

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.
{zykov,mizan,jacome,gustavo,ajay}@ece.utexas.edu

ABSTRACT

Device and interconnect fabrics at the nanoscale will have a density of defects and susceptibility to transient faults far exceeding those of current silicon technologies. 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. Based on a parameterized microarchitecture, we exhibit the benefits of optimizing these trade-offs.

Categories and Subject Descriptors: C.1 [Computer Systems Organization] Processor Architectures, Performance of Systems, B.8 [Hardware] Performance and Reliability

General Terms: Performance, Design, Reliability.

Keywords: Nanotechnologies, Fault Tolerant Microarchitectures, Performance Optimization, Reliability-Delay Trade-offs.

1. 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]. 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 [4, 5].

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: (1) will have a density of defects which is much higher than current silicon technologies [1]; and (2) are likely to be much more susceptible to transient faults (soft errors) [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. Further-

more, 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.

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. Based on a simple parameterized microarchitecture, we exhibit the benefits of *explicitly* exploring this novel class of trade-offs at the microarchitectural level.

2. RELIABILITY-DELAY TRADE-OFFS AT THE MICROARCHITECTURE LEVEL

For the purpose of this research we will use a parametric model that captures how a component's delay might scale with its desired reliability. The reliability of a component is the probability it performs its function correctly on a given use. Our model is based on fundamental considerations on how reliability increases with redundancy, and how the increased area associated with redundancy would lead to higher delays – see [6] for details. This strongly suggests 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 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. 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 out-of-order (OOO) superscalar processors. 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.

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,

*This work is supported in part by SRC Grant CRS 1152.001 and NSF Grant CCR 031019

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2005, June 13–17, Anaheim, California, USA.

Copyright 2005 ACM 1-59593-058-2/05/0006 ...\$5.00.

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. 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 components, i.e., the actual delays are likely to be between these two limits.

2.1 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*. Naturally, 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 .

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 [7]. 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. For additional details on the RA microarchitecture see [6].

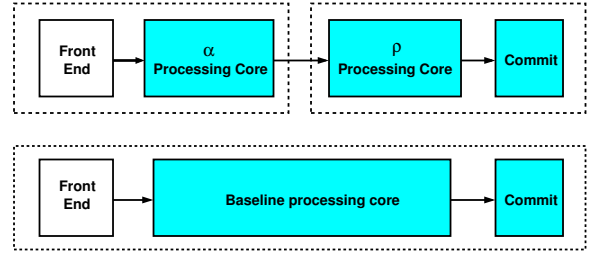


Figure 1: Top-level pipeline organization for RA and baseline machines

2.2 Other forms of reliability driven speculation

Under the ‘pessimistic’ scaling assumption, improving the speed at which the processing core resolves data dependencies requires exploiting reliability-delay trade-offs also on the issue and write-back stages of the alpha pipeline. 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. Thus, load memory accesses are now first issued with $(1 - \alpha)$ reliability in the alpha pipeline, and then validated in the ρ pipeline. We consider two fundamental types of errors for the unreliable issue and write-back stages: (1) errors that can be detected locally, for example, using reliable watchdog timers (e.g., instruction is never sent to functional unit); and (2) errors that can be only detected when the instruction is re-executed in the ρ pipeline (e.g., instruction is issued with incorrect operands) [6]. Clearly, the second type of errors incurs a performance penalty that is much more severe than that associated with the first type of error. ¹

3. EXPERIMENTAL METHODOLOGY

3.1 Modeling different hypothetical nanotechnologies

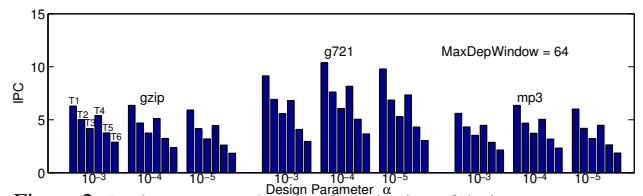


Figure 2: Design space exploration for selection of design parameter α .

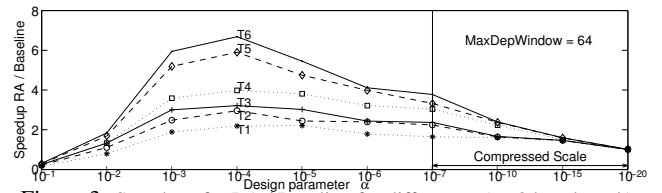


Figure 3: Speedups for RA / Baseline for different α (mp3 benchmark).

A total of six possible technology scalings T1 – T6 were examined – see Table 1. Latencies, measured in cycles, are presented for

¹In the experiments discussed in Section 4, we assume the errors of types 1 and 2 are equally likely.

Technology	FU latencies in cycles $\rho(\alpha)$				Relative cycle delay
	int ALU	int Mul/Div	fp ALU	fp Mul/Div	
T1	3 (1)	8/54 (4/22)	5 (3)	10/32 (5/13)	1.00
T2	4 (1)	11/79 (4/22)	7 (3)	15/46 (5/13)	1.35
T3	4 (1)	13/91 (4/22)	9 (3)	17/53 (5/13)	1.69
T4	5 (1)	16/123 (4/23)	11 (3)	22/71 (5/14)	1.24
T5	8 (1)	24/175 (4/23)	15 (3)	32/102 (5/14)	1.95
T6	9 (1)	27/193 (4/23)	18 (3)	36/113 (5/14)	2.66

Table 1: Latencies for reliability $1-\rho$ and $1-\alpha$.

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. For a detailed discussion supporting these numbers see [6].

3.2 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.

Figure 2 shows a sample of our design space exploration results for the reliability-aware (RA) machine described in Section 1.1. 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 2.3 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 .

3.3 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 set several values for *MaxDepWindow*, namely, 16, 64 and 256, and configured 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 re-

liability aware versus baseline machines. Experiment 2 focuses on actual scalability 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.

3.4 Experimental Setup

The simulators used in our experiments are derived from the SimpleScalar simulator [8], targeted at the PISA instruction set. The main modifications and extensions to the simulator are described in [6]. We selected 10 benchmarks from the CPU2000 and Mediabench suites, including integer (gzip, parser, mpeg2, jpeg, bzip2, g721) and FP (ammp, mp3, equake) applications, with different degrees of ILP. All benchmarks were compiled with the `-O2 -funroll-loop` flags and 10M instructions were analyzed for each one of them, after forwarding a sufficiently large number of instructions.

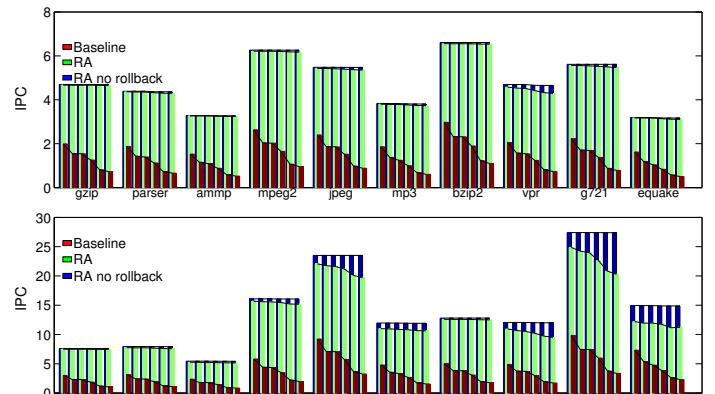


Figure 4: IPC results, *MaxDepWindow* = 16, 256, optimistic, $\alpha=10^{-4}$

4. EXPERIMENTAL RESULTS FOR MACHINES WITH OPTIMISTIC DELAY SCALING

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 ‘RA no-rollback’ in the figures, was simulated to assess the penalties incurred in the actual RA machine due to faulty operation. 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 and 256. Graphs for *MaxDepWindow* 64 can be seen in [6].

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

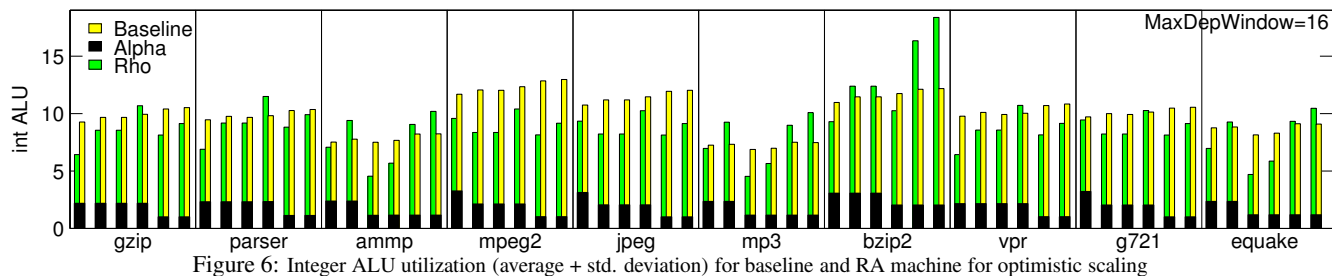


Figure 6: Integer ALU utilization (average + std. deviation) for baseline and RA machine for optimistic scaling

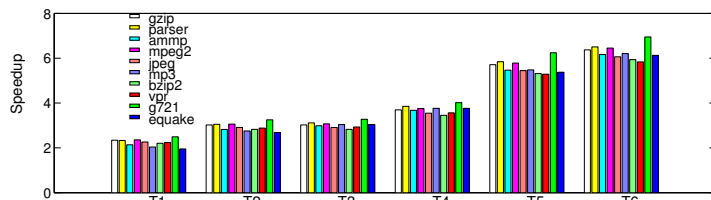


Figure 5: Speedups RA/Baseline, MaxDepWindow=16, optimistic, $\alpha=10^{-4}$

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 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). 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 clock cycles) are normalized to be *identical* for all technologies (see Section 2.1).

Results of experiment 2 for optimistic scaling. Figure 6 exhibits representative results obtained for Experiment 2 under the optimistic scaling assumption. The graph shows 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. 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 to 64 and 256 – see [6]). However, the superior scalability of the reliability-aware machine lies in observing that most of its int-ALUs are instantiated in the much simpler ρ pipeline. That is, the much more ‘power-hungry’ 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 –

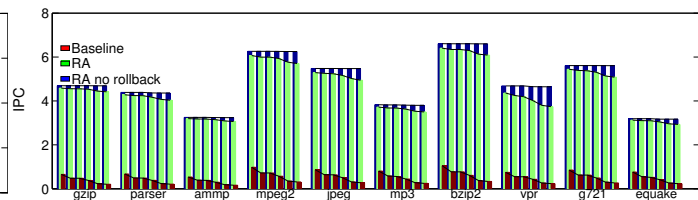


Figure 7: IPC results, MaxDepWindow = 16, pessimistic, $\alpha=10^{-4}$

namely, 18.9%, 16.6%, and 15.1%, for *MaxDepWindow* 16/64/256.

5. EXPERIMENTAL RESULTS FOR MACHINES WITH PESSIMISTIC DELAY SCALING

The speedup of the RA machine over the baseline is much higher under the ‘pessimistic’ scaling assumption – see e.g. Figure 7. 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. Concerning experiment 2, the α pipeline again requires a small fraction of the int-ALUs used in the baseline machine – see [6] for detailed data.

6. RELATED WORK AND CONCLUSIONS

The idea of speculatively executing instructions on a fast unreliable core, first proposed in the DIVA processor [7], is also explored in our work, yet in a conceptually different way. Namely, [7] focuses on “reducing the burden of correctness in microprocessor designs”, while we address the problem of maximizing the performance of a machine targeted at fault-prone substrates. Specifically, we demonstrated the value of introducing a new class of design parameters, capturing *reliability–delay* requirements for time-critical microarchitectural components, and the need to develop novel reliability-aware microarchitectures that expose such performance-critical trade-offs, to enable performance optimization.

7. REFERENCES

- [1] G. Bourianoff, “The future of nanocomputing,” *IEEE Computer Magazine*, August 2003.
- [2] Y. Huang et al., “Logic gates and computation from assembled nanowire building blocks,” *Science*, vol. 294, 2001.
- [3] D. Rotman, “The nanotube computer,” *MIT Technology Review*, March 2002.
- [4] A. DeHon, “Array-based architecture for FET-based nanoscale electronics,” *IEEE Trans. on Computers*, vol. 2, no. 1, March 2003.
- [5] S. C. Goldstein and M. Budiu, “Nanofabrics: Spatial computing using molecular electronics,” in *Proc. ISCA 28*, 2001.
- [6] A. V. Zykov et al., “High performance computing on fault-prone nanotechnologies: Novel microarch. techn. exploiting reliability delay trade-offs,” Tech. Rep., UT-CERC-TR-MJ-0502, 2005.
- [7] T. Austin, “DIVA: A reliable substrate for deep submicron microarchitecture design,” in *Proc. MICRO*, November 1999.
- [8] D. Burger and T. Austin, “The simplescalar tool set,” *Computer Architecture News*, vol. 25, no. 3, June 1997.