (previously the instruction set) must, out of necessity, be appended with some fine-print. While the I_S fault model is only a suggested starting point, more expressive models that place constraints on temporal execution of instructions or data ranges for operands might provide additional benefits.

This brings us to the idea of an Instruction Subset Architecture (I_SA) — **we define an I_SA to be a single- or multi-core processor architecture that provides the user with**

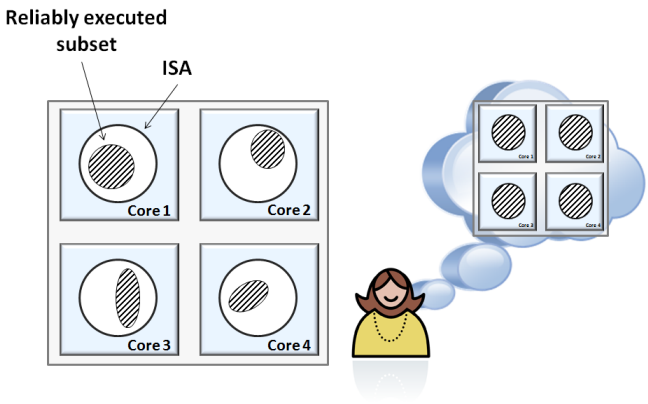

Figure 2: A multi-core processor with four cores. As shown, each core can only reliably execute a subset of the ISA. The goal of an I_SA architecture is that from a user's perspective, each core can functionally execute the entire ISA.

a full-range of functionality in the presence of an I_S fault model, as illustrated in Figure 2. We believe that this challenge can only be met by re-thinking all levels of the design process — both bottom-up (digital circuits, computer-aided design and test methodologies) and top-down (operating systems, compilers and micro-architecture).

Before we outline the new design challenges, it is instructive to examine current processor architectures in the presence of the I_S fault model. As mentioned before, transistors in the core logic are conventionally considered single points of failure, i.e., a single fault disables the entire core. This is, however, clearly not the case for a number of transistors in the datapath. Faults in the *execute* stage are a good example — a fault in an integer multiply unit only disables instructions that make use of that unit. Perhaps less obvious is the fact that faults in the *decode* stage are not necessarily catastrophic either — a stuck-at fault in an op-code register will only affect half the instructions (less if some op-codes contain don't cares) while a fault in an operand register would only affect instructions that use both operands. Additionally, not all control logic faults are single points of failure, for example, faults in the *instruction fetch* stage may only disrupt conditional or unconditional jump instructions (imagine a fault in the multiplexer that controls the program counter in instruction fetch).

There are, of course, numerous potential single points of failure in most modern process architectures. However, we have not yet seen any experimental studies that map transistor or gate faults to instruction set faults although this seems to be an almost obvious question. The fact that the I_S model forces us to ask this and, as we will see, other interesting questions, is in our opinion, reflective of the benefits of migrating fault models to higher levels of design abstraction.

3 Instruction Subset Architectures

The design of I_SAs, i.e., architectures that guarantee a full range of functionality in the presence of a subset of faulty instructions with acceptable performance degradation, raises a range of interesting and previously unaddressed challenges, some of which we will now highlight.

C1: I_S aware design methodologies and CAD tools. Conventional design methodologies and CAD tools assume that, from a functionality perspective, *all logic gates are equal*, since gates are assumed to be single points of failure. From the perspective of an I_S fault model, however, this is not the case. Some gates cause more instructions to be faulty than others, and it is therefore important to incorporate this information in all stages of design — this represents in our view, a fundamental change in CAD methodologies starting from logic synthesis to physical design and verification tools.

C2: Test methodologies for the I_S fault model. How can we efficiently figure out which instructions are faulty and which are not on any given core? Existing test infrastructure focuses on identifying faulty gates, not faulty instructions. Run-time testing, either via dedicated built-in self test hardware or software (or both) for the I_S fault model is another critical challenge.

C3: Micro-architecture, compiler and operating system techniques for I_SA design. C1 and C2, if successfully addressed, will result in cores in which the number of working instructions degrades gracefully with increasing failure rates and test methodologies accurately identify the faulty instructions. The critical top-down challenge is to devise new micro-architecture, compilers and operating systems based solutions that provide the user with full functionality without significantly sacrificing performance. Potential solutions include emulating faulty instruction using run-time recompilation techniques, executing faulty instructions on neighboring cores that do not have that instruction in their non-functioning set, using virtualization techniques to mask the impact of faulty instructions from the user, guaranteeing a basic minimal set of functioning instructions via overdesign where the minimal subset is proven to capable of emulating *any* other instruction (think ultra-reduced instruction set computing [2]).

References

[1] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. Multi-bit error tolerant caches using two-dimensional error coding. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007.

[2] F. Mavaddat and B. Parhami. URISC: the ultimate reduced instruction set computer. *International Journal of Electrical Engineering Education*, 1988.