Flexible Error Protection for Energy Efficient Reliable Architectures[*]

Timothy Miller[†], Nagarjuna Surapaneni[⋆] and Radu Teodorescu[†]
[†]Department of Computer Science and Engineering
[⋆]Department of Electrical and Computer Engineering
The Ohio State University
{millerti,teodores}@cse.ohio-state.edu, nagarjun@ece.osu.edu

Abstract

Technology scaling is having an increasingly detrimental effect on microprocessor reliability, with increased variability and higher susceptibility to errors. At the same time, as integration of chip multiprocessors increases, power consumption is becoming a significant bottleneck that could threaten their growth. To deal with these competing trends, energy-efficient solutions are needed to deal with reliability problems.

This paper presents a reliable multicore architecture that provides targeted error protection by adapting to the characteristics of individual cores and workloads, with the goal of providing reliability with minimum energy. The user can specify an acceptable reliability target for each chip, core, or application. The system then adjusts a range of parameters, including replication and supply voltage, to meet that reliability goal. In this multicore architecture, each core consists of a pair of pipelines that can run independently (running separate threads) or in concert (running the same thread and verifying results). Redundancy is enabled selectively, at functional unit granularity. The architecture also employs timing speculation for mitigation of variation-induced timing errors and to reduce the power overhead of error protection. On-line control based on machine learning dynamically adjusts multiple parameters to minimize energy consumption. Evaluation shows that dynamic adaptation of voltage and redundancy can reduce the energy delay product of a CMP by $30-60\%$ *compared to static dual modular redundancy.*

1 Introduction

Transistor scaling to minute sizes makes modern microprocessors less reliable and their performance and power consumption less predictable and highly variable. Microprocessor chips are especially vulnerable to three classes of errors. **Soft errors**, or single event upsets (SEU), occur as a result of particle strikes from cosmic radiation and other sources. As technology scales, the soft error rate in chips is expected to increase due to the higher number of transistors and the lower operating voltages. **Timing errors** occur when the propagation delay through any exercised path in a pipeline stage exceeds the cycle time of the processor. Timing errors can have multiple causes including variation in threshold or supply voltages, circuit degradation as a result of aging, high temperature, etc. **Hard errors** are permanent faults in the system, caused by breakdown in transistors or interconnects. Several factors can cause permanent failures including aging, thermal stress and manufacturing variation [1]. To ensure the continued growth in chip performance, microprocessors must be resilient to all of these types of errors. Moreover, reliability solutions must work within limited power budgets.

The core of our solution is a reliable processor architecture that dynamically adapts the amount of protection to the characteristics of each individual chip and its runtime behavior. In this multicore architecture, each core consists of a pair of pipelines that can run independently (separate threads) or in concert (running the same thread and checking for errors). Redundancy is enabled selectively, at pipeline stage granularity, to allow targeted error protection at reduced cost. The architecture also employs timing speculation for mitigation of variation-induced timing errors and fine-grain voltage scaling to reduce the power overhead of the error protection.

Different applications have different reliability requirements. An OS kernel or financial application may require very high protection, and previous works provide numerous solutions to this problem. On the other hand, less critical applications like word processors and video players can tolerate the occasional error and therefore require only a moderate or low level of protection. Our system allows *failure rate targeting* in which the user or the system is allowed to specify an acceptable failures-in-time rate (or FIT target) for the entire chip or individual cores. Targeting a desired FIT rate has several benefits. It allows the same CMP to be deployed in systems with different reliability requirements. It allows the system to dynamically adjust the amount of protection needed to achieve a FIT depending on the application activity and supply voltages, resulting in energy savings. And it allows distinct reliability goals to be assigned to individual applications.

Our system uses an optimization algorithm that adjusts a range of parameters, including which functional units (FUs) are replicated and their supply voltage, to meet that target with minimum energy. Our optimization relies on models of key parameters of the system such as power consumption and expected error rates. In the presence of variation, these parameters are difficult to model analytically so we use machine learning-based models that are trained at runtime.

Compared to static dual modular redundancy (DMR), our system reduces the average energy delay product by 30% when no errors are allowed and up to 60% as the FIT target is relaxed. Based on preliminary results from synthesis of a simple RISC processor implementation we find the area overhead of our system to be about 4% and the impact on cycle time to be about 10% compared to static DMR.

This paper makes the following contributions:

- Presents an architecture that provides simultaneous protection against soft and timing errors and some hard errors.

[*]This work was supported in part by the National Science Foundation under grant CNS-0403342.

- Introduces *FIT targeting* which allows the degree of error protection to vary dynamically to reduce energy usage.
- Proposes a machine-learning approach to online modeling of power consumption and timing errors of variation-affected, unpredictable CMPs and an optimization algorithm based on hill-climbing that uses these models to find optimal energy configurations.
- Presents a novel implementation of timing speculation that uses pipeline registers of the shadow pipeline instead of dedicated flip-flops. This implementation allows no-cost timing speculation when full replication is enabled.

2 Related Work

Several existing or proposed architectures deal with soft errors by replicating entire functional units (FUs). The IBM G5 [2] uses full replication in the fetch and execution units with a unified ECC-protected L1 cache. Others proposed replication and checking for soft errors at latch level [3]. Fine-grain replication is appealing because it allows targeted protection of only the sections or paths in a chip that are deemed most vulnerable at design time. However, dynamically enabling/disabling replication at latch level would make control very complex and costly. Our architecture uses replication at FU granularity that is selectively enabled at runtime depending on desired protection.

Techniques that detect and correct timing errors take two main approaches. One is to use secondary latches to capture the delayed signals like in Razor [4]. Another uses a simple checker that verifies execution of the main processor as in DIVA [5]. Timing speculation has been used to reduce voltage aggressively to save power [4] or to over-clock a processor to improve performance [6]. We implement timing speculation differently from prior work. We use the pipeline registers of the shadow pipeline instead of special flip-flops.

Previous work on hard faults has proposed mechanisms for efficient detection of hard errors using the processor's built in self-test (BIST) mechanism [7] and using spare logic to replace faulty components [8, 9]. In Core Cannibalization [8], pipelines are arranged in triples; two pipelines are used for execution, and the third is used for spare parts at the pipeline stage granularity. In StageNet [9], multiple processor pipelines are interconnected using crossbar switches after each pipeline stage allowing re-routing of instructions in case of failures. The complex routing logic introduces longer and variable pipeline latency, requiring additional logic to make up for the loss of result forwarding. Our design groups pipelines into pairs, with simple two-way routing logic with less impact on the processor design.

EVAL [10] uses on-line adaptation of supply voltage and body bias, controlled by a machine learning algorithm. EVAL is targeted exclusively at timing errors and improving performance in the face of process variation. While EVAL is efficient for this purpose, it has no capability to mitigate soft errors or hard failures.

Aggarwal *et al.* [11] present a mechanism for partitioning CMP blocks at coarse granularity. Processor cores and memory controllers can be configured into groups to achieve, among other possibilities, dual and triple modular redundancy. This system can be configured for different reliability needs but the coarse granularity makes the approach less flexible. Our architecture provides redundancy and checking at fine granularity, allowing more efficient recovery and more targeted error protection.

In [12], authors present a reinforcement learning approach to schedule requests from multiple out-of-order processors competing for access to a single off-ship DRAM channel. In a circuit area no worse than a branch predictor, they enjoy a 22% boost in throughput over other cutting-edge schedulers.

3 Flexible Redundant Architecture

In this architecture, each core consists of a *pair of pipelines*. Routing and configuration logic allows each pipeline to run independently (each running a separate thread) or in concert (both running the same thread and checking results at the end of each pipeline stage). Routing and checking logic is provided at pipeline stage granularity.

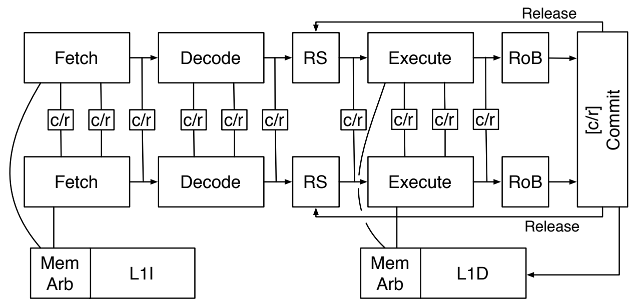


Figure 1: Architecture of the proposed "pipeline pair" for one core, with routing and checking logic at pipeline stage granularity.

3.1 Support for Soft Error Detection

Figure 1 shows an overview of a pipeline pair, based on the Intel Core architecture. Some blocks in the diagram, such as Decode, are comprised of multiple pipeline stages, and the Execute block stands in for several multi-stage functional units (FUs), including integer and floating point ALUs, and load/store. One pipeline is always enabled, referred to as the *main* pipeline. The second, *shadow*, pipeline can have some of its FUs selectively enabled. Each pipeline stage has routing and checking logic, indicated by c/r in the diagram. All stages are separated by simple two-way routers (multiplexers) that allow results from one stage to be routed to the inputs of the next stages in both pipelines. This allows stages that are disabled in the shadow pipeline to be bypassed. The shadow stages that are enabled can receive their inputs from the previous stage of the either pipeline.

We assume a deterministic out-of-order architecture. Although instruction scheduling decisions are made dynamically, if the two pipelines start with identical initial conditions and receive identical inputs, they will make identical scheduling decisions. At each pipeline stage, computation results and control signals are forwarded to checkers. Checkers are used to verify the computation of stages that are replicated. The checking takes place in the cycle following the one in which the signals are produced, and the inputs to the checkers come from the pipeline control and data registers. This keeps checkers out of the critical path.

Fetch and Decode are replicated, and individual pipeline stage outputs are verified by checkers. The reservation station (RS) allows for register renaming and forwarding of operands between instructions. The RS (also replicated) has multiple outputs corresponding to each compute unit it serves (i.e. ALU, Multiplier, Load/Store). The RS outputs corresponding to each compute unit are verified by separate checkers. The RS entry is not freed until commit from the reorder buffer (RoB) succeeds. In the following cycle, checkers compare the issued instructions. The same is true for each pipeline stage of each Execute unit.

Retirement from the RoB is handled by a special Commit unit. When only timing speculation is being performed, Commit acts like any other checker; if a timing error is detected, execution is stalled, and results are taken from the shadow pipeline. When full replication is enabled, Commit checks the integrity of instructions dequeued from the two RoBs. If a disagreement is detected, Commit discards the instruction and signals reservation station(s) to reissue. The Commit stage is not replicated and represents a potential single point of failure. To protect it, some other hardening approach must be used. For instance, latch-level redundancy [3] or transistor up-sizing can be employed.

The L1 instruction and data caches are not replicated and are shared by the two pipelines. The caches are protected by ECC so replication is not necessary for data integrity. Cache supply voltage is kept high enough to avoid timing errors. In replicated mode, both pipelines fetch the same instructions and data from the L1. In independent mode, the two pipelines fetch separate instruction and data streams from a shared L1. To ensure fairness, half of the cache ways (of set associativity) are reserved for each pipeline. Arbitration logic (Mem Arb) manages memory allocation and requests in the cache. When full replication is enabled, both pipelines will request the same access; arbitration ensures that the addresses (and data for writes) are the same, issues one access to the memory array and returns data to both pipelines.

3.2 Support for Timing Speculation

This architecture can also be configured to implement timing speculation at pipeline stage granularity. Timing speculation is useful in mitigating the effects of variation on circuit delay and also allows the aggressive lowering of supply voltage to save power. If a FU is not fully replicated, this is achieved by selectively enabling only the pipeline registers of the shadow pipeline, which has a slightly delayed clock at the same clock frequency as the main pipeline. Using routing logic, computation results of a stage in the main pipeline are also latched in the pipeline registers of the shadow pipeline as shown in Figure 2. The delay in the shadow pipeline's clock (ΔT) gives extra time to the signals propagating through the main pipeline. Computation results are latched in the main pipeline's register at time T and in the shadow pipeline's register at time $T + \Delta T$. If a timing error causes the wrong value to be latched by the main pipeline, the extra time ΔT will allow the correct value to be latched in the shadow register. The content of the two registers is compared by a checker in the next cycle.

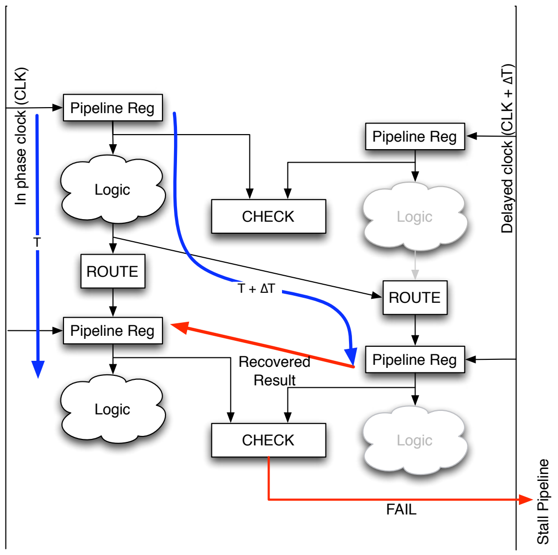


Figure 2: Main and shadow pipeline stages with timing speculation enabled. Only the shadow registers are turned on in the shadow pipeline.

Our implementation is different from previous work [4] in that we use the shadow pipeline registers as a safety net for delayed signals, instead of special flip-flops. Our approach has significant advantages: it allows us to cover all the critical paths in the system, rather than trying to predict which paths are likely to be critical (which is almost impossible because of variation), and it also allows no-cost timing speculation for FUs that have full replication enabled.

3.3 Support for Mitigation of Hard Faults

Although it is not the main focus of this paper, this architecture can cope with some hard faults. When the two pipelines have complementary failures, they can be merged at pipeline stage granularity to form one functional pipeline, as in [8].

3.4 Error Recovery

Errors are detected by comparing the content of the main and shadow pipeline registers (data and control signals). The comparison takes place in the cycle following the computation. When the results disagree, a stall signal is asserted and recovery is initiated. The recovery process depends on the type of error each FU is configured to capture.

When a FU is configured to detect only timing errors, the pipeline registers in the shadow pipeline have extra time to latch the results of the previous stage and are therefore assumed to hold the correct results. These recovered results are forwarded to the corresponding pipeline register in the main pipeline through the routing logic as shown in Figure 2. Execution then resumes with the correct result in the main pipeline register. The penalty for a timing error is at most two cycles and may be hidden if it occurs after the RS stage.

When a FU is fully replicated, both soft errors and timing errors can be detected but not distinguished. When an error is detected in the reservation station or a stage prior, the checker triggers a full pipeline flush followed by a re-execution, similar to a branch mispredict. When an error is detected in a stage following the RS, the checker logic in Commit causes the instruction to be discarded and reissued from the RS. If the fault was caused by a soft error, re-executing the instruction will eliminate the fault. However, if the error is timing-related, it is likely to reoccur. To deal with the latter case, both instructions and stages that experience errors are flagged with an error marker. If the error occurs again in the same stage, while executing a marked instruction, the error is assumed to be timing related and the correct result is forwarded from the shadow pipeline register.

The checkers represent single points of failure in this system. Since checkers are small, hardening (transistor replication and up-sizing) can be done with low overhead.

3.5 Additional Hardware Needed

Routing and checking – The routing configuration for a FU pair selects which block of combinatorial logic feeds each pipeline register. Each pair of pipeline stages has an associated checker that can detect when the pair of pipeline registers disagrees. This can be enabled when pipelines are running in lockstep or phase-shifted.

Power gating – Each FU and each pipeline register can be enabled separately. Power gating cuts both leakage and dynamic power by disconnecting idle blocks from the power grid. This technique has been extensively studied and can be implemented efficiently at coarse (FU) granularity [13].

Voltage selects – As part of a strategy to minimize energy consumption we allow different FUs to receive different supply voltage levels. Depending on the number of separate voltages needed, different hardware support is needed. To keep the overhead low, rather than providing each FU with its own supply voltage, one option is to have only two or three voltage levels. Each FU (and its pipeline register(s)) selects among those. For two or three voltage levels, off-chip voltage regulators are sufficient. Chips in production today commonly use several voltage domains [14] using off-chip regulators. In order to provide each FU with its own voltage, on-chip voltage regulators [15] must be used.

Clock controls – There are two PLL circuits for each pipeline pair, and each PLL produces a configurable clock signal, along with a phased-delayed clock with configurable delay for timing speculation.

4 FIT Targeting and Timing Speculation

An important feature of the proposed architecture is its ability to adapt to different reliability goals depending on the needs and resource constraints of the system. When maximal protection against soft errors is not needed, some redundancy can be selectively and dynamically disabled to reduce power. The system designer can choose a tolerable error rate or FIT (the number of failures for 1 billion hours of operation). For instance, IBM targets a FIT of 114 or 1000 years mean time between failures (MTBF) for its Power2 processor-based systems [16].

A FIT target can be set for the entire CMP, for individual cores, or per-application. This allows the system to adapt the level of protection against soft and timing errors to different applications and environments. For instance, a core running essential system services might be configured with a low FIT target, while cores running user services might tolerate a higher FIT. Moreover, when targeting a system FIT rate, the number of cores in the system will determine the per-chip FIT rates since their contribution to the total FIT rate is additive. The expected FIT for a core is the sum of the FIT for all its functional units (FUs). In our system, caches are protected with ECC, so their contribution to the expected system FIT rate is assumed to be zero. The FIT rate for a FU with full redundancy enabled is also assumed to be zero. If redundancy is not enabled, the FIT rate is a function mainly of the raw soft error rate for that FU, its supply voltage and the FU's *architectural vulnerability factor* (AVF), or a probability that a soft error will result in an actual system error.

Previous work [17, 18] has demonstrated that predicting AVF is possible and practical at runtime by examining a set of architectural parameters such as IPC, ROB utilization, branch mispredictions, reservation station utilization, instruction queue utilization, etc. We use a similar approach to predict dynamic AVF, but at FU granularity.

4.1 Saving Energy with Timing Speculation

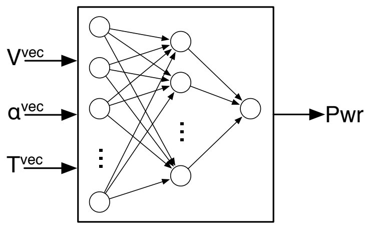
In addition to selective replication, timing speculation is used to save power independent of the FIT target. To reduce power consumption the voltage is lowered, on a per FU basis, to the point of causing timing-related errors with a low probability. As long as the cost of detecting and correcting errors is low enough, the voltage level that achieves minimum energy will often come with a non-zero error rate. If full replication is enabled, timing speculation can be performed with no additional power overhead. However, if full replication is not enabled the system must determine if, for each FU, timing speculation is beneficial. As a failsafe mechanism, we determine whether or not pipeline register replication and checking are required, using a special circuit path (called a critical path replica — CPR) embedded in each FU [19]. The CPR is longer than the critical path of the unit, allowing detection of impending timing errors. Replication is automatically switched on and off based on this sensor.

5 Runtime Control System

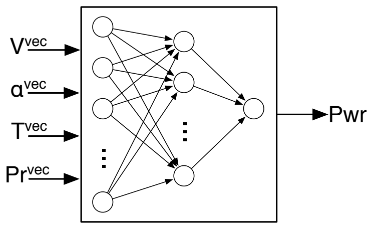
FIT targeting and timing speculation are controlled by a runtime optimization mechanism. The system is first assigned a FIT target by the manufacturer or user. The FIT target can change at runtime if the reliability goals for the system or application change. Next, the runtime optimization system searches for the replication and timing speculation settings that achieve the FIT target with minimum energy. This step is solved using an optimization algorithm with inputs from a set of machine learning-based models for power and timing error probability.

5.1 Machine Learning-based Modeling

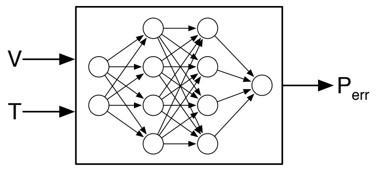
Process variation results in different power and delay characteristics for each FU within each pipeline [20, 19]. These characteristics are difficult to predict and model analytically. To deal with this challenge we use artificial neural nets (ANNs) to model the power and timing error probability for all FUs in the system. The models are trained using measured data such as temperature, current power consumption for each pipeline, past error rate and utilization.



(a) Primary power



(b) Shadow power



(c) Error probability

Figure 3: The architecture of Artificial Neural Nets used for power and error probability prediction.

5.1.1 ANN Architecture

To model energy based on temperature, voltage, and utilization, we use three ANN architectures, shown in Figure 3. An ANN models a function that takes N inputs and yields M outputs. ANNs are typically architected in layers of nodes. In the input layer, there are N nodes, each corresponding to an input. Likewise, in the output layer there are M nodes. A simple ANN with no hidden layers is called an ADALINE (Adaptive Linear Element), where each output is simply the inner product of the N inputs and a set of N weights, plus a linear bias. An ADALINE requires $M(N+1)$ weights. To model *non*linear functions, we add hidden layers. The first hidden layer is computed as in the ADALINE, but then each hidden node's value is processed though an *activation function* that adds nonlinearity.

Primary Power ANNs (Figure 3(a)) predict power consumption for the primary pipeline (P_p). There are 3 inputs for each FU: voltage, utilization (counted proportion of active cycles), and temperature (interval average). There are 12 nodes in one hidden layer and one output node.

Shadow Power ANNs (Figure 3(b)) predict power consumption for the shadow pipeline (P_s). There are 5 inputs for each FU: voltage, utilization, temperature, and binary values indicating replication (11_b for none, 01_b for full, 10_b for pipeline register only). There are 10 nodes in one hidden layer and one output node.

Error Probability ANNs (Figure 3(c)) predict raw probability of an error occurring on each cycle ($P(E)$). Since each FU has its own error counter, error probability for each is modeled separately. Each ANN has two inputs: voltage and temperature (interval average). There are four nodes in each of two hidden layers and one output node.

The number of errors (N_E) experienced by a given FU is the product of $P(E)$, utilization, and clock cycles in the measurement interval (C_m), rounded to the nearest integer. Recovery penalty (R_p) is computed from the total number of errors over all FUs, which depends on the error protection mode of each FU. When full replication is not required, a FU's shadow pipeline register is enabled when $N_E > 0$. Total energy is $(P_p+P_s)\times(1+R_p/C_m)$.

ANNs are trained on-line by comparing predictions against measurements and adjusting weights to improve prediction.

There are several approaches for implementing ANNs in hardware. In [21], a small, fast, low-power ANN is built from analog circuitry. Other alternatives include simple digital logic as in [12]. We give an estimate for the amount of hardware needed in our case in Section 7.1.

5.2 Runtime Optimization System

The energy optimization given a FIT target is performed at regular intervals. Figure 4 shows a flowchart of the optimization process. The optimizer relies on profiling information collected during most of the interval, followed by optimization calculations. Profiling and optimization is performed in parallel to program execution. At the end of a profiling phase, temperatures are measured and utilization counters are used as input to the error, power, and FIT models during optimization. When optimization has completed, new voltages and replication settings are applied. The optimizer could be implemented in software and run periodically at the end of each adaptation interval or run continuously in an on-chip programmable controller similar to Foxton [22].

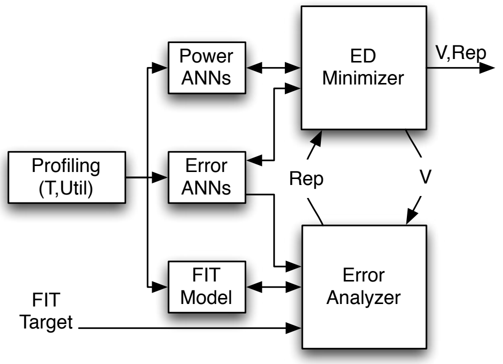


Figure 4: Runtime optimization system

For every interval, our objective is to find a set of configuration settings that (a) minimize energy or the energy delay product, (b) prevent timing errors, and (c) meet a specified FIT target. The number of combinations of voltage and replication settings to consider is exponential, and interactions between settings make it impossible to compose local optimizations to find a global optimum. We therefore employ a hill-climbing algorithm (Figure 5) to search the voltage space and an error analysis function to compute replication settings. The algorithm starts with maximum voltages for all FUs and lowers them one step at a time, checking for errors, and computing ED. Voltages are lowered until minimum ED is found.

Given a vector of voltages, the *Error Analyzer* computes replication settings that meet the FIT target and prevent timing errors. If requirements can be met, the analysis yields a set of replication settings (none, full, or partial for each FU). Otherwise, it reports failure to invalidate this configuration.

To meet a FIT target, the *Error Analyzer* identifies a set of FUs for which full redundancy must be enabled in order to get as close as possible to the FIT target without exceeding

```
Create an array of voltages, one for each functional unit; initialize all to 1.0V.
Repeat
    For FU in functional units
        Lower voltage for FU by one step
        Calculate total ED
        If total ED is less than the best so far, keep this voltage level and flag
          → a change. Otherwise, restore it to the previous value.
    End For
Until there are no more changes
Repeat
    For FU in functional units
        For V in all possible voltage levels
            Holding all other units constant, set voltage for FU to V
            Calculate total ED
            If total ED is less than the best so far, keep this voltage level and
              → flag a change. Otherwise, restore it to the previous value.
        End For
    End For
Until there are no more changes
```

Figure 5: Hill-climbing search for optimal voltages

it. To do this, we apply a greedy algorithm that we call "best fit FIT first." A FU is selected whose estimated FIT rate[1] is closest to the difference between the target FIT and the current estimated total. Enabling full redundancy effectively reduces a FU's FIT to zero, yielding a reduced estimated total FIT. If the FIT target is not met, another FU is selected to be replicated. This is repeated until the FIT target is met.

For each remaining FU that has not been selected for full replication, the *Error Analyzer* selects partial (pipeline register) replication if it is vulnerable to timing errors at its current voltage. ANNs are used to predict power and probability of error. The optimization yields the voltages for which ED was minimum, along with replication settings.

6 Evaluation Methodology

We use a modified version of the SESC cycle-accurate execution-driven simulator [23] to model a system similar to the Intel Core 2 Duo modified to support redundant execution. Table 1 summarizes the architecture configuration.

Architecture: Core 2 Duo-like processor
Technology: 32nm, 4GHz (nominal)
Core fetch/issue/commit width: 3/5/3
Register file size: 40 entry; Reservation stations: 20
L1 caches: 2-way 16K each; 3-cycle access
Shared L2: 8-way 2 MB; 7 cycle access
Branch prediction: 4K-entry BTB, 12-cycle penalty
Die size: $195mm^2$; V_{DD}: 0.6-1V (default is 1V)
Number of dies per experiment: 100
V_{th}: μ: 250mV at 60 ^{o}C
σ/μ: 0.03-0.12 (default is 0.12)
ϕ (fraction of chip's width): 0.5

Table 1: Summary of the architecture configuration

6.1 Variation, Power and Temperature Models

We model variation in threshold voltage (V_{th}) and effective gate length (L_{eff}) using the VARIUS model [24, 19]. Table 1 shows some of the process parameters used. Each individual experiment uses a batch of 100 chips that have a different V_{th} (and L_{eff}) map generated with the same μ, σ, and ϕ. To generate each map, we use the geoR statistical package [25] of R [26]. Resolution is 1/4M points per chip.

To estimate power, we scale results given by popular tools using technology projections from ITRS [27]. We use SESC [23] to estimate dynamic power at a reference technology and frequency. In addition, we use the model from [24] to estimate leakage power for same technology. We use HotSpot [28] to estimate on-chip temperatures.

6.2 Timing and Soft Error Models

We use the timing error model developed in [24]. The model takes into account process parameters such as V_{th}, L_{eff} as well as floorplan layout and operating conditions such as supply voltage and temperature. It considers the error rate in logic structures, SRAM structures and hybrids of both, with both systematic and random variation. The model has been validated with empirical data [29]. With this, we estimate the timing error probability for each functional unit (FU) of each chip at a range of supply voltages.

For soft errors, we use the approach in SoftArch [30]. We determine the raw soft error rate for 50nm technology from [31]. Failure in Time (FIT) values for latch and combinational logic chain were also extracted from [31]. We scale that to 32nm using the predictions from [32]. Based on the transistor count for the Core 2 Duo floorplan we estimate the number of transistors in latches and combinational logic in each FU. Based on that count and the mix of logic chains and latches, we determine FIT values for each FU. To model AVF we use an approach similar to [17]. For logic-dominated FUs we measure activity for those units and scale the expected FIT accordingly. For memory dominated FUs we consider both activity and occupancy.

6.3 Benchmarks

We use benchmarks from the SPEC CPU2000 suite (*bzip2, crafty, gap, gzip, mcf, parser, twolf, vortex, applu, apsi, art, equake, mgrid* and *swim*). The simulation points present in SESC are used to run the most representative phases of each application with the reference input set.

7 Evaluation

In this section, we show the effects of FIT targeting and timing speculation on energy reduction. We also show an evaluation of the area, power and timing overheads of the proposed architecture.

7.1 Overheads

The proposed architecture introduces some timing, power and area overhead. To estimate it, we synthesized the Open Graphics Project HQ microcontroller [33] for Xilinx Spartan 3 FPGA. The synthesis was performed with and without routing and checker logic to determine the additional area consumed. Based on synthesis results, verified against related work [8], we estimated area overhead for parts of our design, as follows: 2% for pipeline registers, per pipeline; 2% for routing, per pipeline; and 2% for the shared checker. Therefore, the additional die area powered on for timing speculation is up to 6%. In the experiments we conservatively assumed an overhead of

[1]Dynamic FIT rate for a FU is a function of unit-specific architecture vulnerability factor (AVF), raw soft error rate, utilization (for logic) or occupancy (for memory), and voltage. AVF is determined through simulation or testing. Raw soft error rate is a user-provided environmental factor. Utilization and occupancy are measured at run time. Voltage is selected by the optimizer. Each FU's AVF is not substantially affected by variation, so a simple analytical model is used to estimate FIT.

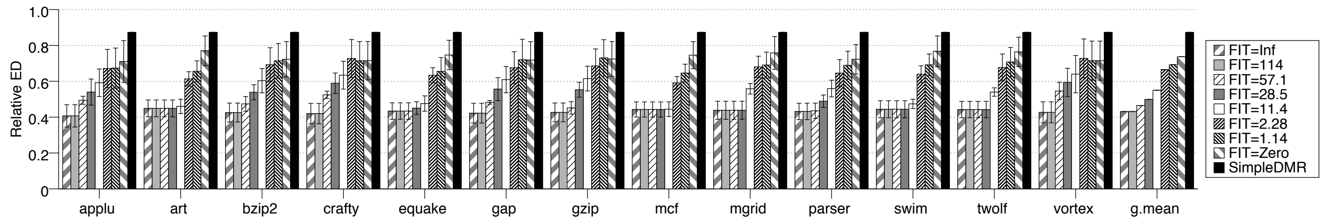


Figure 6: ED savings for different FIT targets. Different applications require different amounts of energy to achieve the same FIT target.

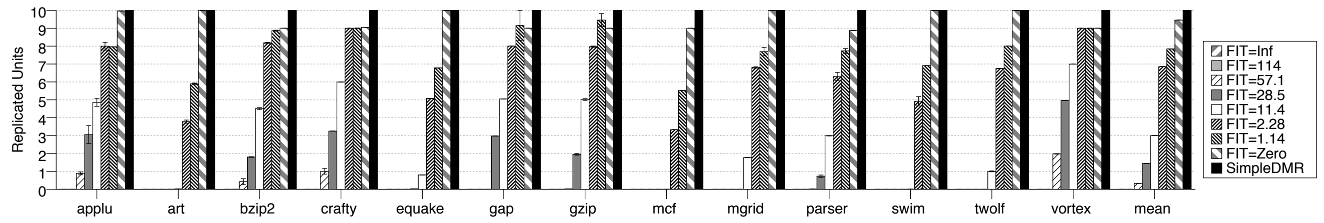


Figure 7: Average number of replicated FUs per benchmark for multiple FIT targets.

10%. The cycle time overhead is incurred due to the presence of multiplexing before pipeline registers and routing between pipelines. To estimate this impact synthesis was performed with and without routing logic. Depending on target die size, cycle time impact ranged between 10% and 15%. All of these overheads are accounted for in the energy evaluation.

The optimization algorithm described in Section 5.2 requires fewer than 1300 queries of the error analysis function, which translates into under 1300 forward evaluations of each ANN and a small amount of computation for the dynamic FIT rate. We use an optimization interval of 1 millisecond. We profile over one interval and then perform the search for the best voltages to use over the next interval. To share ANN hardware across an 8 core system, a decision must be made in 0.125ms, or 500K cycles at 4GHz. There are less than 1500 weights for all ANNs. Thus, there are less than 2 million products and sums to be computed. The complete optimization can be performed in the given time using 4 single-precision multipliers, 4 adders, and one logistic function.

7.2 Energy Reduction with FIT Targeting

We evaluate the energy reduction from FIT targeting and timing speculation compared to a configuration that uses full replication with no timing speculation (*StaticDMR*). We also compare to a lower overhead DMR with replication at core level that we refer to as *SimpleDMR*. The *SimpleDMR* does not have the overhead of routing and fine-grain checking needed for fine-grain redundancy allowing it to run at a 10% faster clock rate. We use the energy delay product (ED) a common metric to evaluate energy efficiency that accounts for both energy and execution time.

Figure 6 shows the reduction in ED relative to *StaticDMR* for all benchmarks, averaged across all dies, at FIT targets ranging from zero to unlimited. The higher the FIT rate we are willing to tolerate, the lower the energy delay relative to *StaticDMR*. For a high FIT (above 50, corresponding to MTBF of about 2×10^3 years) little replication is needed for most benchmarks, and power savings approach 60%. As the FIT target is lowered, replication is enabled more often, and power savings are less. A FIT target of 11.4 (MTBF=10^5 years) yields average power savings of about 50%. For very low FIT rate of $1.1 - 1.4$, savings are around 30%. Note that even in the extreme case in which no errors are allowed (FIT target is zero), the energy reduction from timing speculation alone is 24% compared to *StaticDMR*.

Compared to our baseline *StaticDMR*, the *SimpleDMR* has about 12% lower ED mainly due to the faster clock rate. Dynamic adaptation however more than makes up for the increase in cycle time, resulting in 10% lower ED than *SimpleDMR* even in the conservative case of zero FIT.

Some of the ED savings come from selective enabling of FU replication. Figure 7 shows the average number of FUs replicated for each benchmark at the various target FIT rates. For a FIT target of 11.4, replication is enabled for an average of 3 FUs across benchmarks. Replication varies significantly across benchmarks. For instance, for a FIT of 2.3 there is significant variation in average replication across benchmarks from 10 for *vortex* to 4 for *art*. This is due to variation in utilization and occupancy of various FUs. This shows the importance of dynamic adaptation of redundancy settings to match not only the FIT target but also the behavior of the application.

7.3 ANN Prediction Accuracy

An important factor in the performance of the energy optimization algorithm is the accuracy of the ANN predictions. In our experiments, the average ANN prediction error is less than 0.5% and the maximum prediction error is less than 5%. We also conduct the energy reduction experiments with a perfect predictor instead of the ANNs. We found that the average energy delay for the experiments with the ANN comes within 2% of that achieved with a perfect predictor.

8 Conclusions

This paper proposes a new approach to reliability management that allows *FIT targeting* in which the user or the system is allowed to specify an acceptable FIT target for individual cores or applications. We show that FIT targeting coupled with voltage tuning and timing speculation can result in significant energy savings compared to a static DMR architecture.

References

[1] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Design Automation Conference*, June 2003.

[2] T. Slegel, I. Averill, R.M., M. Check, B. Giamei, B. Krumm, C. Krygowski, W. Li, J. Liptay, J. MacDougall, T. McPherson, J. Navarro, E. Schwarz, K. Shum, and C. Webb, "IBM's S/390 G5 microprocessor design," *IEEE Micro*, vol. 19, no. 2, pp. 12–23, March/April 1999.

[3] S. Mitra, M. Zhang, N. Seifert, B. Gill, S. Waqas, and K. S. Kim, "Combinational logic soft error correction," in *International Test Conference*, November 2006.

[4] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *International Symposium on Microarchitecture*, December 2003, pp. 7–18.

[5] C. Weaver and T. Austin, "A fault tolerant approach to microprocessor design," in *Dependable Systems and Networks*, July 2001.

[6] B. Greskamp and J. Torrellas, "Paceline: Improving single-thread performance in nanoscale cmps through core overclocking," in *International Conference on Parallel Architectures and Compilation Techniques*, September 2007, pp. 213–224.

[7] K. Constantinides, O. Mutlu, and T. Austin, "Online design bug detection: RTL analysis, flexible mechanisms, and evaluation," in *International Symposium on Microarchitecture*, November 2008, pp. 282–293.

[8] B. F. Romanescu and D. J. Sorin, "Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults," in *International Conference on Parallel Architectures and Compilation Techniques*. ACM, 2008, pp. 43–51.

[9] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke, "The StageNet fabric for constructing resilient multicore systems," in *International Symposium on Microarchitecture*. IEEE Computer Society, 2008, pp. 141–151.

[10] S. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas, "EVAL: Utilizing processors with variation-induced timing errors," in *International Symposium on Microarchitecture*, November 2008, pp. 423–434.

[11] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith, "Configurable isolation: building high availability systems with commodity multi-core processors," *Computer Architecture News*, vol. 35, no. 2, pp. 470–481, 2007.

[12] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *International Symposium on Computer Architecture*. IEEE Computer Society, 2008, pp. 39–50.

[13] H. Jiang and M. Marek-Sadowska, "Power gating scheduling for power/ground noise reduction," in *Design Automation Conference*. ACM, 2008, pp. 980–985.

[14] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar, "An integrated quad-core Opteron processor," in *International Solid-State Circuits Conference*, February 2007, pp. 102–103.

[15] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *IEEE International Symposium on High-Performance Computer Architecture*, February 2008.

[16] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *International Symposium on Microarchitecture*. IEEE Computer Society, 2003.

[17] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," in *International Symposium on Computer Architecture*, June 2007.

[18] A. Biswas, N. Soundararajan, S. S. Mukherjee, and S. Gurumurthi, "Quantized AVF: A means of capturing vulnerability variations over small windows of time," in *IEEE Workshop on Silicon Errors in Logic - System Effects*. Stanford University, March 2009.

[19] R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Mitigating parameter variation with dynamic fine-grain body biasing," in *International Symposium on Microarchitecture*, December 2007, pp. 27–39.

[20] X. Liang, G.-Y. Wei, and D. Brooks, "ReVival: A variation-tolerant architecture using voltage interpolation and variable latency," *IEEE Micro*, vol. 29, no. 1, pp. 127–138, 2009.

[21] R. S. Amant, D. A. Jimenez, and D. Burger, "Low-power, high-performance analog neural branch prediction," in *International Symposium on Microarchitecture*. IEEE Computer Society, December 2008, pp. 447–458.

[22] R. McGowen, C. A. Poirier, C. Bostak, J. Ignowski, M. Millican, W. H. Parks, and S. Naffziger, "Power and temperature control on a 90-nm Itanium family processor," *Journal of Solid-State Circuits*, January 2006.

[23] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, K. Strauss, S. Sarangi, P. Sack, and P. Montesinos, "SESC Simulator," January 2005. [Online]. Available: http://sesc.sourceforge.net

[24] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A model of parameter variation and resulting timing errors for microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, February 2008.

[25] P. Ribeiro Jr. and P. Diggle, "geoR: A package for geostatistical analysis," *R-NEWS*, vol. 1, no. 2, 2001.

[26] R Development Core Team, "The R project for statistical computing," 2010. [Online]. Available: http://www.R-project.org

[27] "International Technology Roadmap for Semiconductors," 2009. [Online]. Available: www.itrs.net

[28] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *International Symposium on Computer Architecture*, June 2003.

[29] R. Teodorescu, B. Greskamp, J. Nakano, S. R. Sarangi, A. Tiwari, and J. Torrellas, "VARIUS: A model of parameter variation and resulting timing errors for microarchitects," in *Workshop on Architectural Support for Gigascale Integration*, June 2007.

[30] X. Li, S. Adve, P. Bose, and J. Rivers, "SoftArch: an architecture-level tool for modeling and analyzing soft errors," in *Dependable Systems and Networks*, June 2005, pp. 496–505.

[31] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Dependable Systems and Networks*, 2002, pp. 389–398.

[32] P. Hazucha, T. Karnik, J. Maiz, S. Walstra, B. Bloechel, J. Tschanz, G. Dermer, S. Hareland, P. Armstrong, and S. Borkar, "Neutron soft error rate measurements in a 90-nm CMOS process and scaling trends in SRAM from 0.25-μm to 90-nm generation," in *IEEE International Electron Devices Meeting*, December 2003.

[33] T. Miller and P. Urkedal. The open graphics project. [Online]. Available: http://opengraphics.org