

High Performance Computing on Fault-Prone Nanotechnologies: Novel Microarchitecture Techniques Exploiting Reliability-Delay Trade-offs

Andrey V. Zykov, Elias Mizan, Margarida F. Jacome, Gustavo de Veciana, Ajay Subramanian
Department of Electrical and Computer Engineering
The University of Texas at Austin
Computer Engineering Research Center Technical Report UT-CERC-TR-MJ-0502

Abstract—Device and interconnect fabrics at the nanoscale will have a density of permanent faults and susceptibility to transient faults far exceeding those of current silicon technologies. These aggressive fault regimes appear to be intrinsic to nanoscale devices and will thus fundamentally impact design principles. In this paper we introduce a new performance optimization dimension at the microarchitecture level which can mitigate overheads introduced by fault tolerance. This is achieved by directly exposing reliability versus delay design trade-offs while incorporating novel forms of speculation which use faster but less reliable versions of a microarchitecture’s performance critical components. To explore these trade-offs, we propose a rough parametric model for the relative scaling of delay with reliability of hypothetical nanoelectronic components. Based on a parameterized microarchitecture, we exhibit the benefits of optimizing these trade-offs. We present extensive simulation results for different possible delay-reliability scalings, and show these techniques can provide substantial improvement in performance (IPC increases by factors of 2.3 – 18.1) and/or improved resource efficiency with reduced complexity.

I. INTRODUCTION

Recent striking successes in devising and assembling nanoelectronic devices suggest that the ability to build large scale nanofabrics for computation is now on the 10–15 year horizon [1], [2], [3], [4]. Nanotechnologies based on carbon nanotubes and silicon nanowires are particularly promising. Nanotube switches can theoretically operate at unprecedented speeds, e.g., 100–200GHz, while nanowire junction arrays can be configured as OR, AND, and NOR logic gates, with gain, and thus be used to realize basic computation[2]. Although many challenges lie ahead, many predict that it will be possible to assemble workable computer memory and logic devices from nanoscale building blocks before silicon devices hit their limits[1]. As such, it is critical to start investigating the design methods and computing system architectures required to take these technologies into design/production environments [5], [6].

Irrespective of the ‘winning’ (charge carrier transport-based) nanotechnologies, it is widely recognized that devices and interconnects at the nanoscale will exhibit fault densities much greater than state-of-the-art silicon technology. Indeed, they will have: (1) a density of defects which is much higher than current silicon technologies [1], [7]; and (2) are likely to be much more susceptible to transient faults (soft errors), see e.g. [1]. These increases are, in part, due to the physical dimensions being considered. From a materials perspective, decreasing the size of structures increases the ratio of surface area to volume, making imperfections in materials interfaces more critical to the proper function of interconnects and devices. Furthermore, at such reduced scales, the discrete nature of atomic matter and charge becomes significant. Namely, a single charge or defect may significantly impact the structural stability of a nanodevice, as well as its timing/performance characteristics and sensitivity to fluctuations in the local electrostatic environment (electric noise). These observations point to a reliability problem which is intrinsic to nanoscale regimes and is thus here to stay.

Contributions of this paper. Concerns with the projected increases in transient faults motivated a substantial amount of recent research on microarchitectural approaches to realize error detection and recovery [8], [9], [10], [11]. Yet, as discussed in the sequel, much of this work would fall short for highly fault prone nanoelectronics. In this paper we formalize the problem of exploring *reliability–delay* trade-offs at the microarchitecture level, for performance enhancement. This is achieved by a novel form of *reliability-driven* speculation relying on faster but less reliable versions of a microarchitecture’s performance critical components or stages. In order to explore these trade-offs, we propose a parametric model for relative scalings of delay versus reliability on hypothetical nanoelectronic components. Then, based on a simple parameterized microarchitecture, we exhibit the benefits of *explicitly* exploring this novel class of trade-offs at the microarchitectural level.

II. PARAMETRIC MODEL FOR DELAY VERSUS RELIABILITY SCALING IN NANO-ELECTRONIC COMPONENTS

Substantial progress has been made towards devising basic building blocks for nanoscale electronics based on carbon nanotubes and silicon nanowires. For example, [5] and [6] have proposed basic architectures for realizing ‘universal’ computation with all logic and signal restoration operating at the nanoscale. The proposed architectural styles are based entirely on large arrays of crossed nanowires and/or nanotubes, forming programmable fabrics which can be configured to realize any logical function. When assembled into arrays, some of the constituent nanoscale wires/tubes will have poor or non-existent contacts, and individual switches may be defective. To tolerate these defects, both local wire sparing and array sparing are introduced. Such built-in defect tolerance will incur delay penalties which increase with the expected density of hard faults [12]. Thus, in general, defect tolerance should be viewed as introducing a base delay overhead.

Designers also need to worry about transient faults. A natural approach to increasing the reliability of systems prone to a substantial rate of transient faults is to make use of redundant resources and arbitration. However this has a cost; in particular, higher delays should be expected due to the longer interconnects required to leverage and arbitrate such redundancy. The existence of such trade-offs is central to our work and thus we begin by proposing a crude, but reasonable, delay-reliability model. We do not claim absolute fidelity for our model, as we will abstract details of the technologies and future design methodologies. Our aim is to obtain a reasonable parametric model to capture *relative scalings* for delay of a component versus its reliability. In this paper the reliability of a component is the probability it performs its function correctly on a given use.

Increasing reliability via resource redundancy. As mentioned above, one can achieve high reliability by using n redundant (unreliable) units and arbitrating their outputs, i.e., checking that at least $n/2 + 1$ units agree. Under this arrangement as n increases the probability of failure decreases exponentially. Two problems arise in this context: (1) the complexity of realizing ideal arbitration over n outputs; and (2) the reliability of the arbitration circuit itself. Let us discuss these briefly.

Devising an ideal arbiter over n inputs requires on the order of $\Theta(n \log(n))$ gates and may result in a structure that is quite sensitive to failures (by ideal we mean a full n -way arbitration circuit). An alternative is to construct a non-ideal arbitration circuit based on a tree of *fixed* sized small arbiters, e.g., $\log_3(n)$ levels of three-way majority vote arbitration. In this case only $\Theta(n)$ gates would be required but this comes at some loss in overall reliability versus ideal arbitration. One can show that the reliability $r(n)$ at the output of such arbitration trees would improve quickly in the number of resources n , in fact it behaves roughly as

$$r(n) \approx 1 - \exp[-cn^{1/n}]$$

where c depends on the reliability of a given unit and the size of arbiters used in the tree, and units are assumed to fail independently. Thus the number n of redundant resources required to achieve a target reliability $(1 - \rho)$ is roughly

$$n \approx \left[\frac{\ln(\rho)}{c} \right]^{\gamma_1} \quad (1)$$

where $\gamma_1 \in (1, 1.46)$ depends on the size of the arbiters used to construct the tree, e.g., $\gamma_1 = \ln(3)/\ln(2) = 1.46$ for three-way majority voters and decreases to 1 as more complex arbiters are used, e.g. five-way majority voters. This corresponds to at least a logarithmic growth in the required probability of error ρ .

The above assumed the arbiters were themselves reliable. Unfortunately this need not be the case, and the reliability of the arbiters eventually limits the gains one can obtain through increased redundancy. One might however expect circuits performing simple arbitration functions, e.g., three way arbiter units, can be made selectively more reliable. Below we assume the conservative scaling captured by (1) applies.

Increased redundancy leads to increased delay. Increasing the number of redundant resources leads to increases in area, and thus longer interconnect lines, which in turn increase the expected delays associated with such components. Consider an unreliable unit realized on a square region with area A . Then, neglecting increases in area associated with additional arbitration circuits and interconnect lines, the combined area of n resources should be at least nA . In order to profit from arbitration across redundant resources, lines will need to bring outputs from various resources together. Depending on the efficiency of the architectural design style, the worst case length of these lines $l(n)$ will grow with n . This growth might be proportional to \sqrt{nA} for efficient layouts, or $n\sqrt{A}$, for poor layouts. In practice these are likely to grow even faster in n . The delay associated with such redundant arrangements will depend on devices, number of arbitration levels and particularly on the length of long interconnect lines. We expect the dominant factor to be interconnect length, with delay growth somewhere between linear and quadratic¹. Thus we propose the following scaling relationship for delay as the number of resources grow

$$d(n) = \delta + \beta l(n)^{\gamma_2} = \delta + \beta n^{\gamma_2 \gamma_3} \quad (2)$$

where δ and β are constants and $\gamma_2 \in (1, 2)$ captures the growth delay as a function of interconnect length and $\gamma_3 \in (0.5, 1)$ captures a range of possible growth characteristics for interconnect length with the number of redundant units. Combining (1) with (2) we have the following relationship between delay and reliability

$$d(\rho) = \delta + \beta \left[\frac{\ln(\rho)}{c} \right]^\gamma \quad (3)$$

where $\gamma = \gamma_1 \gamma_2 \gamma_3 \in (0.5, 3)$. In the sequel, we vary γ within this range of values to capture different technology scalings.

Modeling relative delay–reliability trade-offs. Our primary goal is to evaluate trade-offs that can be achieved for microarchitectural components. To this end, we need only model *relative* delays associated with resources achieving different reliabilities, say $(1 - \alpha)$ and $(1 - \rho)$. In particular, based on (3) we can scale the relative delays for FUs achieving different reliabilities as follows

$$\frac{d(\alpha) - \delta}{d(\rho) - \delta} = \left(\frac{\ln(\alpha)}{\ln(\rho)} \right)^\gamma. \quad (4)$$

In summary, we have motivated a scaling model that roughly captures how delays might grow with the required reliability of a given component. The parameter δ captures the intrinsic complexity of the function performed by the component, while γ captures a range of possible technology and design characteristics that might be associated with future nanosystems. Although delays grow logarithmically in the probability of failure, see (2), the differences can be quite marked. For example, based on (4), if $\rho = 10^{-6}$, $\alpha = 10^{-3}$, $\gamma = 2$ then $d(\rho) - \delta = 4(d(\alpha) - \delta)$ i.e., the excess delay increases by a factor of 4.

III. EXPLOITING RELIABILITY-DELAY TRADE-OFFS AT THE MICROARCHITECTURE LEVEL

The basic considerations underlying our parametric delay model suggest that highly fault-tolerant microarchitecture component designs for nanotechnologies will incur substantial delay overheads. Here then lies the fundamental question addressed in this paper: can one effectively ‘hide’ the performance overheads incurred by fault-tolerant component designs, at the microarchitecture level? We will show that this is indeed the case.

A new set of tradeoffs at the microarchitecture level. The key novelty of our work is the introduction of a new *performance optimization*

¹Quadratic growth may result when interconnects become too long to be reasonably considered reliable. When this is the case, one can show the need to place a quadratic number of arbiters/buffers across a multiplicity of lines in order to maintain a fixed high reliability per line. These buffers would contribute to the delay associated with the line

dimension in microarchitectural design. This is achieved by exposing reliability-delay trade-offs through novel forms of *speculation* relying on faster but less reliable versions of a microarchitecture’s performance critical components. Not unlike other microarchitecture-level design parameters, architects will need to perform design space exploration to identify the most favorable reliability-delay tradeoff for each ‘speculative’ component. We shall call this broad class of techniques *reliability-driven speculation* and microarchitectures enhanced with such features *reliability-aware* (RA) microarchitectures.

Selection of baseline microarchitecture. The principle of reliability-driven speculation, and associated performance optimization, can be applied to essentially any architecture, including EPIC/VLIW, dataflow, etc. In this paper, we will demonstrate its impact on a well known class of machines – out-of-order (OOO) superscalar processors. We augment this machine to expose performance-critical reliability–delay tradeoffs, and compare its performance to that of the baseline machine. Given the substantial body of recent work on fault-tolerant microarchitectural techniques targeted at OOO superscalar processors, this provides a good context in which to highlight the novelty of our approach. We revisit this topic in Section VIII.

Where can/should one apply reliability-driven speculation? The performance of an OOO superscalar machine is critically impacted by several factors, including the speed at which data dependencies between instructions can be resolved within the processing core, the ability to quickly access data from memory, and the ability to deliver a steady stream of instructions to the processing core. Thus, the machine’s processing core, front-end (i.e., fetch engine), and memory subsystem are all excellent candidates on which to apply reliability-driven speculation. In this paper, we will focus on applying these ideas on a machine’s processing core. Specifically, we show that reliability-driven speculation can be used to increase the ‘speed’ at which data dependences between instructions are resolved, including the latency incurred in executing producer instructions/operations and that incurred in transferring, ‘on-the-fly’, such results to consumer instructions/operations. In order to eliminate effects caused by extraneous factors and sharpen our analysis, the front-end and the memory subsystem for both the baseline and the reliability-aware machines are assumed to be ‘perfect’ in our experiments.

Optimistic and pessimistic assumptions on technology scaling. We consider two scaling regimes for the target nanotechnologies: ‘*optimistic*’ and ‘*pessimistic*.’ Under the optimistic scaling only the latency of complex transformational components – that is, functional units – will scale aggressively with reliability, i.e. with the level of fault tolerance of a design. Under the pessimistic scaling, all pipeline stages will scale similarly with reliability. The latter is likely to be overly conservative for non-transformational phases, i.e., the actual delays are likely to be between these two limits. In the following subsections we present reliability-driven speculation techniques suitable for each such scaling scenarios.

A. Reliability-driven data speculation

Under the optimistic scaling scenario, reliability-delay tradeoffs can be exposed by using two different fault-tolerant designs for the microarchitecture’s functional units (FUs) – one with high reliability $(1 - \rho)$, say $\rho = 10^{-20}$, and another with lower reliability $(1 - \alpha)$, say $\alpha = 10^{-4}$, where ρ and α denote the corresponding probabilities of failure. From now on we refer to components with reliability $(1 - \alpha)$ as *unreliable*, and components with reliability $(1 - \rho)$ as *reliable*. As suggested in Section II, unreliable FUs would also have a reduced latency relative to more reliable counterparts. To exploit this, one can design a processing core which speculatively executes instructions on faster unreliable FUs and subsequently validates these results on reliable FUs. Such a core would quickly generate results for consumer instructions yet, upon detecting an error, would pay a roll-back penalty to restore reliable state and subsequently refill the pipeline. This form of reliability-driven *data* speculation has the potential to enhance performance relative to a standard/baseline core which purely executes instructions on reliable FUs – yet, the enhancement depends critically on the selection of *design parameter* α (see, e.g. Figure 3).

Reliability-aware microarchitecture: the α and ρ pipelines We implemented the reliability-driven *data* speculation introduced above on a

two-pipeline microarchitecture comprising a ‘speculative’ (or α) pipeline and a ‘validation’ (or ρ) pipeline, see Figure 1. The FUs instantiated in the execute phase of the ρ pipeline are reliable while their counterparts in the α pipeline have lower reliability – namely, $(1 - \alpha)$, where α is a *design parameter*. The ρ pipeline is much simpler than the α pipeline since, when instructions enter the ρ pipeline, all of their data dependencies are guaranteed to have been resolved in the α pipeline, and thus they can be immediately issued on an available reliable FU, as in [13]. To ensure the ρ pipeline is not a bottleneck, the number of reliable FUs available to it should exceed the number of unreliable FUs instantiated in the α pipeline by a factor which is roughly the ratio of the latency of a reliable FU to that of its unreliable counterpart. Indeed, this would permit overlapping of as many instruction validations in the ρ pipeline (irrespective of possible data dependencies among them) as necessary to sustain the throughput of the α pipeline. In fact, with such dimensioning of FU resources it can be shown that, when no faults are generated in the α pipeline, the IPC of our two-pipeline processing core is bound by the latency of its *speculative execute stage* (i.e., the smaller latencies associated with the FUs in the α pipeline), hiding those of the validation pipeline. However, if an incorrect value is detected during validation, a performance penalty is incurred, which reflects the cost of flushing both pipelines, rolling back to a reliable state, and then refilling both pipelines. Thus, as the design parameter α increases, i.e., FUs in the α pipeline are made faster but less reliable, the error-free performance of the core would improve, but the number of ‘miss-speculations’ would increase, eventually compromising the actual performance of the core. This is the key trade-off exposed by our reliability-aware (RA) microarchitecture.

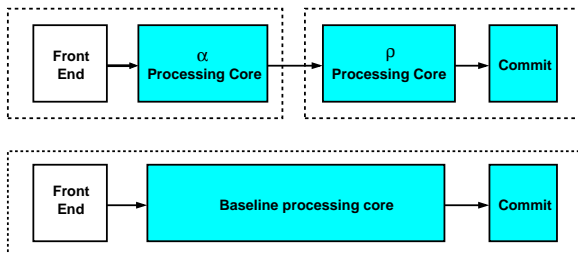


Fig. 1. Top-level pipeline organization for RA and baseline machines

Additional details on the RA microarchitecture. The dynamic speculative state of the machine is maintained on a register file (α -RF) which is only accessed by instructions in the α pipeline. After an instruction completes execution in the α pipeline, it is forwarded in out-of-order manner to the ρ pipeline, for validation. The architected register file (ρ -RF) is updated in-order, after an instruction has been validated. Stores are sent to memory in the ρ pipeline, while loads are issued in the α pipeline. Rollback starts when a faulty instruction is the next one to commit its results.

B. Other forms of reliability driven speculation

Under the ‘pessimistic’ scaling assumption, the latency of *all pipeline phases* will scale similarly. Thus, improving the ‘speed’ at which the processing core resolves data dependencies requires exploiting additional reliability-delay trade-offs on pipeline stages that *critically impact such latencies*. Namely the *issue* and *write-back* stages of the α pipeline are now also delay critical. For simplicity, we refer to the design parameters controlling the reliability-delay tradeoffs for these stages as α , yet, clearly they can be optimized independently. Let us consider first the $(1 - \alpha)$ reliable issue stage in the α pipeline. Load memory accesses are now first issued with $(1 - \alpha)$ reliability in the α pipeline, and then validated in the ρ pipeline. We consider three classes of possible errors when instructions are dispatched from reservation stations to functional units: (1) instruction is never sent to a functional unit; (2) instruction is sent to the wrong functional unit; and (3) instruction is sent with incorrect operands. Errors of Type 2 and 3 can be only detected when the instruction is re-executed in the ρ pipeline, while errors of Type 1 can be detected locally, e.g., using reliable watchdog timers. Thus, recovery from Type 2 and 3 errors incurs a performance penalty which is much more severe than that for Type 1 errors². Similarly, we consider three basic types of errors for the

‘unreliable’ writeback phase: (4) data corruption; (5) a wake-up of the wrong dependent instruction; and (6) deadlock – the result of the computation is never sent to its producer and/or consumer instructions. Errors of Type 4 are handled in the ρ pipeline, while errors Type 5 and 6 can be detected locally, using reliable watchdog timers. Thus, in this case recovery from a Type 4 error incurs a performance penalty that is much more severe than those associated with Type 5 and 6 errors.

IV. EXPERIMENTAL METHODOLOGY

A. Modeling different hypothetical nanotechnologies

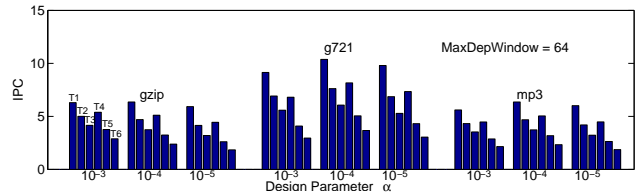


Fig. 2. Design space exploration for selection of design parameter α .

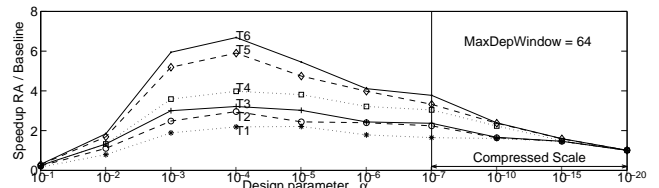


Fig. 3. Speedups for RA / Baseline for different α (mp3 benchmark).

In the model presented in Section II, the parameters γ, δ capture the relative delay overheads as reliability increases. We shall introduce an additional parameter $k \in (0.3, 0.95)$ to capture the *relative* contribution to delay of redundancy versus the intrinsic delay associated with a component’s function. The parameter k is specified with respect to the delay $d(\alpha^*)$ of each functional unit type, where $\alpha^* = 0.1$. Specifically $kd(\alpha^*)$ is the delay contribution associated with computation, while $(1 - k)d(\alpha^*)$ is the contribution associated with redundancy – the latter is also given by $d(\alpha^*) - \delta$, so $\delta = kd(\alpha^*)$. As the value of k gets smaller, one captures FUs with higher intrinsic computation complexity.³ We captured differences in latency among functional unit types, by setting those specified in Simplescalar [14] to be those achieved when $\alpha^* = 0.1$. With this as a starting point, for each α, ρ and γ, k we can compute scaled latencies using equation (4). Then, to obtain FU latencies in terms of cycles, we normalized all latencies by that of the minimum latency FU, i.e., the integer ALU, and rounded up.

Our purpose is to examine a range of possible technology scalings, based on varying two parameters, γ and k . A total of six possible technology scalings T1 – T6 were examined where $\gamma \in \{1.0, 1.4\}$ and $k \in \{0.90, 0.75, 0.60\}$. Latencies for integer and floating point FUs for various choices of reliability corresponding to different values of α and a required acceptable reliability of $\rho = 10^{-20}$, were determined for use in our experiments. Table I shows a relevant subset of these values. Latencies, measured in cycles, are presented for FUs with a probability of failure ρ and α – the α delays are in parenthesis, and are always 1 for the integer ALU, because it defines the cycle time. The rightmost column in Table I exhibits the relative cycle delays among the technology scalings being considered, i.e., the ratio of the delay of a cycle under a given scaling relative to the fastest considered.

B. Identifying suitable reliability-delay trade-offs for ‘maximum’ performance enhancement

As discussed above, the actual performance enhancement achieved through reliability-driven speculation depends critically on the selection of the *design parameters* capturing the reliability-delay trade-offs exposed by the microarchitecture. Since we are using a rough parametric model for the relative scaling of delay with reliability, during design space exploration we only consider variations in α within an order of magnitude. Our aim is to illustrate the associated tradeoffs at work, and how one may select ‘good’ values for such parameters, using representative workloads.

²We did not explicitly consider other classes of *locally* detectable errors, e.g., deadlock of the phase, since their impact on performance would be essentially ‘negligible’ relative to the more severe Type 2 and 3 errors.

³The k value reported in Table I corresponds to integer ALUs. For more complex FUs, k is scaled appropriately.

| Design Parameter α | Technology parameters | | ID | FU latencies in cycles | | | | Relative cycle delay |
|---------------------------|-----------------------|------|----|------------------------|---------------|--------|---------------|----------------------|
| | γ | k | | $\rho(\alpha)$ | | | | |
| | | | | int ALU | int Mul/Div | fp ALU | fp Mul/Div | |
| 10^{-4} | 1.0 | 0.9 | T1 | 3 (1) | 8/54 (4/22) | 5 (3) | 10/32 (5/13) | 1.00 |
| | | 0.75 | T2 | 4 (1) | 11/79 (4/22) | 7 (3) | 15/46 (5/13) | 1.35 |
| | | 0.6 | T3 | 4 (1) | 13/91 (4/22) | 9 (3) | 17/53 (5/13) | 1.69 |
| | 1.4 | 0.9 | T4 | 5 (1) | 16/123 (4/23) | 11 (3) | 22/71 (5/14) | 1.24 |
| | | 0.75 | T5 | 8 (1) | 24/175 (4/23) | 15 (3) | 32/102 (5/14) | 1.95 |
| | | 0.6 | T6 | 9 (1) | 27/193 (4/23) | 18 (3) | 36/113 (5/14) | 2.66 |

TABLE I
TECHNOLOGY PARAMETERS AND CORRESPONDING FU LATENCIES FOR RELIABILITY $1-\rho$ AND $(1-\alpha)$.

| | Baseline optimistic | Baseline pessimistic | RA optimistic | RA pessimistic |
|--------------------|---|--|--|---|
| Fetch engine | 'Perfect': instructions are fetched at very high throughput, i.e. no I-Cache / I-TLB misses, 100% branch prediction accuracy and no packet breaks at taken branches | | | |
| Memory | 'Perfect': D-Cache / D-TLB accesses always hit | | | |
| MaxDep Window | Implemented with a counter that keeps track of instructions waiting for data dependences. When counter reaches predefined threshold, decoding stops. We run simulations for thresholds 16,64,256. | | | |
| Pipeline resources | Number of functional units and memory ports as well as size of instruction window, queues and reservation stations is set for each machine as discussed in Section IV-C. | | | |
| FU delay model | FU Delays are set for reliability $(1-\rho)$ | | α pipeline FU delays are set for reliability $(1-\alpha)$ ρ pipeline FU delays are set for reliability $(1-\rho)$ | |
| Fault injection | No Faults | | Faults injected every cycle. Instructions issued to faulty FUs are tagged wrong. | Same as optimistic for FUs. Instructions fail at issue and writeback with probability α . |
| Error handling | No Faults | | Faults detected in-order. All subsequent instructions are squashed. Rollback completes in 1 cycle. | Same as optimistic + $r\%$ of issue/writeback faults are detected by a watchdog timer, $(100-r)\%$ cause a rollback. Faults detected by a watchdog timer cause a pipeline stall with latency equal to that of a reliable integer ALU. |
| Load/Store model | Single cycle latency | Latency equal to reliable integer ALU | Loads issued in α pipeline, stores issued in ρ pipeline. Latencies identical to baseline. | Loads issued in α pipeline with reliability $(1-\alpha)$. Stores issued in ρ pipeline. |
| Pipeline model | 5-stage OOO superscalar | Same as optimistic with scaled pipeline latencies. | 7-stage OOO superscalar. 2-stage frontend, 3-stage α pipeline, 2-stage ρ pipeline. | Same as optimistic with scaled pipeline latencies except for α -issue/ α -writeback |

TABLE II
SIMULATOR MODIFICATIONS AND EXTENSIONS FOR BASELINE AND RA MACHINES FOR BOTH OPTIMISTIC AND PESSIMISTIC SCALING ASSUMPTIONS

Figure 2 shows a sample of our design space exploration results for the reliability-aware (RA) machine described in Section III-A. Design parameter α (shown on the x -axis) defines the target reliability $(1-\alpha)$ for the 'speculative' FUs instantiated in the α pipeline. For the three benchmarks shown, $\alpha = 10^{-4}$ delivers the best performance (measured in instructions per cycle) across all hypothetical nanotechnologies T1-T6, (*MaxDepWindow* defines the actual machine configurations used in these experiments, see Section IV-C for details). Figure 3 shows the *speedup* achieved by our reliability-aware machine over a baseline machine (i.e., a 'traditional' OOO core), for the *mp3* benchmark, with design parameter α varying from 10^{-1} to 10^{-20} . As seen, when α is set to 10^{-4} , the speedup is substantial, and is more pronounced for technologies with more aggressive delay scalings, since the ability to 'hide' delay overheads becomes more critical in those cases. More importantly, this figure clearly shows the new optimization dimension exposed via design parameter α . Through exhaustive design space exploration, we found 10^{-4} to be the best overall choice for the various trade-offs exposed by the RA microarchitecture

C. Design of experiments

Experiment 1: Assessing relative performance of reliability aware versus baseline machines. In order to assess the performance enhancement potential of reliability-driven speculation, we consider an abstract 'single-bottleneck' processing core, i.e., a hypothetical core whose performance is limited by the capacity of a single 'abstract resource'. Specifically, the performance bottleneck is the maximum number of instructions waiting on data dependencies that can be queued in the processing core, denoted *MaxDepWindow*. We chose this bottleneck because it is representative (even if abstractly) of the microarchitectural components that dominate overall cost and complexity. In the sequel, we will set several values for *MaxDepWindow*, namely, 16, 64 and 256, and configure the RA and baseline machines so that all other resources appear to be unlimited, i.e., will not slow down execution, for any of the benchmarks. This is a *realization* of a 'single-bottleneck' machine. We then determine the IPC achieved for such RA and baseline machines, for a number of representative benchmarks.

Experiment 2: Assessing relative efficiency/scalability of reliability aware versus baseline machines. Experiment 2 focuses on actual scal-

ability issues, namely, on assessing how the two machines *scale* with different levels of target performance. In this set of experiments, we contrast baseline and RA machines that deliver the *same* performance (IPC), for a common workload. To this end, we start by considering each 'pair' of baseline and RA single-bottleneck machines obtained in our first set of experiments, for a specific *MaxDepWindow* value and technology. Since the RA machine is always faster than its baseline counterpart, for each benchmark we let the IPC delivered by the baseline machine be the 'target performance', and then decrease the number of integer ALUs on the α pipeline of the RA machine until it matches the IPC of the baseline. Then, for these pairs, i.e., baseline and 'reduced' RA machines, we report the number of integer ALUs used in the α pipeline of the RA machine and utilization statistics (average + std. deviation) for integer ALUs in the ρ pipeline of the RA machine, and integer ALUs in the baseline machine. To summarize, our second set of experiments introduces a new bottleneck on the RA machine – the number of integer ALUs instantiated in the α pipeline. Since a substantial percentage of instructions in our benchmarks execute on integer ALUs (95-98.3% for integer and 73.1-87.3% for FP applications), comparing the number of integer ALUs in each of the machines is a good indicator of their relative 'efficiency' in delivering the particular target performance.

D. Experimental Setup

The simulators used in our experiments are derived from the Simplescalar v3.0 simulator infrastructure [14], targeted at the PISA instruction set. The main modifications and extensions to the simulator are summarized in Table II. We selected 10 different benchmarks from the SPEC CPU2000 and Mediabench suites, including integer (gzip, parser, mpeg2, jpeg, bzip2, g721) and FP (ammp, mp3, quake) applications, with different degrees of ILP. All benchmarks were compiled with the -O2 -funroll-loop flags and 10M instructions were executed and analyzed for each one of them, after forwarding a sufficiently large number of instructions.

V. EXPERIMENTAL RESULTS FOR MACHINES WITH OPTIMISTIC DELAY SCALING

A. Results of experiment 1 for optimistic scaling

Figure 4 exhibits the instructions per cycle (IPC) for the baseline machine, the RA machine, and the RA machine when no faults are injected in the α pipeline. The latter case, denoted ‘no-rollback’ in the figures, was simulated to assess the penalties incurred in the actual RA machine due to faulty operation of unreliable FUs and stages in the α pipeline. The corresponding IPC values are shown as three superimposed bars, in 10 groups of six corresponding to the 10 benchmarks and six technology scalings, T1–T6 from left to right. These results correspond to $\alpha = 10^{-4}$, with each graph being associated with a different constraint on the maximum number of instructions waiting on data dependencies supported by the core, *MaxDepWindow*, 16, 64 and 256.

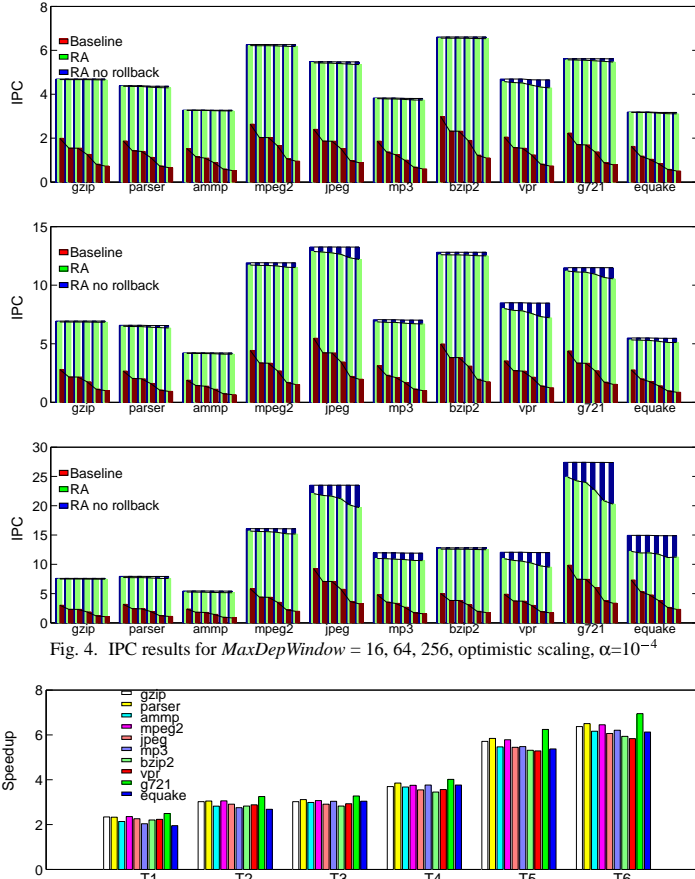


Fig. 4. IPC results for *MaxDepWindow* = 16, 64, 256, optimistic scaling, $\alpha=10^{-4}$

Fig. 5. Speedups RA / Baseline, *MaxDepWindow* = 16, optimistic scaling, $\alpha=10^{-4}$

Figure 5 exhibits the speedup of the reliability aware versus the baseline machine, but this time the results are grouped by technology scaling, i.e., six groups of 10 benchmarks. Due to space limitations we present results only for *MaxDepWindow* 16. As it can be seen, for each technology scaling, the *speedup* is similar across benchmarks. However changes in technology, corresponding to an increase in the ratios of FU latencies for the ρ pipeline over those of the α pipeline, result in substantially increased speedups – from an average speedup of 2.31 for technology T1 (2.23, 2.35 and 2.35, for *MaxDepWindow* 16, 64 and 256, respectively), to an average speedup of 6.42 for technology T6 (6.26, 6.6 and 6.41, for *MaxDepWindow* 16, 64 and 256, respectively). Note that the speedups are roughly invariant to the *MaxDepWindow* considered. The increase in speedup from T1 to T6 exhibits the ability of reliability-driven speculation to hide delay overheads associated with fault tolerant FU designs – as the latency of $(1 - \rho)$ reliable FUs increases, the corresponding speedups increase as well.

While the speedup of the RA machine over the baseline depends to first order on the target technology, as shown in Figure 4, the IPC of both machines decreases as we move from T1 to T6, i.e., as the latency of the $(1 - \rho)$ reliable FUs increases. Specifically the IPC for the baseline decreases on average by 65% (64.9%, 65.6% and 65.6%, for *MaxDepWindow* 16, 64 and 256, respectively) as we move from T1 to T6 – this substantial IPC degradation results from increases in FU latencies across

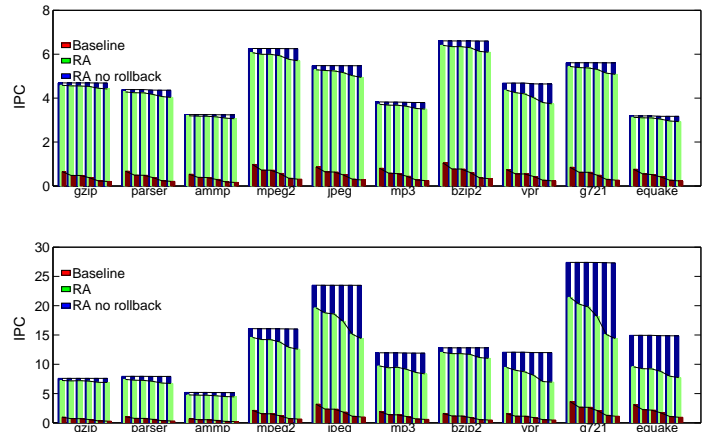


Fig. 7. IPC results for *MaxDepWindow* = 16, 256, pessimistic scaling, $\alpha=10^{-4}$

the six technologies. In contrast, the IPC of the RA machine decreases on average by only 3.82% (1.75%, 3.48% and 6.26%, for *MaxDepWindow* 16, 64 and 256, respectively). The decreased IPC is caused by increases in rollback costs across the six technologies, namely, the higher the FU latencies on the ρ pipeline, the longer it takes to refill the pipeline after a flush. Finally, note that the IPC numbers for the RA machines simulated with no α pipeline errors, and thus no-rollback overheads, remain essentially flat. This is so because, for simulation purposes, the integer ALU latencies for the α pipeline (in number of clock cycles) are normalized to be *identical* for all technologies (see Section IV-A).

B. Results of experiment 2 for optimistic scaling

Figure 6 exhibits representative results obtained for Experiment 2 under the optimistic scaling assumption – see Section IV-C. The graphs show the number of int-ALUs used in the baseline and in the α and ρ pipelines of the RA machine, for a *MaxDepWindow* of 16 and 256. On average, the *total* number of int-ALUs used the RA machine’s pipeline is slightly smaller than the number of int-ALUs used on the baseline machine: 96.9% (109%, 94.6%, and 87.1% as *MaxDepWindow* varies from 16, 64 and 256). However, the superior scalability of reliability-aware machine lies in observing that most of its int-ALUs are instantiated in the much simpler ρ pipeline. That is, the α pipeline, which has a ‘complexity’ commensurate with that of the baseline machine, need only include a fraction of the int-ALUs that are needed in the baseline machine – namely, 18.9%, 16.6%, and 15.1%, as *MaxDepWindow* varies from 16, 64 and 256, giving an overall average of 16.8%.

VI. EXPERIMENTAL RESULTS FOR MACHINES WITH PESSIMISTIC DELAY SCALING

A. Results of experiment 1 for pessimistic scaling

Figure 7 shows IPC graphs for $r=50\%$ (see Table II). The trends discussed in the previous section also hold under this scenario. However we add some additional observations:

- The upper bound on performance of the reliability-aware machines (i.e., ‘no-rollback’ case) is identical to their counterparts in the previous section. Indeed, since with no errors in the α pipeline, the machine operates at maximum throughput, which is independent of the non-critical pipeline stages.
- The speedup of the RA machine over the baseline is much higher under the ‘pessimistic’ scaling assumption. For example, for T1 and *MaxDepWindow* 16, 64 and 256, we observe an average speedup of 5.86, 6.26 and 6.16, respectively. For T6, the corresponding average speedup increases even more dramatically, to 17.6, 18.1, and 16.9 respectively. This is so because the delays incurred to resolve data dependencies, (i.e., issuing to the reservation stations, executing, and writing back the results) have increased substantially for the baseline machine. In contrast, reliability-driven speculation incorporated on the performance critical stages of the RA machine ‘hides’ most such increases.
- The rollback penalty is much larger, and thus the IPC of the RA machine decreases for the pessimistic scaling case. There are two reasons for this. First, more errors are occurring, since the reliability-aware machine now includes two additional unreliable stages in its α pipeline. Sec-

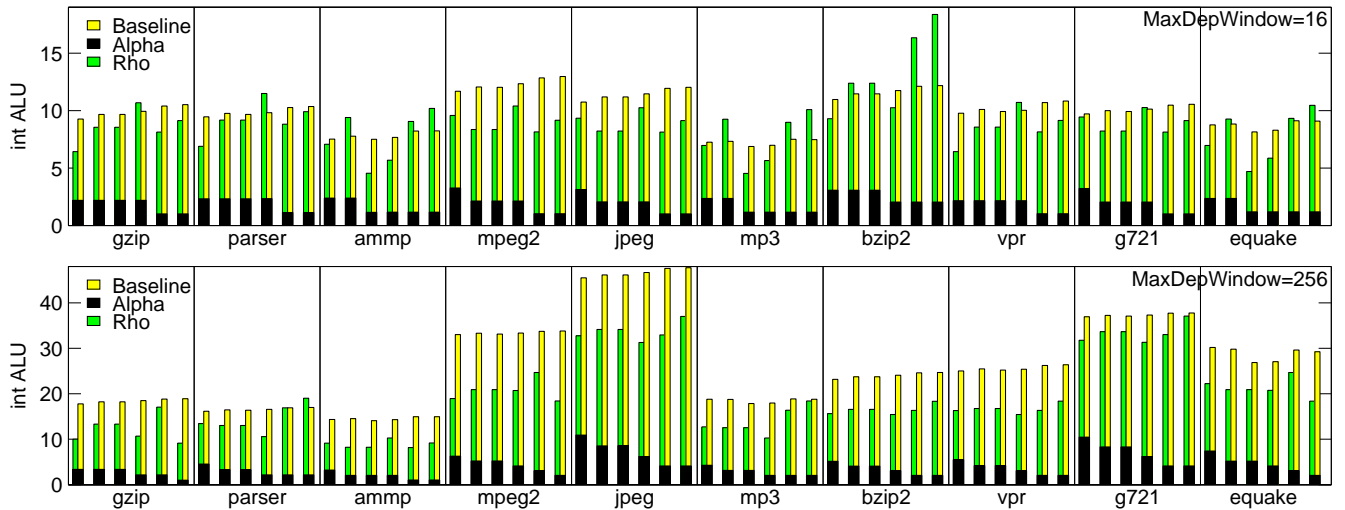


Fig. 6. Integer ALU utilization (average + std. deviation) for baseline and RA machine for optimistic scaling

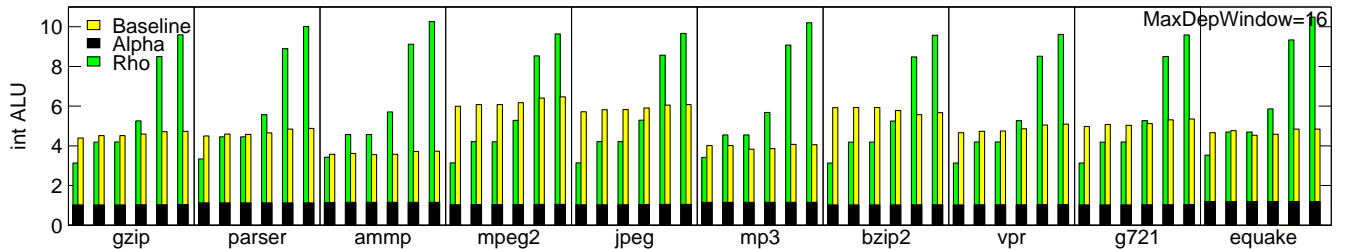


Fig. 8. Integer ALU utilization (average + std. deviation) for Baseline and RA machine for pessimistic scaling.

ond, the pipeline is now deeper, and thus takes longer to refill when it is flushed.

B. Results of experiment 2 for pessimistic scaling

The trends discussed in the previous section still hold under the pessimistic scaling scenario – see e.g., Figure 8, showing the int-ALU utilization for *MaxDepWindow* 16. Namely, the α pipeline again requires a small fraction of the int-ALUs used in the baseline machine – 22.6%, 16.9%, and 14.4%, as *MaxDepWindow* varies from 16, to 64 to 256. Note that the small relative variation of these percentages versus the previous optimistic scaling (increases of 3.7%, 0.3%, and -0.7%, respectively) does not reveal any substantive trend, and is mostly due to lack of ‘resolution’ for our experiment design. In particular, for *MaxDepWindow* 16, the RA machine with a single integer ALU in the α pipeline (i.e., with the minimum possible number of such FUs) exceeds by a substantial margin the IPC of the corresponding baseline, i.e., we are not able to match the performance of the baseline, and are thus biased, resourcewise, against the RA machine.

VII. RELATED WORK

Recent work has focused on leveraging simultaneous multi-threading and chip-multiprocessor microarchitectures for fault-tolerance driven redundant execution of programs, see e.g., [8], [9], [10]. Other researchers have focused on supporting redundant instruction execution on superscalar processors, see e.g. [11]. Despite substantial differences, these approaches share a common assumption – transient faults are assumed to be rare. When this is the case, it suffices to introduce fault tolerance at a high level of design abstraction (or ‘granularity’), while still using standard (non-fault tolerant) component designs.

The notion of *dynamic verification* introduced in the DIVA processor [13] embodies a fundamental departure from the above mentioned approaches, and is closest to the work presented in this paper. The DIVA processor includes a fast unreliable/speculative OOO core and a slower reliable checker. The reliable checker is “implemented with large transistors (that carry ample charge) and large timing and voltage margins, making it resistant to natural radiation interference and noise-related faults”[13]. Because the checker dynamically verifies the results produced by the speculative core, “voltage and timing margins in the core can be significantly tightened, resulting in faster and cooler implementations”[13]. The paper reports that “with sufficient buffering of specu-

lative core results, the latency of the DIVA checker will not impact core processor performance”[13]. The idea of speculatively executing instructions on a faster (less reliable) cores, first proposed in [13], is also explored in this paper. However, there are major conceptual differences relative to our work. Specifically, [13] focuses on “reducing the burden of correctness in microprocessor designs”, while we address the problem of maximizing the performance of a machine targeted at a highly fault-prone substrate. Specifically, in this paper we demonstrate the value of introducing a new class of design parameters, capturing *reliability vs. delay* requirements for select time-critical microarchitectural components, and the need to develop novel reliability-aware microarchitectures that directly expose such performance-critical trade-offs, to enable aggressive performance optimization.

VIII. CONCLUSIONS

In realizing fault-tolerance for computing systems targeted at fault-prone nanotechnologies, one will incur substantial performance overheads. In this paper we propose a new microarchitectural optimization dimension that addresses this problem. Specifically, we introduce the exploration of *reliability–delay* trade-offs at the microarchitecture level so as to optimize performance. This is achieved via a novel form of *reliability-driven* speculation relying on faster but less reliable versions of a microarchitecture’s performance critical components or stages. Our extensive simulations, over a wide array of possible delay-reliability scalings for such technologies, show the promise of reliability-driven speculation in hiding overheads associated with achieving fault-tolerance.

We are currently working on applying these same principles to novel, highly-scalable special-purpose architectures, that would be better able to take advantage of the tremendous device densities afforded by nanotechnologies.

REFERENCES

- [1] G. Bourianoff, “The future of nanocomputing,” *IEEE Computer Magazine*, pp. 44–49, August 2003.
- [2] Y. Huang, X. Duan, Y. Cui, L. Lauhon, K. Kim, and C. Lieber, “Logic gates and computation from assembled nanowire building blocks,” *Science*, vol. 294, pp. 112–1317, 2001.
- [3] D. Rotman, “The nanotube computer,” *MIT Technology Review*, March 2002.
- [4] T. Rueckes, K. Kim, E. Jose-Levich, G. Tseng, C.-L. Cheung, and C. Lieber, “Carbon nanotube based non-volatile random access memory for molecular computing,” *Science*, vol. 289, pp. 94–97, 2000.
- [5] A. DeHon, “Array-based architecture for fet-based nanoscale electronics,” *IEEE Transactions on Computers*, vol. 2, no. 1, pp. 23–32, March 2003.
- [6] S. C. Goldstein and M. Budiu, “Nanofabrics: Spatial computing using molecular electronics,” in *Proc. 28th International Symposium on Computer Architecture*, 2001.
- [7] T. Hanrath and B.A. Korgel, “Supercritical fluid-liquid-solid (sfls) synthesis of si and ge nanowires seeded by colloidal metal nanocrystals,” *Advanced Materials*, vol. 15, no. 5, pp. 437–440, 2003.

- [8] E. Rotenberg, "AR/SMT: A microarchitectural approach to fault tolerance in microprocessors," in *Proc. International symposium on fault tolerant computing*, 1998, pp. 84–91.
- [9] S.K. Reinhart and S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proc. International Symposium on Computer Architecture*, July 2000, pp. 25–36.
- [10] T. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-fault recovery using simultaneous multithreading," in *Proc. 29th Int. Symp. Computer Architecture*, 2002.
- [11] J. Ray, J. Hoe, and B. Falsafi, "Dual use of superscalar datapath for transient-fault detection and recovery," in *Proc. 34th Int. Symp. Microarchitecture*, 2001.
- [12] Margarida F. Jacome, Chen He, Gustavo de Veciana, and Stephen Bijansky, "Defect tolerant probabilistic design paradigm for nanotechnologies," in *Proc. DAC*, 2002, pp. 596–601.
- [13] T. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in *Proc. International Symposium on Microarchitecture*, November 1999, pp. 176–207.
- [14] D. Burger and T. Austin, "The simplescalar tool set," *Computer Architecture News*, vol. 25, no. 3, pp. 13–25, June 1997.