# DARP: Dynamically Adaptable Resilient Pipeline Design in Microprocessors

Hu Chen, Sanghamitra Roy, Koushik Chakraborty
BRIDGE Lab, Electrical and Computer Engineering, Utah State University
hu.chen@aggiemail.usu.edu, {sanghamitra.roy, koushik.chakraborty}@usu.edu

## ABSTRACT

In this paper, we demonstrate that the sensitized path delays in various microprocessor pipe stages exhibit intriguing temporal and spatial variations during the execution of real world applications. To effectively exploit these delay variations, we propose Dynamically Adaptable Resilient Pipeline (DARP)–a series of runtime techniques to boost power performance efficiency and fault tolerance in a pipelined microprocessor. DARP employs early error prediction to avoid a major portion of the timing errors. Using a rigorous circuit-architectural infrastructure, we demonstrate substantial improvements in the performance (9.4–20%) and energy efficiency (6.4–27.9%), compared to state-of-the-art techniques.

## 1. INTRODUCTION

Rapid miniaturization of transistor devices has introduced several uncertainties in their operation. Modern microprocessor pipelines experience multiple sources of delay variations, sometimes manifesting as a timing error [1, 17, 15]. Some of the well studied sources of delay variations include process variation and aging [17, 15]. To ensure reliable operation while preserving energy efficiency under delay variation, two of the most popular techniques applied on microprocessor pipelines are timing speculation and clock skew tuning. Timing speculation reduces the guardbands to a point where errors occur and are recovered using error detection and recovery techniques like Razor [7]. Post silicon clock skew tuning is used to adjust the clock skews of the pipeline registers to allow some stages to borrow time from others [18]. Combined with timing speculation, clock skew tuning can be even more effective, as it becomes possible to configure the clock skews more aggressively to allow occasional errors, thereby improving both performance and energy efficiency [20].

Variation in delays of pipeline stages also depends on specific applications running on the microprocessor, as well as dynamic fluctuations in voltage and temperature. For instance, sensitized paths during the execution of real world applications may exhibit strikingly distinct characteristics than expected from a purely static timing analysis. However, to the best of our knowledge, there exist very limited works that exploit delay variance from real world applications. For example, recent works on adaptive clock skew tuning by Ye et al. and Lak et al. tackle process variation and aging based delay variation, *but do not address the delay variation from real world applications* [20, 11].

In this paper, we employ a circuit-architectural analysis to investigate and exploit the delay variation seen in sensitized circuit paths during real world application execution. We identify two distinct classes of these variations, driven by workloads: (a) *temporal*–delay variation within a given pipe stage during different phases of a program; and (b) *spatial*–distinct delay distributions among different pipe stages of a microprocessor. We also exploit early error prediction, a recently proposed technique that allows us to predict timing errors many cycles in advance using the instruction program counter (PC) [16, 19, 4]. We combine early error prediction with clock skew tuning to propose *Dynamically Adaptable Resilient Pipeline* (DARP), which outlines a next wave of innovation in pushing the energy efficient frontier of pipelined microprocessor design.

We make the following contributions in this paper:

- We show a striking temporal and spatial variance in the sensitization of critical paths in a microprocessor component, during the execution of real programs (Section 3). Our rigorous analysis integrates architectural simulation data with a gate level logic analyzer to determine critical paths and critical delays sensitized during program execution.
- We propose DARP, a dynamically adaptable resilient pipeline. In addition to handling the well studied delay variations from process variation and aging, DARP can adapt a pipeline to the spatial and temporal delay variations from real workloads. DARP employs program phase driven early timing error prediction and dynamic frequency and clock skew adjustment to exploit workload specific delay characteristics in pipelined systems (Section 4).
- Using a rigorous circuit-architectural simulation (Section 5), combining synthesized hardware with real world application execution through architectural simulation, we demonstrate dramatic improvement in the performance (9.4–20%) and energy efficiency (6.4–27.9%), compared to state-of-the-art timing speculation techniques (Section 6). DARP schemes have negligible core level power overheads of 0.84% and 3.35%, giving an energy-efficient alternative for robust pipelines.

## 2. RELATED WORK

Previous works most relevant to DARP fall in the broad categories of timing speculation and clock skew scheduling. One approach to boost energy efficiency is by operating at a tighter frequency and detecting and correcting occasional timing errors after occurrence (e.g., Razor [7] and others [2]). Another approach aims to employ sensors for *just in-time* prediction of timing errors, and advocates time borrowing to avoid timing errors [6, 8, 3]. More recent works demonstrate *early prediction* of timing errors, several cycles in advance, by

978-3-9815370-2-4/DATE14/©2014 EDAA

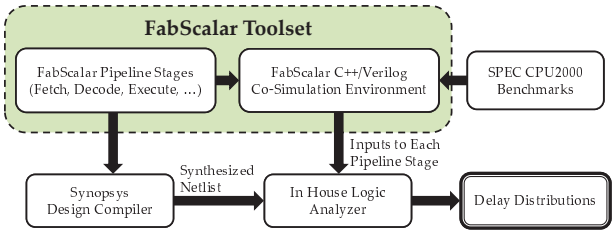


Figure 1: Cross-Layer Methodology.

observing the correlation between instructions and their sensitized path delays [16, 19, 4]. In the domain of clock skew scheduling, recent works by Ye et al. and Lak et al. propose dynamic techniques to combine with timing speculation through error detection [20, 11]. However, their works do not consider real workload execution on a pipelined microprocessor, and are based on random inputs. Our proposed pipeline DARP overcomes this limitation through a cross-layer theme that considers real program driven circuit path sensitization. Moreover, we combine early prediction of timing errors with dynamic clock skew tuning to efficiently exploit the workload driven circuit delay variances.

# 3. MOTIVATION

In this section, we demonstrate the wide variance in sensitized path delays in microprocessor pipe stages during the execution of real applications. Our cross-layer analysis, identifies two distinct sources of this variance: *temporal* and *spatial*. Collectively, these delay variances uncover intriguing possibilities in the runtime adaptation of various pipe stages in a microprocessor, boosting performance and energy efficiency of the entire system.

## 3.1 Sensitized Pipe Stage Delay

There is a strong correlation between a static instruction and the paths sensitized by each of its dynamic instances [16]. However, when we analyze different instructions sensitizing specific paths in a microprocessor pipe stage, we observe a wide disparity in their sensitized delays. For example, during a particular program execution, certain instructions may experience a mere fraction of the clock cycle delay, meeting the timing with a large margin to spare. On the other hand, other instructions may experience a much larger delay, with limited slack. To analyze this intriguing property, we employ a rigorous cross-layer methodology, outlined next.

### *3.1.1 Methodology*

As instructions from various workloads pass through the pipe stages, they sensitize different paths, and therefore observe different logic computation delays. To capture these characteristics, Figure 1 shows our rigorous cross-layer methodology combining architecture level workload simulation with circuit level timing analysis. We use the RTL modules, configured for the out-of-order *Core-1* configuration, from the FabScalar infrastructure [5]. The modules are synthesized using Synopsys Design Compiler to obtain a netlist of gates, Subsequently, we perform architectural simulation of several real world applications using the FabScalar Co-Simulation environment, and extract cycle-by-cycle input vectors for various pipe stage RTL modules. We use these input vectors with our in-house logic analyzer to obtain the delay characteristics of various pipe stages as instructions flow through the pipeline.

### *3.1.2 Results*

Figure 2 shows the delay variance on three pipe stages. The *Retire* stage graduates instructions from the pipeline, making their changes visible outside (Figure 2(a)). For the *Execute* stage, we use the delay variance seen in the *Simple ALU* (Figure 2(b)). The *Issue* stage is responsible for scheduling instructions on various functional units (Figure 2(c)). All the three figures show the cumulative distribution function (CDF) plot for the sensitized delay from workload instructions.

From these figures, we observe a wide range of sensitized path delays in these pipe stage modules. Furthermore, various workloads also show substantially different delay profiles. On a closer inspection, we can broadly characterize these variations into two major classes:

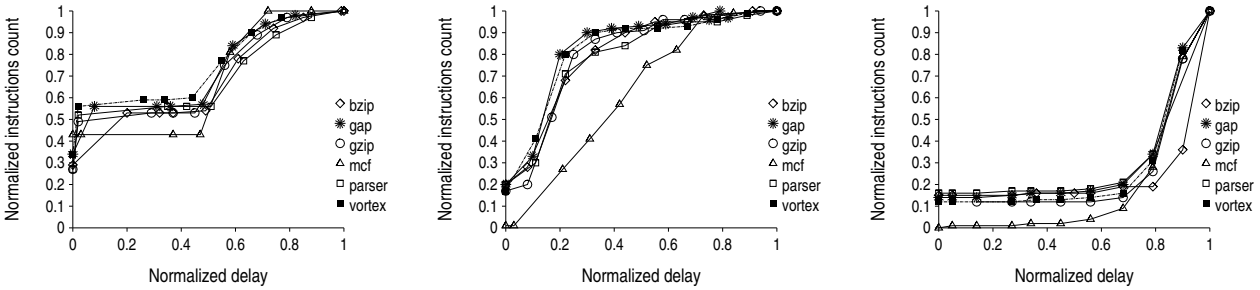
- **Temporal:** Within a given pipe stage, different instructions in a given workload often exhibit substantial variance. For example in Figure 2(a), we observe that 29% instructions in *bzip* exhibit negligible delay in the *Retire* pipe stage, whereas 57% instructions in *vortex* exhibit similar characteristics.
- **Spatial:** Different pipe stages exhibit a range of delay profiles. For example, comparing *Execute* and *Issue* (Figures 2(b) and 2(c), respectively), we can see intriguing distinctions. Across different benchmarks, a vast majority of instructions exhibit 80% or higher percentage of the maximum delay in Issue, whereas only a limited fraction of instructions experience such high delay in the Execute stage. Comparing across different workloads, we also notice that this spatial delay imbalance between the *Execute* and *Issue* stage is substantially more pronounced in *vortex* than in *mcf*, primarily due to more uniform delay characteristics in the *Execute* stage for *mcf*.

## 3.2 Significance

Our results above indicate a wide disparity in delay sensitized from different instructions. On the other hand, previous works have demonstrated *striking similarity* in delay characteristics from recurring instances of the same instruction [16, 19]. Then, a key question is can we simultaneously exploit both these workload driven behaviors in an unified manner in a microprocessor pipeline? We strongly believe this is a genuine possibility and propose our scheme *Dynamically Adaptable Resilient Pipeline (DARP)*. DARP exploits repeatability of delay characteristics of a program by substantially tightening the operating frequency and dynamically predicting upcoming timing errors several cycles in advance. These timing errors are subsequently avoided through a stall insertion in the pipeline, radically diminishing the penalty from error detection and correction with replay. On the other hand, spatial variances in sensitized delay in the pipe stages are seamlessly integrated by our low-overhead controller that dynamically adjusts processor frequency and clock skews in every epoch. In the next section, we describe our proposed schemes in details.

# 4. DYNAMICALLY ADAPTABLE RESILIENT PIPELINE

In this section, we present an overview of designing a DARP system and the associated design challenges.




(a) Retire (b) Execute (c) Issue

Figure 2: CDF of sensitized path delay variance in several microprocessor pipe stages.

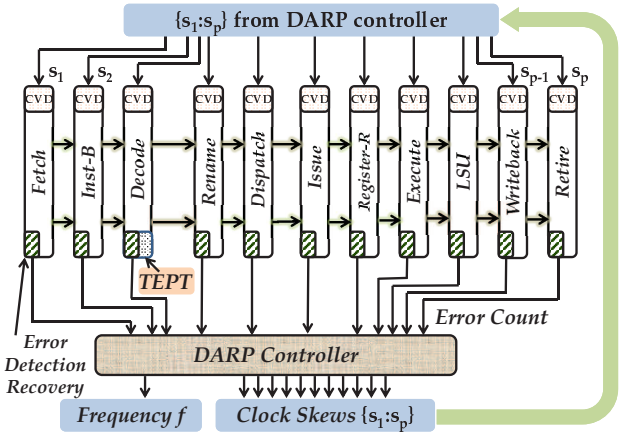



Figure 3: DARP Overview.

## 4.1 DARP Overview

Figure 3 shows an overview of our proposed DARP pipeline design. The figure shows two major augmentations of a microprocessor pipeline: (a) Timing Error Prediction Table (TEPT) in the decode stage coupled with error detection and recovery in every pipeline stage (Section 4.2); and (b) DARP controller design and its auxiliary components (CVD) in each pipeline stage to adjust the clock skew (Section 4.3). Collectively, our techniques work in tandem to substantially improve the power-performance characteristics of a microprocessor pipeline.

## 4.2 Exploiting Early Error Prediction

To exploit temporal variance, we maintain a small, dynamically managed table of instructions prone to cause repeated timing errors in various pipeline stages. We refer this table as the *Timing Error Prediction Table (TEPT)*. Three central aspects of early error prediction are described below:

**Error Detection:** To detect runtime timing errors in various pipe stages, we protect all the potentially critical paths by a double sampling flip-flop in their output pins, similar to Razor [7]. These critical paths are identified by a technique proposed by Lak et al. [11]. When an error is detected, we initiate two actions: (a) insert the error causing instruction PC along with the relevant pipe stage information in the TEPT to avoid timing errors from the same instruction in the near future; and (b) initiate an instruction replay to recover from the fault [7]. During the repeated execution of the same instruction, timing errors are avoided as described next.

**Error Avoidance:** During the decode of an instruction, we access the TEPT to see if that instruction is likely to cause a pipeline error. If a matching entry is found, we update the instruction meta-data to propagate a stall signal as the instruction proceeds in the pipeline. When the instruction enters the pipe stage where it previously caused a timing error, the stall signal is triggered, thereby allowing the instruction to occupy that pipe stage for two consecutive cycles. Forward flow of instructions is avoided for that cycle by recirculating the inputs to all other pipe stages. Consequently, timing error from that instruction is avoided, recovering a bulk of the performance loss. This is possible as pipeline stalls incur at least 10X lower penalty than an instruction replay in modern processors. During error avoidance by stalling, a Razor flip-flop can incorrectly flag an error. However, such a flag is invalidated by combining with the stall signal.

**Dynamic TEPT Management:** Entries in the TEPT are managed at runtime to exploit the workload phase behaviors. Whenever a timing error is detected, we insert the instruction PC into the table. Recall that the detection of a timing error implies that no error was predicted from that PC before. If the table is full, as is expected after a brief table warm-up period, we search for an eviction entry using a pseudo-LRU (least recently used) policy. Pseudo-LRU policy is widely used in modern caches to avoid the complexity of implementing a full LRU policy. Table entries are managed as a CAM (Content addressable memory), which allows the best use of the small space available.

## 4.3 DARP Controller Design

We dynamically exploit the spatial imbalance among pipe stages using: (a) a system level DARP controller that dynamically determines the clock skew configurations of each pipe stage; and (b) lower level clock vernier devices (CVD) that control the clock skews of individual pipe stages. These are detailed next.

### 4.3.1 DARP Controller

The role of the DARP controller is to dynamically configure the frequency of the pipeline and the clock skews in every pipe stage. This is done in hardware using the information about the timing errors in each pipe stage that are not covered by early prediction.

Algorithm 1 outlines the operation of the DARP controller. The DARP controller repeats this algorithm once in each epoch (every $N$ instructions) to dynamically adapt the pipeline frequency as well as clock skews to resolve the spatial imbal-

**Algorithm 1** DARP

**Input:** $\{s_1 : s_p\}$, $f$, $\{n_1 : n_p\}$
**Output:** $\{s_1 : s_p\}$, $f$

1: $k \leftarrow$ *Pipe stage with max errors*; $n_{max} \leftarrow n_k$
2: $l \leftarrow$ *Pipe stage with min errors*; $n_{min} \leftarrow n_l$
3: **if** $n_{min} == 0$ && $n_{max} <= \rho$ **then**
4: $\quad f \leftarrow f + f_{step}$
5: **else if** $n_{min} >= \eta$ **then**
6: $\quad f \leftarrow f - f_{step}$
7: **end if**
8: $T \leftarrow \frac{1}{f}$
9: $n_{avg} \leftarrow avg(n_1 : n_p)$
10: $\{\triangle_1 : \triangle_p\} \leftarrow \{n_1 : n_p\} - n_{avg}$
11: **for** $i \leftarrow 1 : p$ **do**
12: $\quad$ **if** $\triangle_i < 0$ **then**
13: $\quad\quad s_i \leftarrow s_i - 001$
14: $\quad$ **else if** $\triangle_i > 0$ **then**
15: $\quad\quad s_i \leftarrow s_i + 001$
16: $\quad$ **end if**
17: **end for**
18: Adjust $\{s_1 : s_p\}$ to maintain T*p cycles

ance in pipe stage delays. Each pipe stage clock skew is configured using a 3-bit skew configuration of its CVD. The input to the DARP controller is the skew configurations $\{s_1 : s_p\}$ for $p$ pipe stages, the frequency $f$, and the timing error counts $\{n_1 : n_p\}$ from the previous epoch.

**Tuning Operating Frequency**: Steps 1-7 outline our frequency tuning at each epoch. We first estimate the pipe stages with the maximum and minimum timing errors (stages $k$ and $l$). We tighten the frequency by a step interval ($f_{step}$) when $n_{min}$ is zero and $n_{max}$ is lower than a constant threshold $\rho$. Likewise, we relax the frequency when $n_{min}$ exceeds a lower bound given by constant $\eta$. These steps help us to tighten or relax the operating frequency once in each epoch, based on the timing error profile of the pipeline in the previous epoch.

**Configuring Clock Skews**: Steps 9-18 illustrate the dynamic clock skew tuning. We first calculate the set $\{\triangle_1 : \triangle_p\}$. The $i^{th}$ element in this set contains the difference between the timing errors in stage $i$ ($n_i$) and the average timing error $n_{avg}$. The pipe stages that have timing errors below average get a reduced clock skew, while the pipe stages with timing errors above average see an increase in the positive clock skew. The clock skew tuning is done using a 3-bit programmable CVD, detailed in Section 4.3.2. This dynamic reconfiguration of clock skews ensures that the pipe stages sensitizing higher delay paths can borrow time from those sensitizing lower delay paths. This process in turn helps us to further tighten the frequency in the next epoch. Step 18 readjusts the clock skews to ensure the total time for a $p$ stage pipeline is maintained at $T * p$ cycles, $T$ being the time period (step 8).

### 4.3.2 Clock Vernier Device (CVD)

We use clock vernier devices (CVD) as our clock tuning elements (Figure 4 ) [11]. CVDs can generate several skew configurations based on their input bits. The latches $a$, $b$ and $c$ also store the skew configurations to be used in the next epoch. For example, for a 3-bit input, we can have 8 skew configurations. If we set bit 011 to represent zero skew, and perform skew increments/decrements in steps of $\delta$, then we can get the positive and negative skew configurations of $\{-3\delta, -2\delta, -\delta, 0, \delta, 2\delta, 3\delta, 4\delta\}$. These skews help us to dynamically lend time to pipe stages sensitizing higher delay critical paths, and borrow time from pipe stages sensitizing low delay critical paths. To manage the overhead of adding CVDs, we group multiple flip-flops in a pipe stage to share a single CVD, using the clustering algorithm proposed by Lak et al. [11].

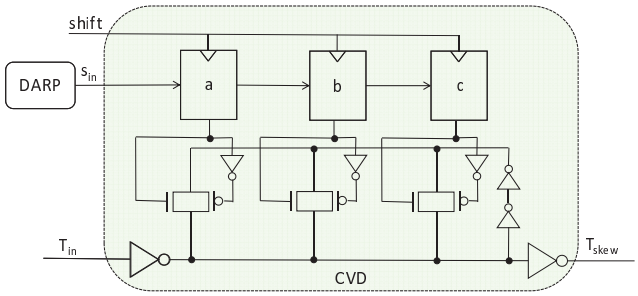


Figure 4: Configuring Clock Vernier Devices using DARP.

### 4.3.3 Implementation

The clock skew adjustment must be made when the pipeline is empty to avoid metastability issues [2]. Thus, at the end of every epoch, we flush the entire pipeline, and run Algorithm 1 to obtain the new frequency and skew adjustments for the new epoch. The pipeline operation is resumed only after skew adjustments are applied and the new frequency is stabilized. To limit the overhead of this operation, we adopt several strategies. First, we choose a large enough epoch of 100K cycles that can amortize the dynamic adjustment cost while offering enough opportunities to adapt to the workload characteristics[1]. Second, we use a fairly simple algorithm in hardware to incrementally adjust clock skews and frequency, avoiding a full blown search. Third, we use a widely popular dynamic clocking technique that allows a rapid adjustment of the clock frequency [12]. Combining these strategies, we limit the overhead of a single dynamic reconfiguration to within 100 cycles per epoch (overhead of 0.1%).

### 4.3.4 DARP for Aged Pipelines

Different stages of a microprocessor pipeline may age asymmetrically over time, as thermal fluctuations are predominant in the back end of the pipeline [13]. Asymmetric aging can substantially change the critical delays of certain pipe stages leading to increased timing errors. The DARP algorithm can automatically adjust the pipeline frequency and clock skews as the processor ages, as it uses the current timing error information from each pipe stage.

# 5. METHODOLOGY

In this section, we describe our cross-layer methodology for power-performance trade-off analysis of DARP.

## 5.1 Architecture Layer

Our architectural simulation is based on the FabScalar infrastructure [5]. We next provide details of our core microarchitecture and specific workloads used in this work.

[1]A thread is typically scheduled for a 10ms time quantum, within which we can run our configuration adjustments 300 times assuming a 3GHz clock.

#### 5.1.1 Core Microarchitecture

We choose FabScalar's *Core-1* configuration, with an eleven stage $p = 11$ out-of-order superscalar pipeline capable of fetching, issuing and committing 4 instructions each cycle. The core uses a two-level cache hierarchy: a 32 KB 4-way split Instruction/Data L1 cache with a latency of 1 cycle and an 8MB 16-way L2 cache with a latency of 25 cycles. The main memory access time is 240 cycles. In each pipeline stage, we employ a timing-error detection and *instruction replay* mechanism similar to [7] and our proposed DARP enhancements (Figure 3). We model cycle accurate timing behavior of our proposed architecture under various schemes.

#### 5.1.2 Workloads

We use several SPEC CPU2000 integer benchmarks representing a range of real world applications, in our analysis. We simulate these benchmarks for 10ms, which is a typical time quanta allotted to a thread from the operating system.

### 5.2 Circuit Layer

In order to evaluate our approach, we use our in-house statistical timing analysis tool to calculate the propagation delay of each pipeline stage on every cycle (Figure 1). To perform gate level timing analysis, we follow several important steps. First, we synthesize our implementation with the Synopsys Design Compiler. Second, to obtain propagation delay of the sensitized circuit paths, we integrate the delay characteristics from HSPICE simulation based on the Predictive Technology Model (PTM) for the 22nm node [21], with the synthesized netlist gate delays. The delay models also incorporate the effects of process variation [10]. This methodology allows us to obtain the circuit delay characteristics of a lower technology node than using a standard cell library. Third, for SRAM based memory modules such as the L1 instruction/data cache, the branch target buffer (BTB) and the conditional branch predictor, we use CACTI 6.0 to get the timing information [14], and subsequently integrate it to the circuit delay.

### 5.3 Timing Error Simulation Methodology

We use a combination of two distinct strategies to simulate timing errors in the microprocessor pipeline: (1) frequency over-scaling; and (2) voltage scaling. We increase the frequency and/or scale down to operating voltage to a point, where timing errors begin to occur in several pipe stages. Combined together, these two strategies help us to evaluate the power-performance characteristics of our proposed schemes as well as previous works in the presence of timing errors.

## 6. EXPERIMENTAL RESULTS

In this section, we present experimental results of comparing our DARP system with contemporary schemes based on online clock tuning and timing speculation.

### 6.1 Comparative Schemes

- **Razor:** This scheme models one of the most popular techniques based on timing speculation [7, 9]. Using this scheme, the operating frequency of a processor can be raised, by allowing occasional timing errors in various stages of the pipeline. These timing errors are detected using double-sampling rear end flip-flops. After error detection, an instruction replay is triggered to correct the error.
- **Online Clock Tuning with Timing Speculation (OCTTS):** This scheme models recent work to combine clock skew tuning with timing speculation [20, 11]. Instead of application driven sensitized path delays, both these schemes use random inputs to drive their clock tuning mechanisms.
- **DARP:** In this scheme, we do dynamic frequency tuning for each benchmark. At each operating frequency, we readjust the clock skews of the pipe stages to best exploit the spatial imbalance (Algorithm 1).
- **DARP-Pred:** In addition to DARP, this scheme employs a 4K sized timing error prediction table (TEPT) to do early timing error prediction along the entire pipeline (Section 4.2). DARP-Pred can avoid most timing-errors than DARP through early prediction, thereby saving several costly instruction replays.

### 6.2 Performance Comparison

Figure 5 shows the performance of our proposed DARP schemes, compared to Razor and OCTTS. The results are normalized to the Razor scheme. By exploiting the delay variation due to topology and process variation in a general purpose out-of-order microprocessor, we observe that OCTTS can boost the application performance over Razor (19% on an average). Our proposed DARP scheme further improves the performance of the system by dynamically exploiting the workload driven spatial imbalance in addition to the previous factors. DARP-Pred additionally exploits the temporal delay variance in pipe stages with early error prediction and avoidance. For example, we notice that DARP-Pred can deliver 17.9% and 20% performance improvements over OCTTS in *bzip* and *gap* benchmarks, respectively. On an average, DARP and DARP-Pred show 6.7% and 13.4% speedups over OCTTS across these benchmarks. The improvements over Razor are substantially larger.

Figure 6 gives further insight on the performance improvement of the DARP schemes. The first and second bar for each benchmark represents the replay cycles in the OCTTS and DARP schemes, respectively, while the third bar is for the replay and stall cycles in the DARP-Pred scheme. These schemes work at the same frequency and all these cycle numbers are normalized to the OCTTS scheme. Since DARP encounters much less timing errors, it has a much lower replay penalty than the OCTTS scheme. On the other hand, DARP-Pred converts a significant part of the replay penalty (all of it in some benchmarks, e.g. bzip) into the stall penalty, further reducing the penalty cycles from the DARP scheme. This is possible as a majority of the timing errors can be avoided using early prediction. Using only Razor at the same frequency incurs substantially higher timing errors and replay penalties, and thus is omitted from the figure.

### 6.3 Energy-Efficiency Comparison

Figure 7 shows the energy-efficiency of both DARP and the comparative schemes, in terms of the Energy Delay Product (EDP). To search the best EDP for each scheme, we explore discrete supply voltages in steps of 5%, down to 70% of the nominal voltage. We see that OCTTS has a notable EDP reduction, with the average value of 5.8%; while the DARP schemes show a further energy-efficiency improvement, with the average EDP reduction of 12.7% and 19.8%,

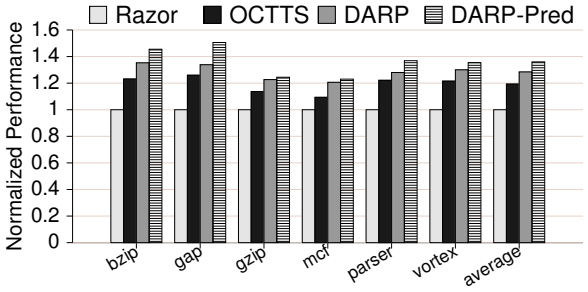



Figure 5: Performance comparison normalized to Razor.

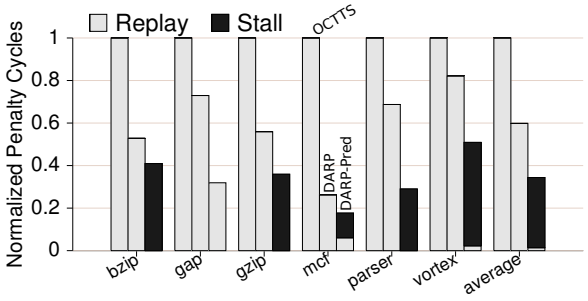



Figure 6: Penalty Cycles from Timing Errors.

respectively, over Razor. This energy-efficiency improvement stems from three factors: (a) workload specific skew adjustments; (b) early error prediction to avoid timing errors; and (c) delay reduction discussed in Figure 5. When compared against OCTTS, the DARP and DARP-Pred schemes show an average EDP reduction of 7% and 15%, respectively.

## 6.4 Power Overhead of DARP

The primary power overhead in DARP comes from the counters in each stage to record timing errors, the DARP controller (Figure 3) and the 4K sized TEPT for DARP-Pred. These overheads are estimated relative to the OCTTS scheme. Hence we exclude the CVDs and buffers for minimum delay padding that are already present in many modern microprocessors. The power overhead of our schemes, are calculated by synthesizing the *Core-1* from the FabScalar infrastructure using the Synopsys Design Compiler with a 45nm FreePDK library. We add all the hardware enhancements of the DARP pipeline and estimate the overhead relative to the core power. Overall, we find a power overhead of 0.84% and 3.35% relative to the core power, for the DARP and DARP-Pred schemes, respectively. Energy efficiency results presented in Figure 7 include these overheads.

# 7. CONCLUSION

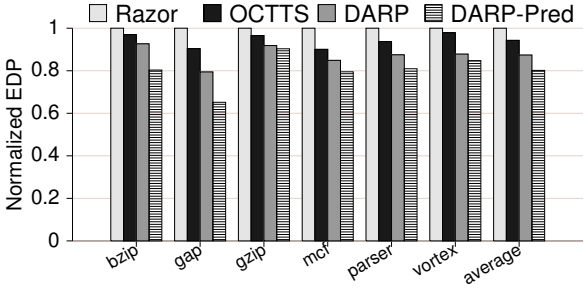



Figure 7: EDP Comparison normalized to Razor.

We present a novel runtime approach to exploit the variations in sensitized path delays among various pipe stages in modern microprocessor designs. Two pillars of our proposed system are early prediction of timing errors from program phases, and exploiting a low-overhead controller for frequency and clock skew tuning. Through a rigorous circuit-architectural infrastructure, we demonstrate significant improvements in the performance (9.4–20%) and energy efficiency (6.4–27.9%), compared to state-of-the-art techniques.

## Acknowledgment

This work was supported in part by National Science Foundation grants CNS-1117425, CAREER-1253024, CCF-1318826 and donation from the Micron Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# 8. REFERENCES

[1] BORKAR, S. Design Perspectives on 22nm CMOS and Beyond. In *Proc. of 46th Proc. of DAC* (2009), pp. 93–94.

[2] BOWMAN, K. AND OTHERS Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance. *J. of Solid-State Circ. 44*, 1 (2009), 49–63.

[3] BOWMAN, K. AND OTHERS Circuit techniques for dynamic variation tolerance. In *Proc. of DAC* (2009), pp. 4–7.

[4] CHAKRABORTY, K. AND OTHERS Efficiently Tolerating Timing Violations in Pipelined Microprocessors. In *Proc. of DAC* (2013), no. 102.

[5] CHOUDHARY, N. K. AND OTHERS FabScalar: composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template. In *Proc. of ISCA* (2011), pp. 11–22.

[6] CHOUDHURY, M. R. AND OTHERS TIMBER: Time borrowing and error relaying for online timing error resilience. In *Proc. of DATE* (2010), pp. 1554–1559.

[7] DAS, S. AND OTHERS RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance. *JSSC 44*, 1 (Jan. 2009), 32–48.

[8] GHASEMAZAR, M., AND PEDRAM, M. Minimizing the Energy Cost of Throughput in a Linear Pipeline by Opportunistic Time Borrowing. In *Proc. of ICCAD* (2008).

[9] KEITH BOWMAN, E. Circuit Techniques for Dynamic Variation Tolerance. In *Proc. of DAC* (2009).

[10] KOTHAWADE, S. AND OTHERS Analysis of Intermittent Timing Fault Vulnerability. *Microelectronics Reliability 52*, 7 (July 2012), 1515–1522.

[11] LAK, Z., AND NICOLICI, N. In-system and on-the-fly clock tuning mechanism to combat lifetime performance degradation. In *Proc. of ICCAD* (2011), pp. 434–441.

[12] MCNAIRY, C., AND BHATIA, R. Montecito: A Dual-Core, Dual-Thread Itanium Processor. *IEEE Micro 25*, 2 (2005), 10–20.

[13] MESA-MARTINEZ, F. J. AND OTHERS Power model validation through thermal measurements. In *Proc. of ISCA* (2007), pp. 302–311.

[14] MURALIMANOHAR, N. AND OTHERS Architecting Efficient Interconnects for Large Caches with CACTI 6.0. *IEEE Micro 28*, 1 (2008), 69–79.

[15] PAN, S. AND OTHERS IVF: Characterizing the vulnerability of microprocessor structures to intermittent faults. In *Proc. of DATE* (2010), pp. 238–243.

[16] ROY, S., AND CHAKRABORTY, K. Predicting Timing Violations Through Instruction Level Path Sensitization Analysis. In *Proc. of DAC* (2012), pp. 1074–1081.

[17] SARANGI, S. AND OTHERS VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects. *IEEE Transactions on Semiconductor Manufacturing 21*, 1 (2008), 3 –13.

[18] TADESSE, D. AND OTHERS AutoRex: An automated post-silicon clock tuning tool. In *Proc. of ITC* (2009), pp. 1–10.

[19] XIN, J., AND JOSEPH, R. Identifying and predicting timing-critical instructions to boost timing speculation. In *Proc. of MICRO* (2011), pp. 128–139.

[20] YE, R. AND OTHERS Online clock skew tuning for timing speculation. In *Proc. of ICCAD* (2011), pp. 442–447.

[21] ZHAO, W., AND CAO, Y. New Generation of Predictive Technology Model for sub-45nm Early Design Exploration. *IEEE Transactions on Electron Devices 53*, 11 (2006), 2816 –2823.