CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2013; **25**:790–807
Published online 10 July 2012 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.2898

SPECIAL ISSUE PAPER

Runtime failure rate targeting for energy-efficient reliability in chip microprocessors

Timothy Miller [1], Nagarjuna Surapaneni [2] and Radu Teodorescu [1,*,†]

[1] *Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA*
[2] *Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH, USA*

SUMMARY

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. To ensure continued performance, growth of microprocessors requires development of powerful and energy-efficient solutions to reliability challenges. 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 chip multiprocessor by $30-60\%$ compared with static dual modular redundancy.

Received 19 May 2011; Revised 22 April 2012; Accepted 10 May 2012

KEY WORDS: microprocessor reliability; low power; timing speculation; soft errors

1. INTRODUCTION

Transistor scaling to minute sizes makes modern microprocessors increasingly prone to reliability problems. As transistors shrink and their integration increases, they become more vulnerable to both transient and permanent faults. To ensure the continued growth in chip performance, microprocessors must become more resilient to errors, requiring more hardware resources to be devoted to fault tolerance and mitigation. At the same time, power consumption is emerging as one of the most significant roadblocks to future technology scaling according to a recent report by the International Technology Roadmap for Semiconductors (ITRS) [1]. Power delivery and heat removal capabilities [2] are already limiting performance in microprocessors today and will continue to severely restrict performance in the future [3]. As a result, reliability solutions for future microprocessors must become much more energy efficient and must work within limited power budgets.

This paper presents a chip multiprocessor architecture that achieves reliable and energy-efficient operation by dynamically adapting the amount of error protection to the characteristics of each

*Correspondence to: Radu Teodorescu, Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA.
†E-mail: teodores@cse.ohio-state.edu

individual chip, its runtime behavior and the desired level of error resiliency. The ability of the system to dynamically adapt allows it to operate reliably within defined targets without waisting energy on high safety margins or over-provisioning.

The proposed 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. The ability to set a FIT target allows the same chip multiprocessor (CMP) to be deployed in systems with different reliability requirements such as laptops or servers. FIT targets can be changed at runtime, allowing different reliability goals to be assigned to individual applications with different requirements. An operating system kernel or financial application may require very high protection. On the other hand, less critical applications such as word processors and video players can tolerate the occasional error and therefore require only a moderate or low level of protection. Adapting the level of reliability protection to the goals of the system and the application improves the energy efficiency of the system.

The building block of the proposed multicore architecture is 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. The architecture also employs a novel mechanism for timing speculation that is used to recover from variation-induced timing errors. The timing speculation support also allows fine-grain voltage tuning that significantly reduces the energy overhead of the error protection mechanism.

The dynamic adaptation of the processor is controlled by an optimization algorithm implemented in firmware. The algorithm adjusts a range of parameters, including which functional units (FUs) are replicated and their supply voltage, to meet that target with minimum energy. The control system 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 with a system with static dual modular redundancy (DMR), our solution reduces the average energy delay product by 30% when no errors are allowed and up to 60% as the FIT target is relaxed. We implemented the building blocks for our architecture in a simple reduced instruction set computer processor. On the basis of the synthesis results, we find the area overhead to be about 4% and the impact on cycle time to be about 10% compared with static DMR.

This paper makes the following contributions:

- Introduces *FIT targeting* that allows the degree of error protection to vary dynamically to reduce energy usage.
- Presents an architecture that provides simultaneous protection against soft and timing errors and some hard errors.
- 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.

Section 2 provides some background on error classes and the implications of process variation. Section 3 discusses related work. Section 4 presents implementation details of the proposed architecture. Section 6 describes runtime control and optimization options for this architecture. Experimental setup is described in Section 7 and evaluation in Section 8. Section 9 concludes.

2. BACKGROUND

2.1. Error classes

The proposed architecture addresses three classes of errors. *Soft errors*, or single event upsets, occur as a result of particle strikes due to cosmic radiation or radiation originating from the chip packaging material. As technology scales, the soft error rate in chips is expected to increase as a result

DOI: 10.1002/cpe

of the increase in the number of transistors (increasing the likelihood of an error as a result of a strike) 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, and so on. *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.

2.2. Process variation

Process variation refers to deviations in transistor parameters beyond their nominal values and results from manufacturing difficulties in very small feature technologies [4].

Several transistor parameters are affected by variation. Of key importance are the threshold voltage (V_{th}) and effective gate length (L_{eff}). These directly impact a transistor's switching speed and leakage power. The higher the V_{th} and L_{eff} variation, the higher the variation in transistor speed across the chip. This slows down sections of the chip, resulting in slower processors, because the slower transistors limit the frequency of the whole processor. This also increases the likelihood of timing errors. Also, as V_{th} varies, transistor leakage varies across the chip. However, low-V_{th} transistors consume more power than high-V_{th} transistors save, resulting in significantly increased overall power consumption.

Overall, process variation makes both power consumption and susceptibility to timing errors unpredictable at design time. As technology scales and variation gets worse, higher safety margins (such as high supply voltage) will be needed to ensure reliable operation. Techniques that adapt to post-manufacturing characteristics and runtime behavior, such as the ones proposed in this work, will be needed to build energy-efficient systems.

3. RELATED WORK

3.1. Error detection, correction, and recovery

Several existing or proposed architectures deal with soft errors by replicating entire FUs. The IBM G5 [5] uses full replication in the fetch and execution units with a unified error-correcting code (ECC)-protected L1 cache. Others proposed replication and checking for soft errors at latch level [6]. 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 such as in Razor [7]. Another uses a simple checker that verifies execution of the main processor as in DIVA [8]. Timing speculation has been used to reduce voltage aggressively to save power [7] or to over-clock a processor to improve performance [9]. 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 mechanism [10] and using spare logic to replace faulty components [11, 12]. In Core Cannibalization [11], 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 [12], multiple processor pipelines are interconnected using crossbar switches after each pipeline stage, allowing rerouting 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.

DOI: 10.1002/cpe

3.2. Dynamic optimization and control

EVAL [13] 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. Although EVAL is efficient for this purpose, it has no capability to mitigate soft errors or hard failures.

EVAL's machine learning algorithm is adaptive to variation, but it is used as a means to speed up configuration parameter search. By contrast, our system uses machine learning to directly model the effects of parameter variation on power and probability of timing errors. Our architecture solves a different optimization problem that considers pipeline stage replication in addition to supply voltage and tolerates timing errors to further lower voltage.

Aggarwal *et al.* [14] 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 redundancies. 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 [15], authors present a reinforcement learning approach to schedule requests from multiple out-of-order processors competing for access to a single off-ship dynamic random access memory channel. In a circuit area no worse than a branch predictor, they enjoy a 22% boost in throughput over other cutting-edge schedulers. Bitirgen *et al.* applied an artificial neural net (ANN) to the problem of optimizing system resource allocation [16].

4. ARCHITECTURAL SUPPORT FOR RELIABILITY ADAPTATION

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.

4.1. Support for soft error detection

Figure 1 shows an overview of a pipeline pair on the basis of 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 multistage FUs, including integer and floating point arithmetic and logic units, 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

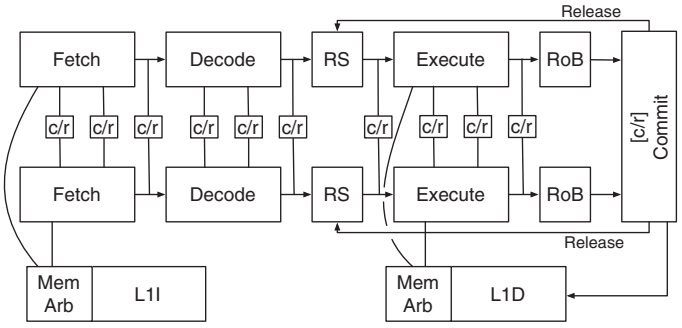

Figure 1. Architecture of the proposed *pipeline pair* for one core, with routing and checking logic at pipeline stage granularity. Mem Arb, arbitration logic; RS, reservation station; c/r, checking and routing logic; RoB, reorder buffer.

DOI: 10.1002/cpe

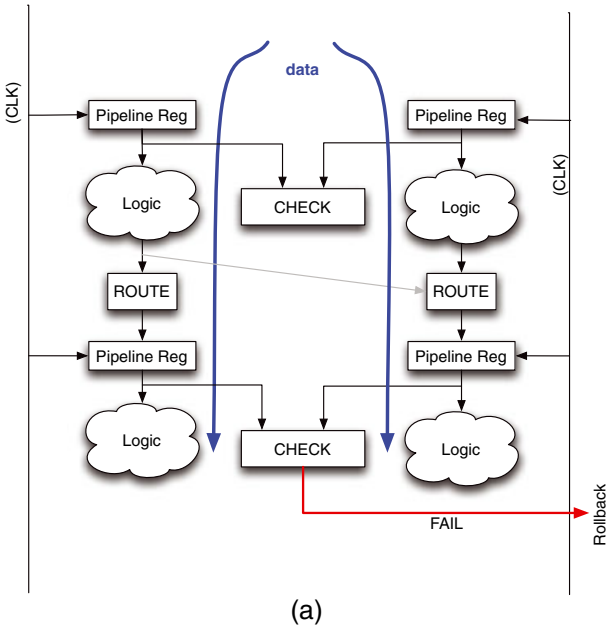

Figure 2. Replicated pipeline stage with routing and checker. Both main and shadow stages and enabled. Inputs to the next enabled shadow stage are routed from the previous main stage (shown by arrows labeled *data*).

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.

Figure 2 shows a pair of pipeline stages with replication fully enabled, and Figure 3 shows partial replication for detecting only timing errors.

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., arithmetic and logic unit, 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 [6] or transistor upsizing 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

DOI: 10.1002/cpe

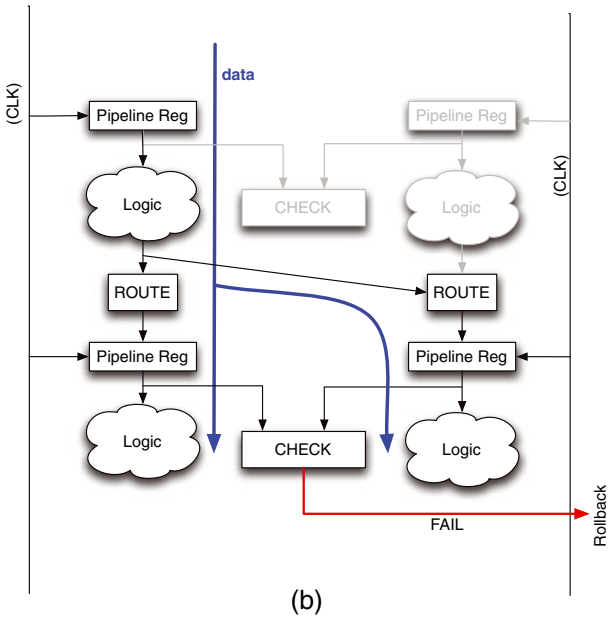

Figure 3. Replicated pipeline stage with routing and checker. Shadow stage is partially disabled (in gray). Inputs to the next enabled shadow stage are routed from the previous main stage (shown by arrows labeled *data*).

data streams from a shared L1. To ensure fairness, half of the cache ways (of set associativity) are reserved for each pipeline. Arbitration logic 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.

4.2. 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 4. 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.

Our implementation is different from previous work [7] 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.

4.3. 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 [11].

DOI: 10.1002/cpe

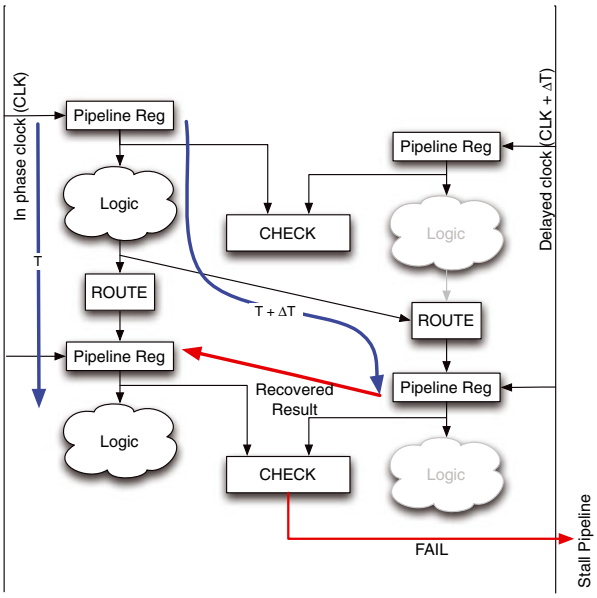

Figure 4. Main and shadow pipeline stages with timing speculation enabled. Only the shadow registers are turned on in the shadow pipeline.

4.4. *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 4. 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 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. Because checkers are small, hardening (transistor replication and upsizing) can be carried out with low overhead.

4.5. *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 [17].

DOI: 10.1002/cpe

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 [18] using off-chip regulators. To provide each FU with its own voltage, on-chip voltage regulators [19] must be used.

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

5. FAILURE RATE TARGETING

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 [20].

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, whereas 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 because their contribution to the total FIT rate is additive. The expected FIT for a core is the sum of the FIT for all its 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 [21, 22] has demonstrated that predicting AVF is possible and practical at runtime by examining a set of architectural parameters such as number of instructions executed per cycle, RoB utilization, branch mispredictions, reservation station utilization, instruction queue utilization, and so on. We use a similar approach to predict dynamic AVF, but at FU granularity.

5.1. 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) embedded in each FU [23]. The critical path replica is longer than the critical path of the unit, allowing detection of impending timing errors. Replication is automatically switched on and off on the basis of this sensor.

6. RUNTIME CONTROL SYSTEM

FIT targeting and timing speculation are controlled by a runtime optimization mechanism as shown in Figure 5. Each core pair in the system is first assigned a reliability (FIT) target by the manufacturer or user. The FIT target can change at runtime if the reliability goals for the system or application

DOI: 10.1002/cpe

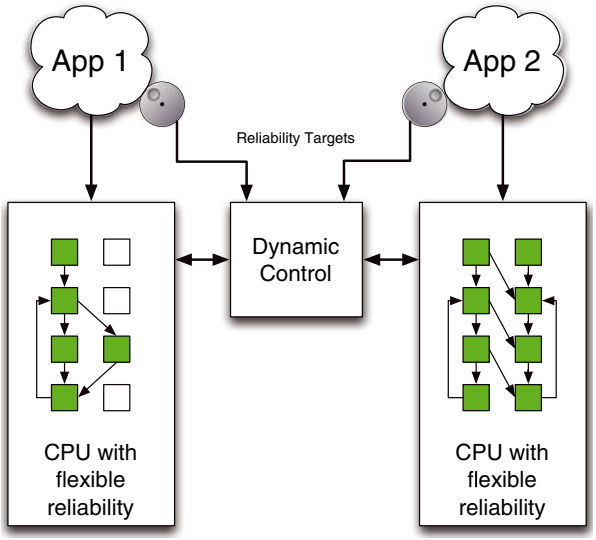

Figure 5. A high-level overview of the runtime control system. Reliability settings for each core pair change depending on the reliability targets.

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.

6.1. Machine learning-based modeling

Process variation results in different power and delay characteristics for each FU within each pipeline [23, 24]. These characteristics are difficult to predict and model analytically. To deal with this challenge, we use 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.

6.1.1. Artificial neural net architecture. To model energy on the basis of temperature, voltage, and utilization, we use three ANN architectures, shown in Figure 6. 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 adaptive linear element (ADALINE), 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 through an *activation function* that adds nonlinearity.

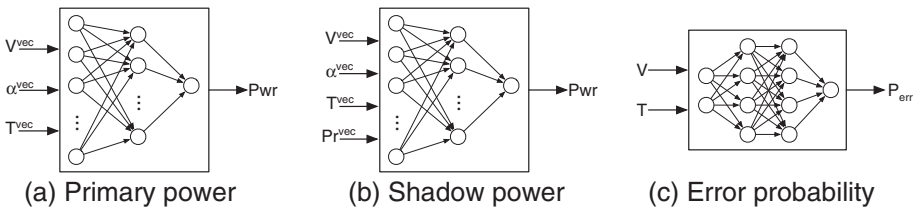

Figure 6. The architecture of artificial neural nets used for power and error probability prediction. (a) Primary power; (b) shadow power; and (c) error probability.

DOI: 10.1002/cpe

A popular activation function is the logistic function, $P(t) = \frac{1}{(1+e^{-t})}$, which maps $[-\infty, \infty]$ to $[0, 1]$. Additional layers are processed in the same way. *Backpropagation* is the process of training the ANN to model the desired function. Our model uses ANNs with the logistic function applied to the hidden layers, as usual, but applies a linear ($P(t) = t$) activation function to the outputs. This is because the power model needs to produce values greater than 1, and we found the error probability model to train more quickly and produce more accurate results this way.

Primary power ANNs (Figure 6(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 6(b)) predict power consumption for the shadow pipeline (P_s). There are five inputs for each FU: voltage, utilization, temperature, and binary values indicating replication (11_b for none, 01_b for full, and 10_b for pipeline register only). There are 10 nodes in one hidden layer and one output node.

Error probability ANNs (Figure 6(c)) predict raw probability of an error occurring on each cycle ($P(\mathrm{E})$). Because 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 $P(\mathrm{E})$, 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 $N_\mathrm{E} > 0$. Total energy is $(P_\mathrm{p} + P_\mathrm{s}) \times (1 + R_\mathrm{p}/C_\mathrm{m})$.

6.1.2. Artificial neural net training. Artificial neural nets are trained on-line by comparing predictions against measurements and adjusting weights to improve prediction. The training occurs in two phases. The first phase is initial training immediately after fabrication, where ANN weights are initialized to generally good values. The ANNs start out with random weights, and a range of benchmarks is executed for profiling purposes. Benchmarks are executed at all voltage levels, even those that may incur timing errors every cycle, because correct execution is not required. Measurements taken include temperatures, error counts, current, and utilization. For each measurement interval, ANNs are forward-evaluated, and when computed outputs disagree with desired outputs, backpropagation is performed, which adjusts ANN weights to reduce prediction error. This way, the ANNs come to model the power and delay characteristics of the processor affected by process variation.

Weights learned in the first training phase should result in reasonable prediction accuracy for a range of workloads beyond those in the training set. However, it would be good if the ANNs could adapt over time both to perform increasingly better on new workloads and to adapt to the effects of circuit aging. Therefore, the second phase of training is on-line training that occurs throughout the lifetime of the processor. Whenever predictions disagree with measurements, backpropagation can be performed to adjust the ANNs.

There are several approaches for implementing ANNs in hardware. In [25], a small, fast, low-power ANN is built from analog circuitry. Other alternatives include simple digital logic as in [15]. We give an estimate for the amount of hardware needed in our case in Section 8.3.

6.2. Runtime optimization system

The energy optimization given a FIT target is performed at regular intervals. Figure 7 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 are 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 [26].

DOI: 10.1002/cpe

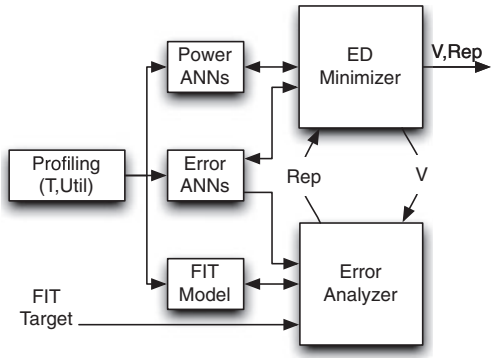

Figure 7. Runtime optimization system. FIT, failure in time; ANN, artificial neural net; ED, energy delay product.

```
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 8. Hill-climbing search for optimal voltages. ED, energy delay product; FU, functional unit.

For every interval, our objective is to find a set of configuration settings that (i) minimize energy or the energy delay product; (ii) prevent timing errors; and (iii) 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 8) 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 energy delay product (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 to get as close as possible to the FIT target without exceeding it. To do this, we apply a greedy algorithm that we call *best fit FIT first*. A FU is selected whose estimated FIT rate is closest to the difference between the target FIT and the current estimated total. Dynamic FIT rate for a FU is a function of unit-specific 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 runtime. Voltage is

DOI: 10.1002/cpe

selected by the optimizer. Each FU's AVF is not substantially affected by variation, so a simple analytical model is used to estimate FIT. 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.

7. EVALUATION METHODOLOGY

We use a modified version of the SESC cycle-accurate execution-driven simulator [27] to model a system similar to the Intel Core 2 Duo modified to support redundant execution. Table I summarizes the architecture configuration.

7.1. Variation model

Chip manufacturers seldom release measurements of process variation for current or future technologies. As a result, we rely on statistical models of variation (e.g., [23, 28–30]) driven by values projected by the ITRS [1]. In this paper, we use the VARIUS model [23, 30] to model variation in threshold voltage (V_{th}) and effective gate length (L_{eff}).

To model systematic variation, the chip is divided into a grid of points. Each grid point is given one value of the systematic component of the parameter, assumed to have a normal distribution with $\mu = 0$ and standard deviation σ_{sys}. Systematic variation is also characterized by a spatial correlation so that adjacent areas on a chip have roughly the same systematic component values. The spatial correlation between two points $\vec{x}$ and $\vec{y}$ is expressed as $\rho(r)$, where $r = |\vec{x} - \vec{y}|$. To determine how $\rho(r)$ changes from $\rho(0) = 1$ to $\rho(\infty) = 0$ as r increases, the spherical function is used. The distance, ϕ, at which the function converges to zero is when there is no significant correlation between two transistors.

Random variation is modeled with a normal distribution with $\mu = 0$ and standard deviation σ_{ran}. Because the random and systematic components are normally distributed and independent, their effects are additive, and the total standard deviation σ is $\sqrt{\sigma_{\text{ran}}^2 + \sigma_{\text{sys}}^2}$. In the model, V_{th} and L_{eff} have different values of σ.

Table I 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 [31] of R [32]. Resolution is 1/4M points per chip.

Table I. Summary of the architecture configuration.

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

DOI: 10.1002/cpe

7.2. Power and temperature models

To estimate power, we scale results given by popular tools using technology projections from ITRS [1]. We use SESC [27] to estimate dynamic power at a reference technology and frequency. In addition, we use the model from [30] to estimate leakage power for same technology. We use HotSpot [33] to estimate on-chip temperatures.

7.3. Timing and soft error models

We use the timing error model developed in [30]. 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 variations. The model has been validated with empirical data [34]. With this, we estimate the timing error probability for each FU of each chip at a range of supply voltages.

For soft errors, we use the approach in SoftArch [35]. We determine the raw soft error rate for 50-nm technology from [36]. FIT values for latch and combinational logic chain were also extracted from [36]. We scale that to 32 nm using the predictions from [37]. On the basis of the transistor count for the Core 2 Duo floorplan, we estimate the number of transistors in latches and combinational logic in each FU. On the basis of 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 [21]. 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.

7.4. 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.

8. 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.

We begin by looking at the potential for energy savings from aggressively lowering the supply voltage of selected FUs while tolerating timing errors. Figure 9 shows energy consumption for a typical FU protected only against timing errors while running a section of a benchmark at different supply voltages at the same clock rate. The ideal energy line shows how energy would scale if we could lower voltage with no impact on timing. As voltage is lowered, total power consumption decreases, resulting in lower energy. When the voltage drops below the safe margin, replication is enabled to detect and correct timing errors.

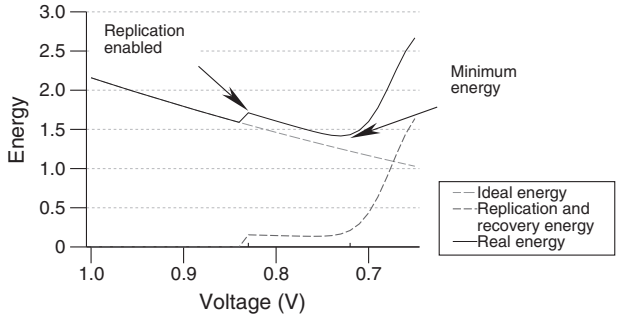

Figure 9. Energy as a function of voltage and functional unit with timing speculation only.

DOI: 10.1002/cpe

The replication and recovery energy line accounts for the increase in power consumption due to replication (for this unit, enabled around 0.84 V) and the time penalty incurred during error recovery. The penalty for error recovery increases sharply as the voltage is lowered beyond 0.7 V. Real energy includes both ideal and penalty energies. We notice a sharp increase in energy as replication is enabled, followed by a continued decrease with voltage as long as the error rate is tolerable. The lowest energy point occurs around 0.73 V.

Figure 10 shows the energy consumption versus voltage for units with full replication. Here, there is no discontinuity because replication is always enabled.

The voltage level with the lowest energy depends on a several factors, including the variation profile of the FU and its sensitivity to timing errors. Ideal voltage is also dependent on the computed workload. Thus, supply voltage for each FU can change significantly over time as utilization and temperature vary.

8.1. Energy reduction with failure in time targeting

We evaluate the energy reduction from FIT targeting and timing speculation compared with a configuration that uses full replication with no timing speculation (*StaticDMR*). We also compare with 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 ED, a common metric to evaluate energy efficiency that accounts for both energy and execution time.

Figure 11 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 energy-delay reduction approaches 60%. As the FIT target is lowered, replication is enabled more often, and energy savings are less. A FIT target of 11.4 (MTBF $= 10^5$ years) yields average energy 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 with StaticDMR.

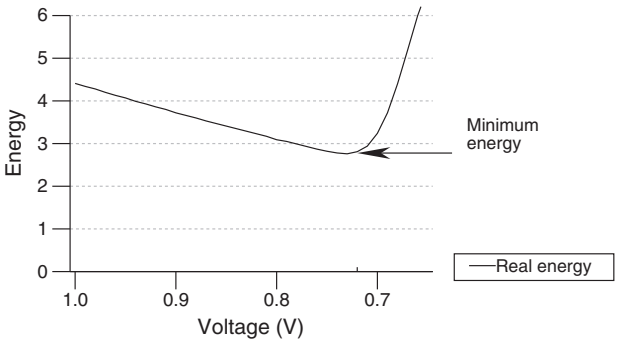

Figure 10. Energy as a function of voltage, for a functional unit with full replication.

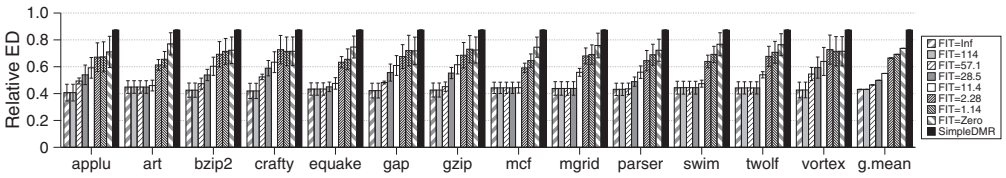

Figure 11. Energy delay product (ED) savings for different failure in time (FIT) targets. Different applications require different amounts of energy to achieve the same FIT target. DMR.

DOI: 10.1002/cpe

Compared with 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 12 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 three 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.

8.2. Artificial neural net prediction accuracy

An important factor in the performance of the energy optimization algorithm is the accuracy of the ANN predictions. Figures 13 and 14 illustrate prediction accuracy for the shadow power ANN as a function of the number of hidden nodes, across all benchmarks. Figure 13 shows the average prediction error over the entire test set, whereas Figure 14 shows the worst case prediction error. The shadow power ANN models a nonlinear function, so there is a large improvement when going from no hidden nodes to having at least one. Notice, however, that there is little improvement when going from four to eight hidden nodes. Increasing the number of hidden nodes to 12 shows another significant improvement because there is now one hidden node for each functional unit, which allows the ANN to better model the behavior of each unit. With 12 hidden nodes, 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.

8.3. Overheads

8.3.1. Power, area and timing. The proposed architecture introduces some timing, power, and area overheads. To estimate it, we synthesized the Open Graphics Project HQ microcontroller [38] for

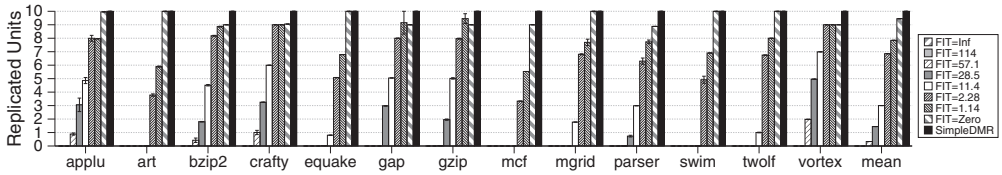

Figure 12. Average number of replicated functional units per benchmark for multiple failure in time (FIT) targets. DMR.

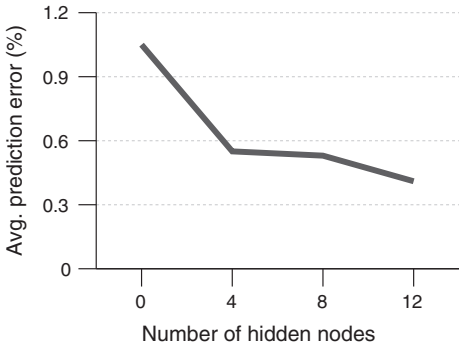

Figure 13. Average prediction error (% deviation from real power) for shadow power artificial neural net versus number of hidden nodes.

DOI: 10.1002/cpe

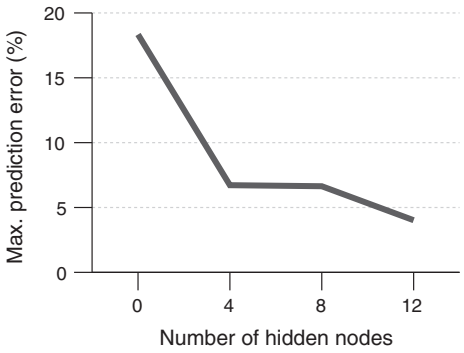

Figure 14. Worst-case prediction error (% deviation from real power) for shadow power artificial neural net versus number of hidden nodes.

Xilinx Spartan 3 FPGA. The synthesis was performed with and without routing and checker logic to determine the additional area consumed. On the basis of the synthesis results, verified against related work [11], 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 10%. The cycle time overhead is incurred because of 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.

8.3.2. Runtime optimization overhead. The optimization algorithm described in Section 6.2 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 ms. 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 0.125 ms, or 500 K cycles at 4 GHz. 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 four single-precision multipliers, four adders, and one logistic function.

9. CONCLUSION

This paper proposes a new approach to microprocessor reliability management that achieves reliable and energy-efficient operation by dynamically adapting the amount of error protection to the characteristics of individual chips (to account for the effects of process variation), their runtime behavior (to account for workload variability), and the desired level of error resiliency. The ability of the system to dynamically adapt allows it to operate reliably within defined targets without wasting energy on high safety margins or over-provisioning. We also show that a machine learning-based dynamic control mechanism performs the runtime optimization required by our system accurately and quickly.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under grant CNS-0403342 and by an allocation of computing time from the Ohio Supercomputer Center.

REFERENCES

1. International technology roadmap for semiconductors, 2009.
2. McGowen R, Poirier C, Bostak C, Ignowski J, Millican M, Parks W, Naffziger S. Power and temperature control on a 90-nm Itanium family processor January 2006; **41**(1):229–237.

DOI: 10.1002/cpe

3. Torrellas J. Architectures for extreme-scale computing. *IEEE Computer* November 2009; **42**:28–35.
4. Borkar S, Karnik T, Narendra S, Tschanz J, Keshavarzi A, De V. Parameter variations and impact on circuits and microarchitecture. In *Design Automation Conference*, 2003.
5. Slegel TJ, Averill I, RM, Check M, Giamei B, Krumm B, Krygowski C, Li W, Liptay J, MacDougall J, McPherson T, Navarro J, Schwarz E, Shum K, Webb C. IBM's S/390 G5 microprocessor design. *Micro, IEEE* March/April 1999; **19**(2):12–23.
6. Mitra S, Zhang M, Waqas S, Seifert N, Gill B, Kim KS. Combinational logic soft error correction. *IEEE International Test Conference, ITC '06* 2006:1–9. DOI: 10.1109/TEST.2006.297681. (Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4079359&isnumber=4042774).
7. Ernst D, Kim NS, Das S, Pant S, Rao R, Pham T, Ziesler C, Blaauw D, Austin T, Flautner K, Mudge T. Razor: a low-power pipeline based on circuit-level timing speculation. *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-36* 2003:7–18. DOI: 10.1109/MICRO.2003.1253179.
8. Weaver C, Austin TM. A fault tolerant approach to microprocessor design. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (Formerly: FTCS) (DSN '01)*. IEEE Computer Society: Washington, DC, USA, 2001; 411–420.
9. Greskamp B, Torrellas J. Paceline: improving single-thread performance in nanoscale CMPS through core overclocking. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT '07)*. IEEE Computer Society: Washington, DC, USA, 2007; 213–224, DOI: 10.1109/PACT.2007.52 (Available from: http://dx.doi.org/10.1109/PACT.2007.52).
10. Constantinides K, Mutlu O, Austin T. Online design bug detection: RTL analysis, flexible mechanisms, and evaluation. *International Symposium on Microarchitecture* 2008:282–293.
11. Romanescu BF, Sorin DJ. Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults. In *Pact '08: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. ACM: New York, NY, USA, 2008; 43–51, DOI: http://doi.acm.org/10.1145/1454115.1454124.
12. Gupta S, Feng S, Ansari A, Blome J, Mahlke S. The stagenet fabric for constructing resilient multicore systems. In *International Symposium on Microarchitecture*. IEEE Computer Society, 2008; 141–151.
13. Sarangi S, Greskamp B, Tiwari A, Torrellas J. EVAL: Utilizing processors with variation-induced timing errors. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture (MICRO 41)*. IEEE Computer Society: Washington, DC, USA, 2008; 423–434, DOI: 10.1109/MICRO.2008.4771810 (Available from: http://dx.doi.org/10.1109/MICRO.2008.4771810).
14. Aggarwal N, Ranganathan P, Jouppi NP, Smith JE. Configurable isolation: building high availability systems with commodity multi-core processors. *SIGARCH Computer Architecture News* 2007; **35**(2):470–481. DOI: http://doi.acm.org/10.1145/1273440.1250720.
15. Ipek E, Mutlu O, Martínez JF, Caruana R. Self-optimizing memory controllers: a reinforcement learning approach. In *International Symposium on Computer Architecture*. IEEE Computer Society, 2008; 39–50.
16. Bitirgen R, Ipek E, Martinez JF. Coordinated management of multiple interacting resources in chip multiprocessors: a machine learning approach. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture (MICRO 41)*. IEEE Computer Society: Washington, DC, USA, 2008; 318–329, DOI: 10.1109/MICRO.2008.4771801. (Available from: http://dx.doi.org/10.1109/MICRO.2008.4771801).
17. Jiang H, Marek-Sadowska M. Power gating scheduling for power/ground noise reduction. In *Design automation conference*. ACM: New York, NY, USA, 2008; 980–985, DOI: http://doi.acm.org/10.1145/1391469.1391716.
18. Dorsey J, Searles S, Ciraula M, Johnson S, Bujanos N, Wu D, Braganza M, Meyers S, Fang E, Kumar R. An integrated quad-core Opteron processor. *International Solid-state Circuits Conference*, 2007; 102–103.
19. Kim W, Gupta M, Wei G-Y, Brooks D. System level analysis of fast, per-core DVFS using on-chip switching regulators. *IEEE International Symposium on High-performance Computer Architecture*, Salt Lake City, UT, 2008; 123–134.
20. Mukherjee SS, Weaver C, Emer J, Reinhardt SK, Austin T. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *International Symposium on Microarchitecture*. IEEE Computer Society: Washington, DC, USA, 2003; 29–39.
21. Walcott KR, Humphreys G, Gurumurthi S. Dynamic prediction of architectural vulnerability from microarchitectural state. *International Symposium on Computer Architecture*, San Diego, California, USA, 2007; 516–527.
22. Biswas A, Soundararajan N, Mukherjee SS, Gurumurthi S. Quantized AVF: a means of capturing vulnerability variations over small windows of time. *IEEE Workshop on Silicon Errors in Logic - System Effects*, Stanford University, 2009 March.
23. Teodorescu R, Nakano J, Tiwari A, Torrellas J. Mitigating parameter variation with dynamic fine-grain body biasing. *International Symposium on Microarchitecture* 2007:27–39.
24. Liang X, Wei GY, Brooks D. Revival: a variation-tolerant architecture using voltage interpolation and variable latency. *IEEE Micro* 2009; **29**(1):127–138. DOI: http://doi.ieeecomputersociety.org/10.1109/MM.2009.13.
25. Amant RS, Jimenez DA, Burger D. Low-power, high-performance analog neural branch prediction. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture (MICRO 41)*. IEEE Computer Society: Washington, DC, USA, 2008; 447–458, DOI: 10.1109/MICRO.2008.4771812 (Available from: http://dx.doi.org/10.1109/MICRO.2008.4771812).

26. McGowen R, Poirier CA, Bostak C, Ignowski J, Millican M, Parks WH, Naffziger S. Power and temperature control on a 90-nm Itanium family processor. *Journal of Solid-State Circuits* January 2006; **41**(1):229–237.
27. Renau J, Fraguela B, Tuck J, Liu W, Prvulovic M, Ceze L, Strauss K, Sarangi S, Sack P, Montesinos P. SESC simulator, 2005. (Available from: http://sesc.sourceforge.net).
28. Liang X, Brooks D. Mitigating the impact of process variations on processor register files and execution units. In *International Symposium on Microarchitecture*. IEEE Computer Society, 2006; 504–514.
29. Marculescu D, Talpes E. Variability and energy awareness: a microarchitecture-level perspective. *Design Automation Conference*, 2005; 11–16.
30. Sarangi SR, Greskamp B, Teodorescu R, Nakano J, Tiwari A, Torrellas J. VARIUS: a model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing* 2008; **21**(1):3–13.
31. Ribeiro Jr PJ, Diggle PJ. geoR: a package for geostatistical analysis. *R-NEWS* 2001; **1**(2):14–18. (Available from: http://cran.R-project.org/doc/Rnews).
32. R Development Core Team. *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, 2006. (Available from: http://www.R-project.org).
33. Skadron K, Stan MR, Huang W, Velusamy S, Sankaranarayanan K, Tarjan D. Temperature-aware microarchitecture. *International Symposium on Computer Architecture* 2003:2–13.
34. Teodorescu R, Greskamp B, Nakano J, Sarangi SR, Tiwari A, Torrellas J. VARIUS: a model of parameter variation and resulting timing errors for microarchitects. *Workshop on Architectural Support for Gigascale Integration*, San Diego, USA, 2007.
35. Li X, Adve SV, Bose P, Rivers JA. SoftArch: an architecture-level tool for modeling and analyzing soft errors. In *Proceedings of the International Conference on Dependable Systems and Networks, DSN 2005*, 28 June-1 July 2005; 496–505, DOI: 10.1109/DSN.2005.88. (Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1467824&isnumber=31476).
36. Shivakumar P, Kistler M, Keckler SW, Burger D, Alvisi L. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings of the International Conference on Dependable Systems and Networks, DSN 2002*, 2002; 389–398, DOI: 10.1109/DSN.2002.1028924. (Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1028924&isnumber=22107).
37. Hazucha P, Karnik T, Maiz J, Walstra S, Bloechel B, Tschanz J, Dermer G, Hareland S, Armstrong P, Borkar S. Neutron soft error rate measurements in a 90-nm cmos process and scaling trends in sram from 0.25-m to 90-nm generation. *International Electron Devices Meeting* 2003:523–526. DOI: 10.1109/IEDM.2003.1269336.
38. Miller T, Urkedal P. (Available from: http://opengraphics.org).