

# Flexible Redundancy in Robust Processor Architecture

Timothy Miller<sup>s</sup>, Nagarjuna Surapaneni<sup>#</sup>, Radu Teodorescu<sup>s</sup> and Joanne Degroat<sup>#</sup>

<sup>s</sup>Department of Computer Science and Engineering

<sup>#</sup>Department of Electrical and Computer Engineering

The Ohio State University

{millerti,teodores}@cse.ohio-state.edu, {nagarjun,degroat}@ece.osu.edu

## Abstract

This paper proposes a reliable processor architecture that dynamically adapts the amount of protection to the characteristics of an individual chip and its runtime behavior. This architecture uses fine-grain redundancy, voltage scaling and timing speculation to adapt to variation and tolerate timing, soft and hard errors. The goal is to provide reliability with a minimum of resources. Our evaluation shows dynamic adaptation of voltage and redundancy can reduce the energy delay product (ED) by up to 19% compared to a static architecture with the same error protection.

## 1. Introduction

Modern microprocessors have scaled transistors to minute sizes, making chips less reliable and their performance and power consumption less predictable and highly variable. To ensure continued improvement in chip performance, new solutions must be developed to deal with these challenges in resource-efficient ways.

This paper presents 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 either independently (with each running a separate thread) or in concert (with both running the same thread and checking results at the end of each pipeline stage). Redundancy can also be enabled selectively, at the pipeline stage level, to allow targeted protection against errors at reduced cost. This architecture also employs timing speculation for mitigation of variation-induced timing errors and allows fine-grain voltage scaling to reduce the power overhead of the error protection and even allow for energy savings.

In this preliminary work, we focus primarily on energy optimization. We first show that fine-grain voltage scaling coupled with selective redundancy and timing speculation results in 18% reduction in energy delay product (ED) compared to a baseline with no replication and no timing speculation. We also consider a scenario in which full replication is enabled to provide maximum protection against soft errors for mission critical applications. To reduce the resulting high overhead we use timing speculation to dynamically lower supply voltage. This results in ED savings of up to 19% compared to a static redundant architecture with no timing speculation.

In this paper we make the following contributions:

- We propose a flexible robust architecture that can trade off error protection for energy consumption.
- We describe and evaluate two scenarios in which energy consumption is reduced by dynamically tailoring the amount of protection to specific chips and phases of an application.

This paper is organized as follows. Section 2 provides some background to this work. Section 3 presents the details of the proposed architecture. Section 4 describes two runtime optimization scenarios. The experimental setup is described in Section 5 and the evaluation is presented in Section 6.

## 2. Background

This section provides background on technology issues that result in the types of errors and variation addressed in this work.

### 2.1. Error Classes

The architecture proposed in this paper addresses three classes of errors which we summarize briefly in this section.

**Soft errors** or single event upsets (SEU) occur as a result of particle strikes due to cosmic radiation or radiation originating in chip packaging material. As technology scales, the soft error rate in chips is expected to increase as a result of the increase in the number of transistors and the lower operating voltages. A number of existing and proposed architectures deal with soft errors by replicating entire functional units. For instance, the IBM G5 processor [24] uses full replication in its fetch and execution units with a unified L1 cache protected by ECC. Other work has proposed replication and checking for soft errors at latch level [17]. Fine-grain replication is appealing because it allows targeted protection of only the sections or paths in a chip that are deemed most vulnerable. However, dynamically enabling/disabling replication at such fine granularity (latch level) would make control very complex and costly. In this architecture we use replication at the functional unit level.

**Timing errors** occur when propagation delay through any exercised path in a pipeline stage exceeds the clock cycle period. Timing errors can have multiple causes including variation in threshold or supply voltages, circuit degradation as a result of aging, high temperature, etc. 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 [5]. Another is to use a simple checker that verifies execution of the main core like in DIVA [27]. Timing speculation has been used before to reduce voltage aggressively to save power [5] or to overclock a processor to improve performance [6]. We use timing speculation selectively and dynamically for some units to reduce the overhead of replication and also to save power over the baseline architecture without replication.

**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 process variation. Previous work that deals with hard faults has proposed mechanisms for efficient detection of hard errors using the processor's built in self-test (BIST) mechanism [3] and methods for using spare logic to replace faulty components [21, 8]. In Core Cannibalization [21], processing pipelines are arranged in triples, where two pipelines are used for execution, while the third is used for spare parts at pipeline stage granularity. In StageNet [8], several processor pipelines are grouped into a network and interconnected using crossbars. As logic units fail, performance degrades gracefully, as stages can be shared between different pipelines. The routing logic introduces variable pipeline latency, requiring additional logic to make up for the loss of result forwarding. Our proposed design

groups pipelines into pairs, with simple two-way routing logic that minimizes the impact on the logical structure of the processors. When no functional units have failed, it is possible to run the two pipelines independently (for performance), share pipeline registers (for timing speculation), run in concert (to catch soft errors), or share functional units (to mask hard faults).

## 2.2. Process Variation

Process variation refers to deviation of transistor parameters from their nominal values and results from manufacturing difficulties in very small feature technologies [1].

Several transistor parameters are affected by variation. Of key importance are the threshold voltage ( $V_{th}$ ) and the effective gate length ( $L_{eff}$ ). These parameters 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, since the slower transistors end up determining 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.

Chip manufacturers do not generally release measurements of process variation for current or future technologies. As a result, we rely on statistical models of variation (e.g., [13, 15, 22, 26]) driven by values projected by the ITRS [9]. In this paper, we use the VARIUS model [22, 26].

To model systematic variation, the chip is divided into a grid. 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  $\sigma_{sys}$ . Systematic variation is also characterized by spatial correlation, so that adjacent areas on a chip have roughly the same systematic component values.

Random variation occurs at the level of individual transistors. It is modeled analytically with a normal distribution with  $\mu = 0$  and standard deviation  $\sigma_{ran}$ . Since the random and systematic components are normally distributed and independent, their effects are additive, and the total  $\sigma$  is  $\sqrt{\sigma_{ran}^2 + \sigma_{sys}^2}$ .  $V_{th}$  and  $L_{eff}$  have different values of  $\sigma$ .

## 3. Flexible Redundant Architecture

This section details the proposed flexible redundant architecture. We present both an in-order and an out-of-order implementation. This architecture has all of its pipeline stages replicated. Routing and configuration logic allows each core to be configured to run either as independent pipelines or in concert (running the same thread and checking results at the end of each stage). The replication occurs at the pipeline stage level and allows redundant computation to target the less reliable or slower sections of the chip.

### 3.1. Support for Soft Error Detection

Figure 1 shows the in-order implementation of a “pipeline pair,” based on a four-stage MIPS pipeline, where each stage is replicated. One pipeline is always enabled; we’ll refer to it as the *main* pipeline. The second, *shadow*, pipeline can have some of its stages selectively enabled. 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 will receive their inputs from the previous stage of the main pipeline. At each pipeline stage, computation results and control signals are forwarded to checker units (one for each pipeline stage). The 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.

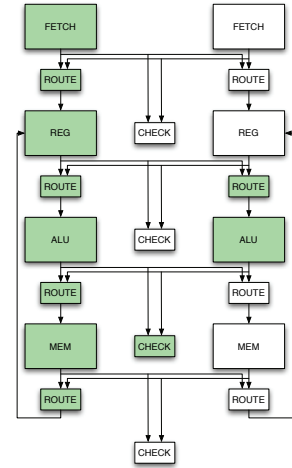
The scaling of the in-order design to an out-of-order superscalar version is relatively straightforward. We base our architecture on the Intel Core architecture. The fetch, decode and rename stages are replicated and their outputs verified by checkers much like in the in-order design. The reservation 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, Store). The RS outputs corresponding to each compute unit are verified by separate checkers. The same instruction is issued by main and shadow RS’s simultaneously. In the following cycle, each checker compares the issued instructions. Likewise, one cycle later, the compute unit’s result has been latched into pipeline registers and is being loaded into the re-order buffer (RoB). If replication is enabled for that unit, then the checker compares results on this cycle.

The RoB retires one or more instructions per cycle, in program order. The retire process commits memory writes, commits register writes to architectural registers, and frees resources associated with pending instructions. The commit is a two cycle pipelined process: In the first cycle the retiring instruction is dequeued from the both the main and shadow RoBs and checked for integrity. The second cycle is dedicated to actually retiring the instruction. This is done to avoid having to commit any unverified data to memory.

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. In replicated mode, both pipelines fetch the same instructions and data from the L1. In independent mode, the two pipelines will fetch separate instruction and data streams from a shared L1. To ensure fairness, half of the cache ways (of set associativity) can be reserved for each pipeline.

### 3.2. Support for Timing Speculation

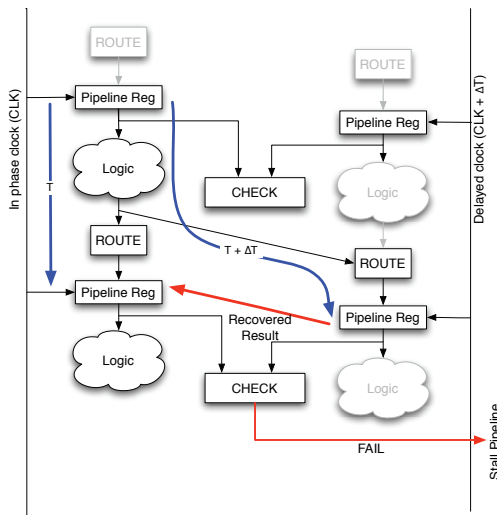
The proposed architecture can also be configured to employ timing speculation, for the purpose of minimizing energy consumption, which requires that timing errors be caught and corrected. This is achieved by selectively enabling only the pipeline registers of the shadow pipeline on a slightly delayed clock (while maintaining the same clock frequency as the main pipeline) similar to the approach in Razor [5]. Using the routing logic, the computation results of



**Figure 1.** In-order architecture with selective redundancy. The shaded blocks are active.

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 ( $\Delta T$ ) gives extra time to the signals propagating through the main pipeline’s stage ( $T$ ). The 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  $\Delta T$  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. The combinatorial logic in the shadow pipeline is disabled to save power.

This mode has the advantage of allowing voltage to be lowered to the point where the probability of a timing error is small but non-zero. As long as the recovery penalty is small, energy usage continues to decrease even as error rate increases, up to a point.



**Figure 2.** Main and shadow pipeline stages are out of phase on skewed clock. Only shadow registers are enabled in the shadow pipeline.

### 3.3. Support for Mitigation of Hard Faults

This architecture can also mask hard faults. Should a functional unit fail, its twin can be shared between the two pipelines. This requires that both pipelines be run on the same clock with no skew.

### 3.4. Error Recovery

The error recovery protocol depends on the type of error the system is configured to capture. When in full replication mode, the system is configured to detect soft errors. In this mode, when an error is detected, the checker triggers a pipeline flush followed by a re-execution, similar to a branch mispredict. 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; the solution is to trust the shadow pipeline register of the unit that experienced the error, just that one time, playing the odds that a soft error will not strike the same instruction in the same logic the second time through.

When the architecture is configured to detect timing errors, the results stored in the main pipeline registers are compared to those stored in the shadow pipeline registers. The comparison takes place in the cycle following the computation. When the results disagree we assume a timing error has occurred. At this point, a pipeline stall

signal is asserted. The pipeline registers in the shadow pipeline have extra time to latch the results of the previous stage and will therefore be 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. In the following cycle, execution can resume with the faulty result replaced by the correct result in the main pipeline register. In an in-order architecture, the resulting penalty for a timing error is about two cycles. In an out-of-order architecture, the penalty may be hidden.

### 3.5. Configuration Options

The primary difference between the Soft Error and the Timing Speculation configurations is that in the former, replication of both combinatorial logic and pipeline registers is always enabled for all units, while for the latter, only shadow pipeline registers are selectively enabled. Both employ clock skew for the shadow pipeline. Also, the two configurations employ different error-recovery mechanisms.

**Routing and checking** – The exact set of routing options depends on whether the pipeline is in-order or out-of-order. The routing configuration for a functional unit pair selects which block of combinatorial logic feeds each pipeline register. For the out-of-order design, this includes options to allow sharing of functional units for hard fault masking. Additionally, each pair of functional units 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 functional unit and each pipeline register can be enabled (power gated) separately. Power gating is a technique for efficiently cutting both leakage and dynamic power consumption by disconnecting idle blocks from the power grid. This technique has been extensively studied and can be implemented efficiently at coarse (pipeline stage) granularity [10].

**Voltage selects** – As part of a strategy to minimize energy consumption we allow different functional units 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 functional unit with its own supply voltage one option is to have only two or three voltage levels. Each functional unit (and its associated pipeline register) selects among those. For two or three voltage levels, off-chip voltage regulators are sufficient. Chips in production today commonly use several voltage domains [4] using off-chip regulators. For more voltage levels, voltage interpolation can be employed at low cost with a mechanisms similar to ReVival [14]. In order to provide each functional unit with its own voltage, on-chip voltage regulators [11] have to be used. This is because the number of pins in the microprocessor packaging limits the number of voltage lines that can be supplied from off-chip regulators.

**Clock controls** – There are two PLL circuits for each pipeline pair, and each PLL produces a configurable clock signal, along with a phase-delayed clock with configurable delay. Each pipeline in the pair can be configured to run on one of these four clock signals. This provides the means to run the pipelines on the same clock, independent clocks, or in timing speculation mode.

## 4. System-level Optimizations

Each application has its own requirements for reliability, performance, and priority. The processor as a whole should be config-

ured to use minimum energy to meet the requirements. The control mechanism must handle combinations of each of the available optimization targets (speed, power, soft errors, hard faults). In this paper, we address primarily power optimization.

#### 4.1. Energy Reduction with Timing Speculation

If soft error protection is not desired, selective replication of just the shadow latches can be used for timing speculation. Voltage is lowered – on a per unit basis or for a group of units – 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. We determine whether or not replication and checking are required, using a special circuit path (called a critical path replica – CPR) embedded in each functional unit [26]. 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. Additionally, when voltage is lowered, replication is summarily switched on and then allowed to switch off a few cycles later, in accordance with the sensor reading.

This leaves us with only voltage levels to optimize. Providing each functional unit with its own voltage rail would be ideal [14], but using fewer rails may require less hardware. We anticipate that a single rail would be limited by the functional unit with the highest error rate and therefore not achieve much savings. Functional unit speeds tend to come in clusters (depending on variation), so adding a second rail should result in a more significant energy reduction; the slowest unit would dictate the highest rail, while the second rail would be used for a number of faster units. Adding additional rails should see additional savings.

#### 4.2. Energy Reduction in Full Replication

For mission critical systems and applications, a high degree of protection against soft errors is needed. In this case, full replication will be enabled in the shadow pipeline. Such a configuration will incur a significant power overhead. To reduce this overhead we use timing speculation to aggressively lower supply voltage. No extra hardware needs to be enabled since full replication is already on.

### 5. Evaluation Methodology

We use a modified version of the SESC cycle-accurate execution-driven simulator [19] to model a system similar to the Intel Core 2 Duo modified to support redundant execution. Table 1 summarizes the architecture configuration. In the following, we discuss the different parts of our infrastructure.

Overall: Core 2 Duo-like processors Technology: 32nm, 4GHz (nominal) Branch prediction: 4K-entry BTB, 12-cycle penalty 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: 1 MB
Die size: $195mm^2$ ; $V_{DD}$ : 0.6-1V (default is 1V) Number of dies per experiment: 100 $V_{th}$ : $\mu$ : 250mV at 60°C $\sigma/\mu$ : 0.12 $\phi$ (fraction of chip’s width): 0.5

**Table 1.** Summary of architectural parameters.

### 5.1. Variation Model Parameters

To model variation, we use the VARIUS model [22, 26] to generate  $V_{th}$  and  $L_{eff}$  variation maps. 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  $\mu$ ,  $\sigma$ , and  $\phi$ . To generate each map, we use the geoR statistical package [20] of R [18]. We use a resolution of 1/4M points per chip.

### 5.2. Power, Temperature and Error Models

To estimate power, we scale the results given by popular tools using technology projections from ITRS [9]. We use SESC, augmented with dynamic power models from Wattch [2], to estimate dynamic power at a reference technology and frequency. In addition, we use the leakage model from [22] to estimate leakage power at the same reference technology. Then, we obtain ITRS’s scaling projections for the per-transistor dynamic power-delay product, and for the per-transistor static power. With these two factors, keeping the number of transistors constant as we scale, we can estimate the dynamic and leakage power for the scaled technology and frequency relative to the reference values.

We use HotSpot [23] to estimate on-chip temperatures. To do so, we use the iterative approach of Su *et al.* [25]: the temperature is estimated based on the current total power; the leakage power is estimated based on the current temperature; and the leakage power is included in the total power. This is repeated until convergence.

To model timing errors we use the error model developed in [22]. The model takes into account process parameters such as  $V_{th}$  and  $L_{eff}$  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 is also validated with empirical data. With this, we estimate the timing error probability for each functional unit of each chip at a range of voltages.

### 5.3. Workloads

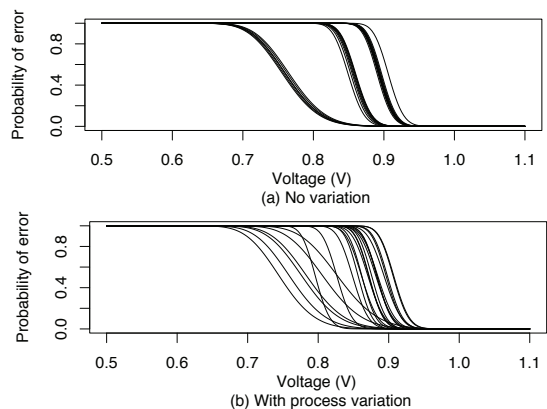
We use a collection of applications from SPECint (*bzip2*, *crafty*, *gap*, *gzip*, *mcf*, *parser*, *twolf*, *vortex*) and SPECfp (*applu*, *apsi*, *art*, *equake*, *mgrid*, *swim*). The simulation points present in SESC are used to run the most representative phases of each application. Each runs with the reference input set for about 2 billion instructions.

## 6. Evaluation

In this preliminary evaluation of our architecture we address several questions. First, we examine the impact of variation on the probability of timing errors at lower supply voltages. Next, we examine the potential for energy savings from aggressively lowering the voltage of select functional units while tolerating timing errors. Finally, we present results on fine-grain adaptation of voltage and replication to minimize energy. We also include an evaluation of the area and power overhead of the proposed architecture.

### 6.1. Effects of Variation on Timing Errors

We use the error and variation model described in Section 5 to examine the behavior of our chip as we lower supply voltages to save power. Figure 3 shows the probability of timing errors versus supply voltage for all the units in our floorplan, at the *same frequency*. Figure 3 (a) shows error rates for a chip with no variation. We distinguish three sets of error curves corresponding to three

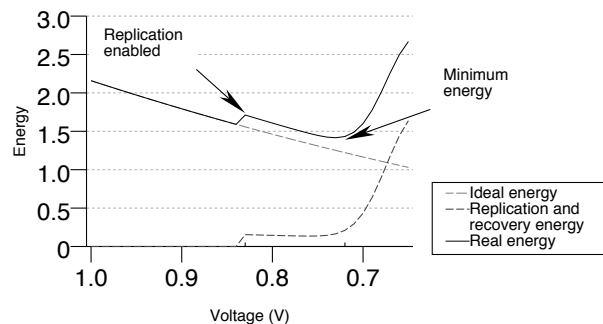


**Figure 3.** Probability of error vs. supply voltage for all functional units in a chip (a) without variation and (b) with process variation.

classes of functional units (memory-dominated, logic-dominated and mixed). The large number of critical paths in the memory units result in a steeper error curve for these units. The logic-dominated units have the slowest growth in error rate. Figure 3 (b) shows the same experiment for a chip with process variation. The error curves have a much wider distribution indicating significant variation in error rates behavior between units. Also, because of variation, error rates are higher at the same voltage than in the no variation case, as evidenced by the shift of most error curves to the right-hand side of the graph. These effects indicate an opportunity for functional unit level adaptation of both voltage and redundancy.

## 6.2. Fine-grain Voltage Reduction and Replication

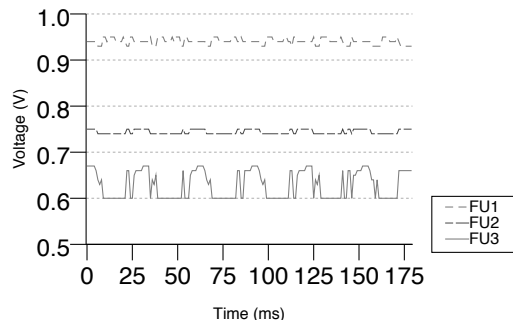
Figure 4 shows the energy consumption for a typical functional unit while running a small section of a benchmark at different supply voltages at the same clock rate. The ideal energy line shows how energy would scale if we could lower voltage without any impact on timing. As the voltage is lowered, total power consumption decreases resulting in lower energy. When the voltage drops below the safe margin, replication is enabled to detect and recover from timing errors. In this configuration, only the shadow pipeline registers need to be enabled, not the entire pipeline stage.



**Figure 4.** Energy as a function of voltage, for a functional unit.

The replication and recovery energy line accounts for the increase in power consumption due to replication (which for this unit is enabled around  $0.84V$ ) and for the time penalty incurred during error recovery. The penalty for error recovery increases sharply as the voltage is lowered beyond  $0.7V$ . The real energy includes both ideal and penalty energy. 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  $0.73V$ .

The voltage level that produces the ideal energy depends on several factors, including the variation profile of the unit or its sensitivity to timing errors. Ideal voltage is also dependent on the workload. Figure 5 shows the ideal voltage for a few units while running a section of the benchmark *bzip2*. We note that the ideal voltages differ significantly depending on the unit, and also over time within a benchmark. This is most likely linked to varying degrees of utilization that impact dynamic power consumption.



**Figure 5.** Changes in ideal supply voltage over time for three functional units of a chip running *bzip2*.

## 6.3. Energy Reduction for Fine-grain Adaptation

### 6.3.1. Timing Speculation

In this section we evaluate the potential energy reduction from fine-grain tuning of supply voltage and replication as detailed in Section 4. The objective is to find, for each benchmark, chip, and time interval, the set of voltages that minimizes the total energy. For all experiments, only voltage changes; frequency is always held constant. In this configuration, only the shadow pipeline registers will be selectively enabled. We consider four sets of restrictions on voltages: only one voltage rail is allowed (single), two rails (selectable per functional unit), three rails, and  $N$  rails (the best case where each unit gets its own selectable voltage).

Every functional unit has a voltage vs. energy curve qualitatively similar to Figure 4, where there are two local minima. To find the best voltage selections, we employ a hill-climbing algorithm that can cope with the discontinuity. For the  $N$ -rail case, every unit is given its own voltage rail. For the single-rail case, the algorithm iterates over a set of voltage levels between  $0.6V$  and  $1V$  and selects the one that requires the least energy. For dual rail, the voltage from the single-rail case is taken as constant, and all levels below are considered for the second rail. For the triple rail case, all levels below those two are considered. For dual and triple rails, the best voltage for each unit must be found, and the same hill-climbing algorithm is employed, albeit with fewer levels to consider.

Figure 6 shows the energy consumption for all benchmarks relative to energy consumption at  $1V$  with no replication. Each set of bars corresponds to an experiment with different numbers of voltage rails allowed. The numbers are averaged across the duration of the benchmark and across all chips. We find that single rail is only marginally effective at reducing energy (with about 5% reduction). The dual and triple rail case are significantly more effective with 11% and 12% savings respectively. With separate voltage for each unit, energy reduction is almost 18%. The error bars represent the distribution or energy consumption across all the chips we test. We note that there is variation in savings across benchmarks. This is due to variation in activity levels in different benchmarks that results in different error rates that in turn affect the optimal energy.

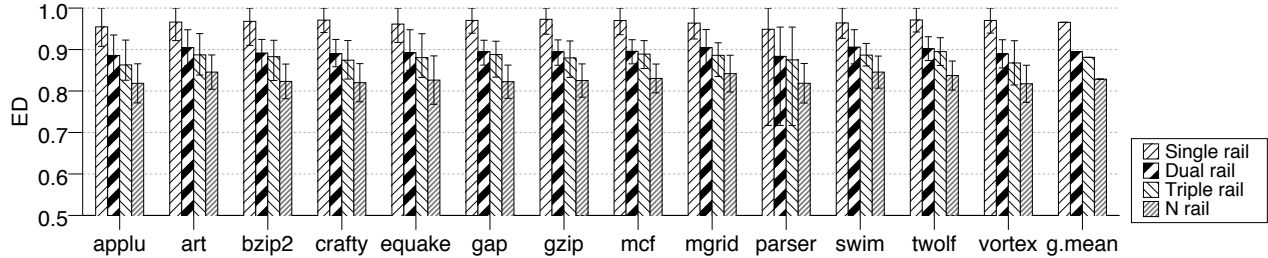


Figure 6. Reduction in ED for fine-grain adaptation with timing speculation for single, dual, triple, and  $N$  rails for all benchmarks.

Applications with lower activity factors such as *mcf* are likely to see somewhat greater savings because the voltage can be lowered further without incurring a dramatic rise in error rate.

The penalty for recovering from timing speculation increases the execution time only slightly, by about 0.25% on average when the optimization targets a reduction in ED. When the optimization targets a metric such as Energy, which puts less emphasis on delay, the execution time increases by about 1%.

### 6.3.2. Full Replication

We perform a similar study to evaluate the potential energy reduction for full replication. In this configuration, replication of both logic and latches of the shadow pipeline is always enabled. Reduction in ED comes from aggressive scaling of supply voltage. For full replication, the savings are nearly the same, relative to a baseline with full replication at 1V. The dual and triple rail case achieve 11% and 13% savings respectively. For the  $N$ -rail case, with a separate voltage for each unit, ED reduction is about 19%.

### 6.4. Overhead

The proposed architecture introduces power and area overhead. To estimate it, we synthesized the OpenRISC 1200 [12] and the Open Graphics Project “HQ” microcontroller [16] for Xilinx Spartan 3 FPGA. For HQ, the ALU was extracted and duplicated; synthesis was performed with and without routing and checker logic to determine the additional area consumed. We also considered overhead reported in related work. Core Cannibalization [21] requires two  $2 \times 2$  crossbars to route signals between primary and spare pipelines, and this extra hardware requires no more than 3.5% area overhead for the three pipelines. According to Gupta et al. [7], adding  $5 \times 5$  crossbars to five processors has a 12% area overhead.

Based on synthesis results and data from related work, we have computed area estimates 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 logic area powered on for timing speculation is up to 6%. In our evaluation, we assume a power overhead proportional to area.

## 7. Conclusions

This paper proposes a new reliable processor architecture that dynamically adapts the amount of protection to the characteristics of each individual chip and its runtime behavior. This architecture provides selective redundancy at the pipeline stage level to allow targeted protection against soft and hard errors. It employs timing speculation for mitigation of variation-induced timing errors and allows fine-grain voltage scaling to reduce the power overhead of the error protection and even allow for energy savings.

We have shown how our techniques are effective at mitigating the effects of variation and timing errors, reducing energy consumption by up to 19%. Initial results also indicate sufficient variation in behavior across different chips and different workloads to warrant an on-line adaptation mechanism, which we leave for future work.

## References

- [1] S. Borkar et al. Parameter variations and impact on circuits and microarchitecture. In *Design Automation Conference*, June 2003.
- [2] D. Brooks et al. Watch: A framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture*, June 2000.
- [3] K. Constantinides et al. Online design bug detection: RTL analysis, flexible mechanisms, and evaluation. In *International Symposium on Microarchitecture*, pages 282–293, Nov. 2008.
- [4] J. Dorsey et al. An integrated quad-core Opteron processor. In *International Solid State Circuits Conference*, February 2007.
- [5] D. Ernst et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *International Symposium on Microarchitecture*, December 2003.
- [6] B. Greskamp and J. Torrellas. Paceline: Improving single-thread performance in nanoscale CMPs through core overclocking. In *Parallel Architecture and Compilation Techniques*, Sept. 2007.
- [7] S. Gupta et al. StageNetSlice: a reconfigurable microarchitecture building block for resilient CMP systems. In *CASES*, 2008.
- [8] S. Gupta et al. The StageNet fabric for constructing resilient multicore systems. In *International Symposium on Microarchitecture*. IEEE Computer Society, 2008.
- [9] International Technology Roadmap for Semiconductors (2006 Update).
- [10] H. Jiang and M. Marek-Sadowska. Power gating scheduling for power/ground noise reduction. In *Design Automation Conference*, 2008.
- [11] W. Kim et al. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *High Performance Computer Architecture*, pages 123–134, Feb. 2008.
- [12] D. Lampret. OpenRISC 1200 IP Core Specification. <http://opencores.org>.
- [13] X. Liang and David Brooks. Mitigating the impact of process variations on processor register files and execution units. In *International Symposium on Microarchitecture*, December 2006.
- [14] X. Liang et al. Revival: A variation-tolerant architecture using voltage interpolation and variable latency. *IEEE Micro*, 29(1):127–138, 2009.
- [15] D. Marculescu and E. Talpes. Variability and energy awareness: A microarchitecture-level perspective. In *Design Automation Conference*, June 2005.
- [16] T. Miller and P. Urkedal. <http://opengraphics.org>.
- [17] S. Mitra et al. Combinational logic soft error correction. In *International Test Conference*, November 2006.
- [18] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2006. <http://www.R-project.org>.
- [19] J. Renau et al. SESC Simulator, January 2005. <http://sesc.sourceforge.net>.
- [20] P.J. Ribeiro Jr. and P.J. Diggle. geoR: A package for geostatistical analysis. *R-NEWS*, 1(2), 2001.
- [21] B. F. Romanescu and D. J. Sorin. Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults. In *Parallel Architectures and Compilation Techniques*, 2008.
- [22] S. R. Sarangi et al. VARIUS: A model of parameter variation and resulting timing errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, 2008.
- [23] K. Skadron et al. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, June 2003.
- [24] T.J. Slegel et al. IBM’s S/390 G5 microprocessor design. *Micro, IEEE*, 19(2):12–23, Mar/Apr 1999.
- [25] H. Su et al. Full chip leakage estimation considering power supply and temperature variations. In *International Symposium on Low Power Electronics and Design*, August 2003.
- [26] R. Teodorescu et al. Mitigating parameter variation with dynamic fine-grain body biasing. In *International Symposium on Microarchitecture*, December 2007.
- [27] C. Weaver and T. Austin. A fault tolerant approach to microprocessor design. In *DSN*, July 2001.