2017-10-01

Edge-TM: Exploiting transactional memory for error tolerance and energy efficiency

Edge-TM: Exploiting Transactional Memory for Error Tolerance and Energy Efficiency

ABSTRACT

Scaling of semiconductor devices has enabled higher levels of integration and performance improvements at the price of making devices more susceptible to the effects of static and dynamic variability. Adding safety margins (guardbands) on the operating frequency or supply voltage prevents timing errors, but has a negative impact on performance and energy consumption. We propose *Edge-TM*, an adaptive hardware/-software error management policy that (*i*) optimistically scales the voltage beyond the edge of safe operation for better energy savings and (*ii*) works in combination with a Hardware Transactional Memory (HTM)-based error recovery mechanism. The policy applies dynamic voltage scaling (DVS) (while keeping frequency fixed) based on the feedback provided by HTM, which makes it simple and generally applicable. Experiments on an embedded platform show our technique capable of 57% energy improvement compared to using voltage guardbands and an extra 21-24% improvement over existing state-of-the-art error tolerance solutions, at a nominal area and time overhead.

KEYWORDS

Energy efficiency, Transactional Memory, Error-resilience, Variability

ACM Reference format:
. 2017. Edge-TM: Exploiting Transactional Memory for Error Tolerance and Energy Efficiency. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 10 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The continuing ability to shrink transistor sizes has led to extensive benefits in integrated circuit designs, including faster processors, more complex designs, and higher levels of integration. However, at the same time, devices have become more susceptible to static and dynamic variability [1]. *Static variability* derives from imperfect manufacturing processes and causes nominally identical elements (such as cores in a multi-core system) to behave differently, consuming different levels of power, and providing different levels of performance. *Dynamic variability* from wearout and temperature and voltage fluctuations combined with aggressive voltage/frequency scaling can cause timing violations on processors' critical paths leading to logic errors in the computation [5], [31], [27].

Errors begin to appear when the operating conditions (**F**requency, **V**oltage, and **T**emperature) approach the *point of first failure* (PoFF). Beyond that point (e.g., decreasing the voltage further), errors become gradually more frequent and the system's behavior can be coarsely modeled with a probability (frequency) of errors as a function of (F,V,T) [10]. In some well-optimized designs, where timing violations can happen simultaneously on multiple critical paths (e.g.,) [15], [18], [17]), the range of operating conditions between the PoFF and a massive number of errors narrows down to a single *Critical Operating Point* (COP) [22] (see Figure 1).

Traditionally, to protect devices against timing errors, designers have conservatively added guardbands to the system's operating frequency and/or voltage, which results in wasted energy and degraded performance. To mitigate the pessimism of guardbands, many works have proposed circuit-level error detection and correction (EDAC) techniques [2, 6, 7, 9, 10, 29]. These techniques introduce significant energy and delay overheads for error correction and while they can handle sporadic errors, they cannot deal with massive errors such as those from COP-induced violations [28].

Software techniques offer higher flexibility and/or better capability to adapt to dynamically changing operating conditions. Most prior work targets intermittent timing errors and proposes solutions to prevent errors via careful workload allocation [4, 8, 16, 23, 24]. A combined hardware/software approach by Papagiannopoulou *et al.*[20] proposes a reactive technique that leverages hardware transactional memory (HTM) to rollback a processor's state to a prior safe point if errors are encountered. Specifically, the processor's voltage is scaled down while keeping frequency constant, up to the PoFF, at which time the processor core enters a recovery mode that restores the core to a safe voltage level. However, since the authors of [20] focus on a COP model, there is no advantage in scaling down the voltage beyond the PoFF. Other works proposing transactional memory (TM)-based fault tolerance (e.g., [30, 32, 34]) do not evaluate energy consumption (our major goal) and consider transient and permanent faults rather than intermittent timing errors.

In this paper we present *Edge-TM*, a HW/SW technique that relies on HTM rollback mechanisms for error correctionn in errant transactions. Different from traditional HTM, Edge-TM is not aimed at protecting shared data in concurrent programming, thus it replaces traditional conflict detection logic with simpler architectural support for error detection. Further, Edge-TM features error management policies that aggressively apply dynamic voltage scaling (DVS) beond the point of first failure for better energy savings. The policy monitors transaction aborts and commits to estimate the

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

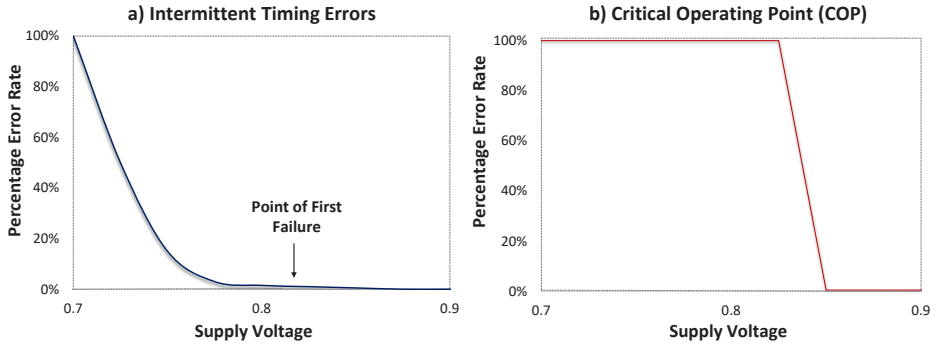

Figure 1: Error rate (%) vs. supply voltage for intermittent timing errors and the COP.

experienced error rate and decides whether to lower, maintain or raise the voltage level. This feature makes our policy capable of dealing with COP systems as well as those experiencing intermittent timing errors.

Through a set of simulations using power/performance numbers extracted from a silicon implementation of the target embedded platform, we show that our proposed scheme can achieve up to 57% improvement in energy compared to using voltage guardbands. Moreover, it can achieve a 21% improvement compared to a policy that increases the voltage immediately after the first failure and a 21%-24% improvement over other state-of-the-art error-tolerance solutions. An overhead characterization of our proposed scheme shows that it induces a modest area and time overhead, comparable to existing techniques.

The rest of the paper is organized as follows. Section 2 provides a background discussion on related work. Sections 3 and 4 describe the Edge-TM design. Section 5 presents our simulation results and Section 6 concludes our paper.

2 BACKGROUND

Intermittent faults occur due to static and dynamic variability and can be activated or de-activated by voltage, frequency or temperature fluctuations. These faults manifest as timing violations on the processor's critical paths; as the voltage is scaled down, intermittent timing errors initially emerge at low rates that later increase exponentially as the voltage is lowered further (Figure 1a). According to the "COP hypothesis" [22], in large CMOS circuits there exists a critical operating frequency f_c and a critical voltage V_c for a fixed ambient temperature T such that any frequency above f_c or voltage below V_c causes massive errors (Figure 1b). This behavior may be especially prevalent in well-optimized designs, where timing violations can happen simultaneously on multiple critical paths [15] [18] [17].

Error detection can be done at the circuit level using techniques that continuously monitor path delay variations. Examples include Razor flip-flops [6, 7, 9, 10], error detection sequential circuits (EDS) [2] or tunable replica circuits (TRC) [29]. Among error-correction circuits, Razor [9] employs counterflow pipelining, a recovery mechanism that uses a bi-directional pipeline to flush errant instructions, but incurs high energy/power overhead and requires modifications to the processor's pipeline. Other works [3, 7] proposed techniques such as instruction replay at half clock frequency or multiple-issue instruction replay. These techniques introduce significant energy and delay overheads for error correction and while they can handle sporadic errors, they cannot deal with massive errors such as those from COP-induced violations [28].

Several software techniques have been proposed to provide robustness to timing errors due to dynamic variations. Early proposals lacked generality and online adaptation capabilities [4][16], or offered costly recovery mechanisms (overheads up to thousands of cycles) [8][23]. Rahimi *et al.* explored fine-grained mechanisms to outline the notion of software vulnerability [24], and explored the use of OpenMP extensions to reduce the recovery cost incurred by HW-based error-correction techniques. Compared to our proposal, their solution requires the availability of hardware error correction. As an additional consequence, it can only deal with sporadic timing errors, for which Razor-like correction circuitry is solely effective [14]. Our approach, in contrast, fully relies on SW policies operating on top of minimal HTM-based designs, and can deal with both sporadic timing errors as well as COP.

Most previous works proposing transactional memory (TM)-based fault tolerance (e.g., [30, 32–34]) do not focus on reducing energy consumption (our major goal) and focus on transient and permanent faults rather than on intermittent timing errors. Yalcin *et al.* [32] consider how TM-based error correction could potentially improve energy efficiency, but do not provide an implementation. Papagiannopoulou *et al.* [20] study HTM-based recovery from COP with an emphasis on energy savings. Compared to this prior work, we broaden the scope of the studied effects to intermittent errors (besides COP), which allows us to follow a less conservative approach that optimistically lowers the voltage beyond the PoFF for better energy savings.

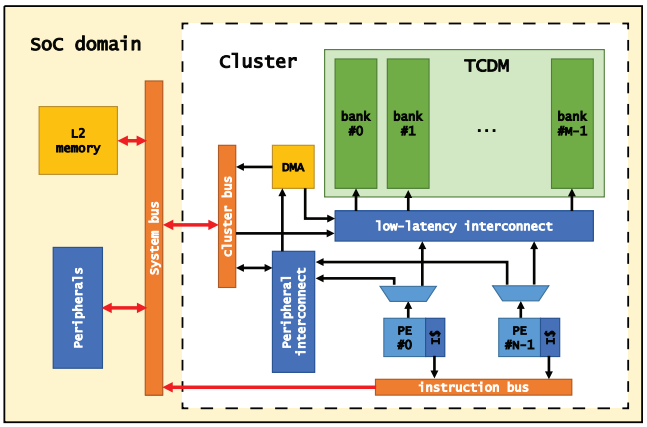

Figure 2: PULP System-on-Chip architecture.

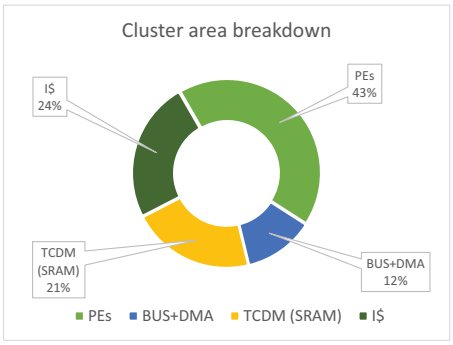

Figure 3: Breakdown analysis of the PULP SoC area.

3 EDGE-TM ARCHITECTURE DESIGN

Edge-TM consists of an integrated HW-SW approach for energy-efficient program execution on low-power, embedded shared memory SoCs. Application developers abstractly write their programs based on the widespread OpenMP API [19]. At the boundaries of OpenMP constructs for parallelism, a compiler transparently inserts function calls to mark the start and end of a *resilient transaction* (RTx). These function calls are directed to an underlying runtime system (RTS) which transparently manages RTxs in an error-resilient manner. Specifically, the RTS maintains a core-level SW error-management policy which optimistically lowers the voltage in small steps for energy savings.[1] Since the operating frequency is not scaled down with the voltage, this will incrementally trigger timing errors.

To make the system resilient to such errors, *Edge-TM* combines circuit-level error detection techniques implemented in the pipelines of each processor with a modified HTM infrastructure for error correction. A snapshot of the processor state is taken before each transaction is started. If an error occurs during transaction execution, this safe state can be restored through the underlying HTM mechanism that is described next. To limit the overhead of this mechanism, its key functionalities are implemented in HW, as an extension to the baseline platform.

We next describe the baseline parallel ultra-low-power platform (PULP) [25] targeted in this work (Section 3.1), followed by a presentation of the extensions introduced to support error tolerance, i.e., error-detection (Section 3.2) and error-correction (Section 3.3). The software policies for error-aware voltage scaling are presented later in Section 4.

3.1 The PULP Architecture

PULP is a scalable parallel computing fabric, organized as a set of *clusters* [26]. Figure 2 shows the main building blocks of a single-cluster PULP instantiation. A cluster includes a parametric number of processing elements (PEs) – typically 8 – consisting of an optimized RISC microarchitecture. PEs feature private instruction caches, while to avoid memory coherency overheads private data caches are replaced by a shared, multi-banked tightly coupled data memory (TCDM). The TCDM is configured as a 64KB, 16-bank explicitly managed SRAM (a scratchpad). As the TCDM features as many R/W ports as the number of memory banks, and the number of banks is twice the number of PEs, single-cycle access latency is ensured for concurrent accesses to different banks. At the top level, a 256KB L2 memory and other peripherals for off-chip communication can be accessed via DMA.

Figure 3 shows the contributions of the hardware components to the total area of a PULP cluster. Overall, the main contributors to the area and energy consumption are the cores and their private instruction caches. For this reason, Edge-TM focuses on scaling the voltage at the individual core level. The shared L1 TCDM is powered through a separate voltage domain and it is always kept at a stable, safe level.

To explore our extensions to this baseline platform for energy-efficient error tolerance we use the PULP simulator, which provides cycle-accurate modeling of the various architectural blocks. The simulator has been extended with energy models derived from a 28nm UTBB FDSOI (STMicroelectronics technology) implementation of the described platform, thus enabling realistic power measurements.

In the following section we describe how the baseline PULP cluster has been extended in the simulator to enable error tolerance. Figure 4 highlights the key extensions using colored blocks.

3.2 Error detection

For error detection, we assume that each core is equipped with runtime error-detection circuitry, such as *error-detection sequential* (EDS) [3] (Figure 4). EDS-based designs distinguish between *critical* and *non-critical* errors. Critical errors are those that happen on the control part of the processor pipeline (e.g., write-back). Since a timing violation along one of these critical paths makes the software flow unpredictable, we cannot handle such errors at the OS, middleware or application level. For this reason, critical errors are prevented by

[1] We assume 20 mV steps, well in line with modern voltage regulators [13].

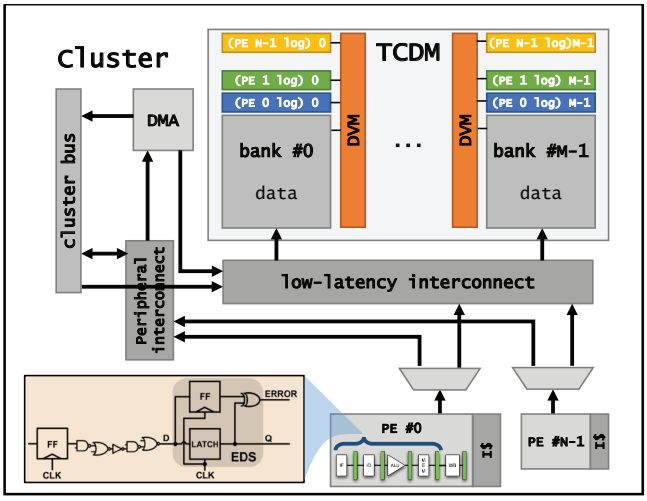

Figure 4: Extensions to the PULP cluster for error-tolerance.

designing such pipeline stages with additional timing guard-bands [3] [29]. This ensures that dynamic-variation timing failures do not occur in these stages. Non-critical errors are those that are protected by EDS circuits, as they only affect the result of computations (e.g., load/store, execute) and can then be safely corrected by the software. The programming model disciplines those cases where control flow issues might be originated from the software by disallowing patterns like pointer-based function calls.

To implement EDS in our simulator, we assume a probabilistic error model, similar to other works [20] [24]. Our error model follows the probability curves reported by Fojtik *et al.* [10], which we have adapted to the operating voltage and frequency range of our target platform. These curves provide the expected error rate as a function of supply voltage levels, assuming intermittent timing errors (Figure 1a). The supply voltage level where the point of first failure is expected post-fabrication, for each frequency and temperature point is extrapolated through exhaustive testing on the PULP chips.

When the EDS detects an error[2], it generates an interrupt to the core. The core's pipeline is flushed and its program counter is set to jump to an interrupt service routine (ISR) where we implement the HTM rollback mechanism (Section 3.3). Note that it must be ensured that the ISR is executed in an error-free manner. In the PULP 28nm chip this can be done by applying forward body biasing to achieve a temporary performance boost [26]. Our simulator models the overhead cycles implied by this technique.

3.3 Error Correction

Hardware transactional memory is a well known speculative execution mechanism for synchronizing shared memory data access in multi-core environments [11]. HTM allows cores to execute critical sections in parallel, as transactions. If a core's transaction completes without encountering data conflicts with other transactions, then it commits and its speculative changes become permanent. If a conflict occurs, one or more of the conflicting transactions is rolled back and restarted.

HTM designs are appearing in high-end, commercial processors (e.g., Intel's Haswell, IBM's Blue Gene/Q), which shows the maturity of the technology[3]. The adoption of traditional HTM in resource-constrained, low-power embedded designs is much more uncertain, as it is hindered by the high area/energy cost and the poor scalability of underlying coherent cache systems. The proposed approach relies on two key features to make HTM suitable for embedded SoCs: (*i*) a state-of-the-art HTM design that is tailored to the characteristics of such SoCs [21], where poorly-scalable data caches are replaced by a shared L1 scratchpad memory, and (*ii*) a further simplified HTM design specifically tailored for managing error correction. Our target architecture described in Section 3.1 can support both these features.

Traditionally, HTM is based on three key components: 1) a *bookkeeping* mechanism to keep track of read/write data accesses and detect conflicts, 2) a *data versioning* technique to keep track of original and speculative data versions for recovery in case of conflicts and 3) a *check-pointing and rollback* mechanism to recover from data conflicts and retry failed transactions.

Note that *Edge-TM* implements a modified version of HTM for error detection and correction (i.e., data synchronization conflicts are not detected), and thus it does not require the bookkeeping mechanism to detect conflicting accesses to shared data[4]. As we discuss in Section 5.1, this also leads to a more lightweight design. We now describe the modifications required for data versioning and check-pointing.

Data Versioning. We use a distributed *logging* scheme to enable data versioning. *Logs* are distributed among the TCDM banks of the PULP cluster and each bank keeps a fixed-size log space for each core in the system, as shown in Figure 4. The first time an address is written in a transaction, its original value needs to be saved in the log. Since this requires a buffering capability equivalent to the transaction's *write-set* size, the logs are quite small compared to traditional HTM.

The log saving and restoration process is done independently at each memory bank through dedicated *Data Versioning Modules* (DVM). Each bank's DVM is a control block that monitors transactional accesses to the bank and manages the cores' logs that reside in that bank. It is also responsible for restoring the log data of the cores that abort their transactions and cleaning the logs of the cores that commit their transactions. All DVMs work in parallel and fully independently, which makes them very fast and efficient.

[2]That is, when the error model injects an error event in the simulated processor pipeline.

[3]If such systems provided HTM and error detection circuits, we could re-engineer our SW policy to operate on top of memory synchronization management.

[4]As our programming frontend is based on OpenMP, we employ traditional OpenMP critical sections and locks to protect concurrent accesses from multiple threads to shared data.

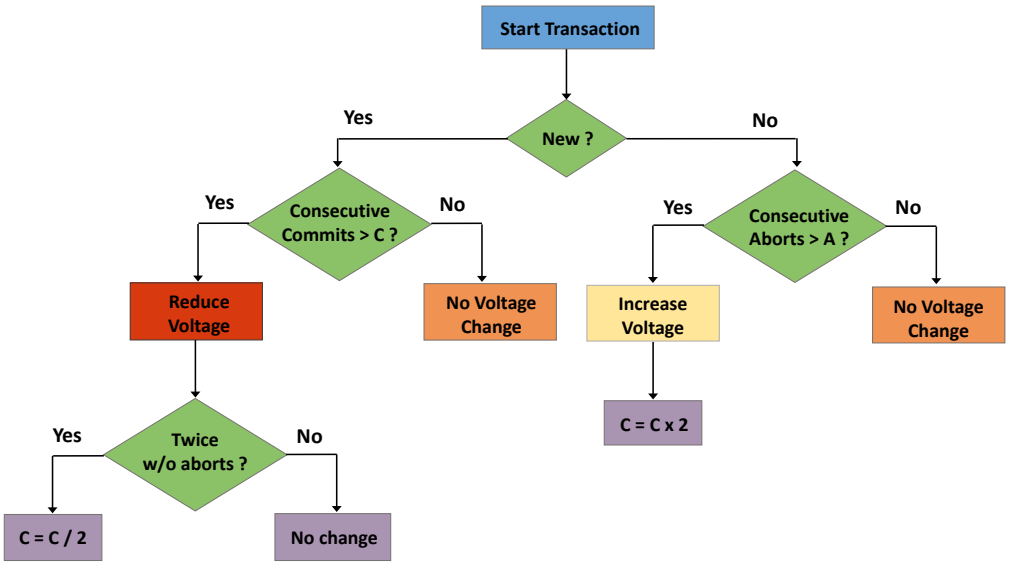

Figure 5: The flowchart of the 'Thrifty uncle/Reckless nephew' policy.

Checkpointing and Rollback. Since specification version 3.0, OpenMP has adopted a *task*-centric execution model. Every program execution unit is explicitly or implicitly represented as a *task* in the runtime system. We build upon this feature to transparently wrap each OpenMP *task* within an *error-resilient transaction*. Transaction size regulation can be easily achieved at the task level or within parallel loops (by grouping independent loop iterations into a single transaction). When a core starts a transaction, its internal state (i.e., program counter, stack pointer, internal registers, stack contents) is saved to be retrieved in case of errors (*check-pointing*). If no errors are detected by the end of the transaction, the transaction *commits*, i.e., the checkpointing information and the logs of the committing core are discarded, and all speculative data changes become permanent. If an error is detected during transaction execution, the transaction *aborts*, i.e., the speculative data changes are discarded. The *rollback* mechanism restores the core's state and the original data from the core's logs back to their original addresses (the transaction rollback and restoration process is done at a safe voltage level, as discussed earlier). The core is then ready to retry the transaction.

On top of the basic error detection and correction mechanisms described so far, *Edge-TM* implements software error management policies, which apply dynamic voltage scaling (DVS) appropriately before an aborted transaction is restarted. These software error management policies are described next (Section 4).

4 EDGE-TM DVS POLICIES

POFF policy. We first implement a simple, baseline error policy, that operates just above the edge of failure. Starting from a safe reference level, the supply voltage is gradually scaled down in small steps to save energy. When the first error occurs (i.e., the PoFF is reached), the voltage is immediately increased by one step. This way the system operates just above the edge of failure throughout the rest of the operation, without taking the risk of allowing further timing errors. Might a new error emerge (e.g., due to temperature fluctuations), the voltage is increased again by one step. This policy is similar to the technique proposed in [20], but while those errors were handled in a *lazy* manner (i.e., error recovery was not triggered until transaction commit time), here we treat them in an *eager* manner (i.e., immediately upon detection). We use this simple transactional memory policy that operates just above the PoFF (i.e., the *POFF policy*) as a reference to compare against [20].

Thrifty Uncle/Reckless Nephew (TURN) policy. Edge-TM offers an adaptive error policy that optimistically lowers the voltage beyond the PoFF, tolerating timing errors and making voltage adjustment decisions based on the feedback provided by the runtime characteristics of the transaction. As the voltage is scaled down for energy savings, the transaction abort rate grows due to increased rate of errors, which in turn leads to increased energy consumption from transaction recovery and re-execution. To reflect these conflicting behaviors we call our new adaptive approach the *Thrifty Uncle/Reckless Nephew (TURN)* policy[5]. The policy has two parts:

- The *reckless nephew* optimistically scales the voltage down for better energy savings, based on the number of consecutive successful transaction commits.
- The *thrifty uncle* moderates the energy loss from the increased number of transaction aborts (due to over-aggressive voltage scaling), by setting up a threshold based on the number of transaction aborts and commits.

Figure 5 shows a flowchart of how the proposed policy works. Starting from a safe reference level, the voltage is

[5]A reference to the thrifty uncle Scrooge McDuck and reckless nephew Donald Duck of Disney's Duck family.

scaled in small steps. When a transaction starts, the policy checks whether this is a failed transaction that is re-starting or a new one. If this is a new transaction, the nephew checks the number of consecutive successful transactions that have proceeded this one, (i.e., the number of consecutive commits). If this number is greater than a pre-defined threshold C, then the voltage can be safely reduced by one step. Otherwise, no voltage change is allowed. If this is a failed transaction that is being re-executed, then the uncle must decide whether the transaction should be re-executed at the same voltage or the voltage should be increased. If this transaction has a record of consecutive aborts that is greater than a pre-defined threshold A, then the uncle increases the voltage by one step. Otherwise, the transaction is re-executed at the same voltage level.

Every time the consecutive abort threshold A is exceeded the uncle realizes that the current voltage level is likely dangerous and not only increases the voltage, but also doubles C to make it more difficult for the nephew to later come back to that level. If the error rate is reduced in the future, e.g., due to a temperature variations or because transactions become smaller, then a lower voltage level might be sustainable. In that case, C must be reduced again to allow for an easier transition to lower voltage levels. If the voltage is reduced twice in a row without any aborts in between, then the uncle divides the threshold C by 2.

To decide a convenient starting value for threshold C we use a simple heuristic based on the potential energy savings (for a transaction) achieved by scaling voltage one step down. Such savings can be approximated as:

$$Savings = 1 - \frac{P_{dyn,2}}{P_{dyn,1}} \approx 1 - \frac{V_2^2}{V_1^2},$$

where:

$$V_2 = V_1 - V_{STEP}.$$

Since our policy works at the granularity of transactions, to save the energy consumed by one transaction executed at V_1, we need to run at least C transactions at V_2 in a row with no errors (i.e., if a single abort is experienced then the savings are lost). In other words, C should be big enough to ensure that the cumulated savings achieved by voltage scaling surpass the energy of one single, unscaled transaction. Thus, C can be chosen as:

$$C = \frac{1}{Savings} = \frac{V_1^2}{V_1^2 - V_2^2}.$$

Within the operative voltage range and V_{STEP} considered in our setup, $C = 20$.

Threshold A could be determined as the minimum number of aborts for which the accumulated energy at the current voltage level exceeds the energy that would have been spent at the next higher voltage level. However, in practice, we have noticed that accumulating energy through online energy monitoring introduces non-negligible additional overhead and the threshold does not vary within or among different programs. Consequently, we empirically found that A should be set to 3.

5 RESULTS

We next provide an evaluation of Edge-TM in terms of energy consumption as well as area and performance overhead. Specifically, Section 5.1 presents an overhead characterization of Edge-TM in terms of area and time, and compares it with previous works ([20, 24, 33]). Section 5.2 provides an evaluation of the point-of-first-failure (*POFF*) and uncle/nephew (*TURN*) policies of Edge-TM in terms of energy consumption. The policies are compared with previous works using a conservative steady-voltage technique that relies on guardbands (*GDBS policy*) as a baseline. In Section 5.3 we describe how parameter tuning in Edge-TM can affect the obtained energy savings. Specifically, we study how transaction size and various transaction workload patterns can affect the obtained energy savings.

To isolate the effects of running our various error correction schemes, we run our experiments on a single-core instance of the platform[6]. Note that our technique can be applied to multiple cores simultaneously; if errors are found in one core, recovery happens independently from other cores. For the evaluation we use three benchmarks from the image processing domain: *Rotate* (image rotation), *Strassen* (matrix multiplication) and *Mahalanobis-Distance* (cluster analysis and classification). To study the behavior of the system with controlled transaction size and memory bounds we also use a modified version of the *Eigenbench* synthetic benchmark [12].

5.1 Overhead characterization

Table 1 shows the area and time overheads implied by Edge-TM, compared to the three most closely related approaches in literature: *i*) the work from Rahimi *et al.* on vulnerability-aware, error-tolerant task scheduling (**VOMP**) [24]; *ii*) the work from Yalcin *et al.* on revisiting transactional memory for fault tolerance (**FaulTM**) [33]; *iii*) the work from Papagiannopoulou *et al.* on revisiting transactional memory for timing error-tolerance (**PWF**) [20].

Concerning error detection, all techniques except *FaulTM* rely on the use of EDS circuitry at the core level. This circuitry is known to introduce low area overhead ($\approx$ 2.2%) to each core [3], which overall results in less than 1% area overhead for the whole cluster level. The time overheads for detecting errors with EDS are also negligible. *FaulTM* implements error detection by redundantly executing transactions on two threads and comparing the write sets after a synchronization barrier upon commit. From the point of view of the implied area overhead, this technique requires full-fledged transactional memory support for data versioning and conflict detection (unlike Edge-TM and PWF, which only implement transactional memory support for rollback). If we were to implement FaulTM on a cacheless MPSoC that we are targeting for our work, it would require distributed data versioning across the TCDM banks (as was proposed in [21]). This would imply an additional $r \times 1 + N + logN$ bits per TCDM bank, where r is the number of data lines (here, words) in each bank and N is the number of cores. This

[6] https://github.com/pulp-platform/pulpino

	ERROR DETECTION			ERROR CORRECTION		
	Technique	**Area overhead [%]**	**Time overhead [cycles]**	**Technique**	**Area overhead [%]**	**Time overhead [cycles]**
Edge-TM	HW (EDS)	2.2% *(core)*; <1% *(cluster)*	–	HW (HTM rollback, in-place updates)	12.5% *(mem)*; 2.65% *(cluster)*	sizeof(*write-set*)×2
VOMP	HW (EDS)	2.2% *(core)*; <1% *(cluster)*	–	HW (Multiple-Issue Replica Instructions	1.4% *(core)*; <1% *(cluster)*	3×<*pipeline-depth*>
FaulTM	HW (HTM conflict detection, LAZY) + SW (write-set comparison)	37.2% *(mem)*; 8% *(cluster)*	WCET(RTx)+ sizeof(*write-set*)×2	HW (HTM rollback, LAZY)	12.5% *mem*; 2.65% *(cluster)*	–
PWF	HW (EDS)	2.2% *(core)*; <1% *(cluster)*	–	HW (HTM rollback, LAZY)	12.5% *mem*; 2.65% *(cluster)*	sizeof(*write-set*)×2

Table 1: Comparison of area and time overheads for various error detection and correction techniques.

results in a TCDM area increase of ≈ 37% and an overall cluster area increase of ≈ 8%.

The time overheads for the FaulTM technique are also very relevant. Besides the redundant execution of transactions, which is clearly not a viable path when energy minimization is the target, FaulTM implies a lazy conflict detection scheme, which implies that errors that might have occurred are only detected at commit time. As a consequence, the overhead of this technique needs to factor in the worst case execution time of the transaction, plus the overheads for the barrier. Finally, comparing the write sets of the two transactions requires $k \times$ *sizeof(write-set)* cycles, where k is the number of cycles required for a single comparison and has a minimum value of 3 (two reads, one compare).

Concerning error correction, all techniques excpet *VOMP* rely on log-based transactional rollback to undo the effects of an errant transaction and restart it. State-of-the-art implementations for cache-less MPSoCs [21] implement this logging feature fully in the TCDM banks, with an area overhead that is proportional to the size of the logs. Considering 1KB logs per thread this implies 12.5% TCDM area increase, which translates in 2.65% cluster area overhead.

Both *Edge-TM* and *PWF* rely on eager data versioning (transactions write in-place in the shared memory and save the original values in the logs). This makes the common case of a successful execution (commit) faster and the case of an errant execution (abort) a bit slower, as the logs have to be restored in the memory. The time overhead for this operation is $k \times$ *sizeof(write-set)* cycles, with $k = 2$ (one read, one write). *FaulTM* uses lazy data versioning, buffering transactions updates which have to be published to shared memory in case of a commit. While this makes the common case slower, it also speeds up error correction, as it is sufficient to drop the content of the logs, with zero additional time overheads.

Next, we discuss the overheads associated with the two Edge-TM policies (POFF and TURN) and compare them with the GDBS policy. Based on measurements, TURN has an 8% execution time overhead over GDBS, while POFF introduces only a 1% overhead. These overheads are due to the extra time needed to setup transactions (i.e., checkpointing/logging), the time introduced by the policy (i.e., time to execute the policy and adjust the voltage), and the delays associated with recovery and re-execution of failed transactions[7]. Each voltage adjustment takes 10 clock cycles. TURN is on average 7% slower than POFF because (*i*) it experiences more transaction aborts/re-executions since it operates at lower voltage levels, and (*ii*) it makes more voltage adjustments. However, the TURN policy spends less than 2% of the total execution time making voltage adjustments (i.e., it quickly learns what is the most sustainable voltage level).

This analysis shows us that our Edge-TM approach requires relatively modest area overhead and is the same or less than that of comparable policies.

5.2 Energy consumption

Figure 6 compares the *TURN* and *POFF* policies, using as a baseline a conservative steady-voltage technique which relies on guardbands (*GDBS policy*) to absorb the effects of static and dynamic variations. For completeness, we also include in the comparison the three error-tolerance techniques published in literature that are closest to ours, and that we introduced in Section 5.1 (i.e., FaulTM, VOMP and PWF). Consistent with the setup described in the original papers, for *FaulTM* and *VOMP* the voltage is not scaled, but it is kept stable at a level where errors are sporadic (this level corresponds to the point of first failure in our setup). *PWF* behaves exactly like our *POFF* policy, with the only difference that *PWF* employs a lazy error detection scheme. We have enhanced our simulation infrastructure to capture the time and energy overheads implied by the error detection and recovery mechanisms discussed in Section 5.1.

Note that for this set of experiments, we ran two different configurations for *Eigenbench*, one with a fixed transaction size and one with random transaction sizes. The importance of transaction size in the obtained results is discussed later in Section 5.3.

From Figure 6 we observe that both *Edge-TM* policies achieve significant energy savings compared to GDBS. Results for the POFF policy are in line with those for *PWF*, confirming that the conservative approach taken in [20] for

[7]Note that variations in execution time between different policies are not affected by changes in frequency, as we only scale the voltage.

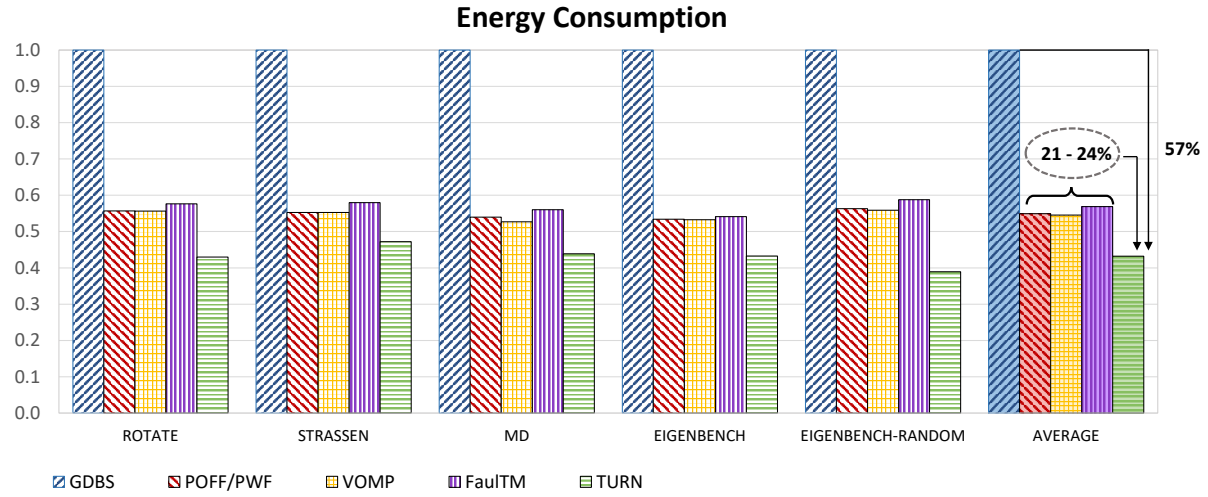

Figure 6: Energy consumption of various policies normalized to the baseline GDBS configuration.

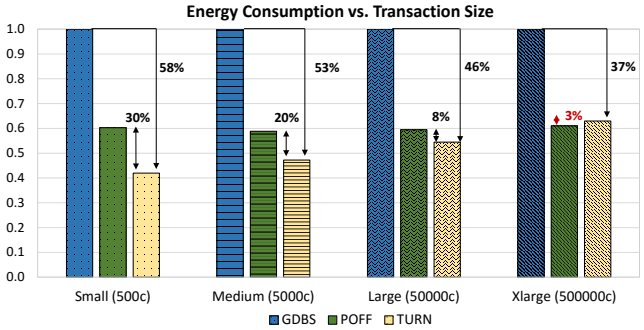

Figure 7: Energy consumption (normalized to GDBS) for different transaction sizes.

the COP assumption is also valuable for intermittent errors. *PWF* has nearly identical energy consumption to *POFF*, because both policies disable further voltage scaling upon encountering the first (few) errant transactions, which makes the additional overheads for lazy error detection in *PWF* negligible. *VOMP* slightly decreases the energy consumption, as it permanently operates at a lower voltage level by correcting the sporadic errors. A similar operating mode is valid for *FaulTM* as well. Overall, *TURN* improves on average by 57% over *GDBS*, by 21% over *POFF*/*PWF*/*VOMP* and by 24% over *FaulTM*. These experiments demonstrates that in presence of intermittent errors optimistically lowering the voltage beyond the PoFF pays off, if the technique is controlled to prevent wasteful transaction aborts.

5.3 Energy savings vs. transaction size

Transaction size has a significant impact on the TURN policy. Intuitively, large (i.e., long-lived) transactions have higher probability to experience an error when operating at low voltages, while several small transactions might successfully commit at the same operating point.

To test the effect of transaction sizes on the opportunities for energy savings, we configure the *Eigenbench* to operate with four different transaction sizes: small (500 cycles), medium (5,000 cycles), large (50,000 cycles) and extra large (500,000 cycles). Figure 7 shows the energy consumption for GDBS, POFF and TURN normalized to GDBS for each transaction size. The TURN policy achieves 58%, 53%, 46% and 37% improvement over GDBS for small, medium, large and extra large transaction size respectively. The energy improvement over the POFF policy is 30%, 20%, 8% and –3% respectively. The extra large transaction size shows the point for which the optimistic TURN policy starts behaving worse than the conservative POFF policy (energy consumption is increased by 3%). Here, the transaction size is too large for it to complete often enough without encountering errors past the point of first failure. In this case, TURN and POFF both operate above the edge of failure, but TURN has additional overheads because it takes extra iterations before the policy learns that it should operate reliably at the higher safe voltage level.

These results roughly suggest which energy savings can be expected given a particular transaction size. To further study this effect and to test the TURN policy for robustness to dynamically varying transaction sizes, we conduct another experiment. Specifically, we consider three *patterns* according to which transaction sizes change over time: *Increasing*, *Decreasing* and *Random*. For the first two, transaction sizes are increased/decreased by one thousand cycles at each iteration, while for the third, sizes are randomly determined at each step within the considered *range*. We consider two transaction size ranges: *narrow* and *broad*. Referring to Figure 7, the narrow range encompasses the sizes considered in the two central groups of bars and it is representative of most of the practical real applications in the embedded domain. The broad range spans the whole set of sizes considered in the same figure and it accounts for scenarios where a big variance in transaction size has to be expected.

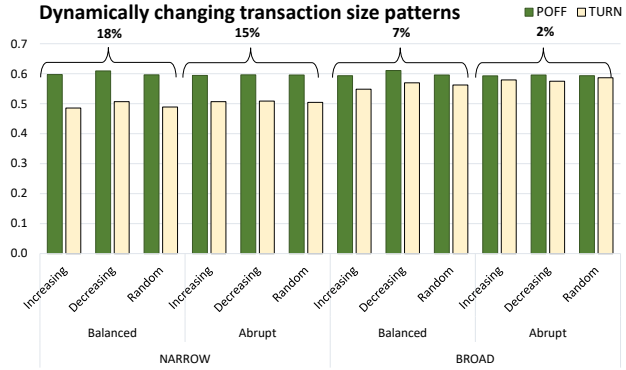

Figure 8: Energy consumption (normalized to GDBS) for various transaction size patterns.

Every run consists of 100000 transactions. To avoid being completely dominated by the energy and time behavior of the larger transactions, we balance the workload by having more small transactions than large transactions (*balanced* configuration). In essence, we make the cycles spent on each transaction size approximately the same. This often results in changing the transaction size after several executions with the current size, which has a beneficial effect on the behavior of the TURN policy (more steady samples from which to adjust). To remove this bias we thus also consider a configuration where transitions between one transaction size and the other are *abrupt* (i.e., we change transaction size at every iteration).

Figure 8 shows the results for this experiment. We plot energy for POFF and TURN, normalized to GBDS. Focusing on the *Narrow* range, when the workload distribution is *balanced* we achieve the highest savings, around 50% compared to GBDS and around 18% compared to POFF. These results are in line with what was shown in Figure 7, as the considered transaction sizes for these experiments lie in the central area of that plot. Note that when changes in transaction size are *abrupt*, TURN experiences a higher number of aborts, which reduce to 15% the savings compared to POFF.

A similar trend can be observed for the *broad* range, but the higher variance in transaction size shrinks the margins for operating below the point of first failure, which brings us closer to the POFF policy. Here, TURN achieves on average 7% improvement over POFF in the balanced workload cases and 2% in the abrupt ones. Of the three patterns, *Increasing* seems to consistently do slightly better than the others. To explain this, Figure 9 shows the behavior of the TURN policy for the three patterns, considering the *broad* range and the *balanced* workload. These plots show execution cycles (time) on the x-axis, voltage level on the primary y-axis (left, blue curve) and number of aborts and commits on the secondary y-axis (right, circular and triangular cloud points).

For the *Increasing* patterns transactions are initially very small, which leads the TURN policy to quickly reach very low voltage levels. Aborts are mainly clustered towards the initial phases of the program, when there is a first transition in transaction size that makes them too lengthy to execute error-free at that voltage level. The system then operates steadily without changes until a second transition brings the voltage one level up again; no further adjustments are required from that point on. The *Decreasing* pattern experiences all its aborts towards the late stages of the program (again, when the transactions are small). From the point of view of the energy consumption this is less convenient, as (compared to the *Increasing* patterns) the time it takes for the policy to stabilize is longer (note that the x-axis is logarithmic) and while doing so it lingers at higher voltage levels. Note that the number of aborts is always very small compared to the commits (right y-axis is also logarithmic). The *Random* pattern, as expected, experiences aborts throughout the entire program life. However, the policy is capable of reacting timely to workloads variations, which in the end leads to no relevant difference in energy savings compared to the other patterns.

We conclude that even though it is best to keep transactions as small as possible, our technique is capable of energy savings for various transaction sizes and access patterns. Adding the capability to dynamically adjust transaction size at runtime would provide an extra level of freedom in designing runtime policies for online selection of the most suitable transaction granularity. We leave this exploration for future work. As future work, we would also like to experiment with combined frequency and voltage scaling.

6 CONCLUSIONS

In this paper we propose Edge-TM, an adaptive HW/SW error management scheme that is based on HTM for error recovery that is capable of dealing with intermittent timing errors and the COP. Edge-TM encompasses two SW error management policies. The first, called POFF, is a simple policy that scales down the voltage for energy savings while allowing operation just above the edge of failure. The second, called the 'Thrifty uncle/Reckless nephew' (TURN), is a more risky, adaptive policy that optimistically scales the voltage beyond the edge of failure and adjusts it using the feedback provided by HTM, with the goal to achieve better energy savings. Experiments on an embedded platform show that Edge-TM can achieve significant energy savings compared to existing techniques, with a nominal area and time overhead. Specifically, POFF and TURN achieve a 45% and 57% improvement over conservative guard-banding, respectively. TURN is also capable of an extra 21-24% improvement over existing state-of-the-art error tolerance solutions. Our findings indicate that the obtained energy savings are affected by transaction size and different workload patterns. We conclude that the combination of carefully tuned SW error management policies and HTM-based error recovery can provide significant energy savings compared to existing solutions.

REFERENCES

[1] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *DAC*, pages 338–342, June 2003.
[2] K. Bowman, J. Tschanz, N. S. Kim, J. Lee, C. Wilkerson, S. Lu, T. Karnik, and V. De. Energy-efficient and metastability-immune

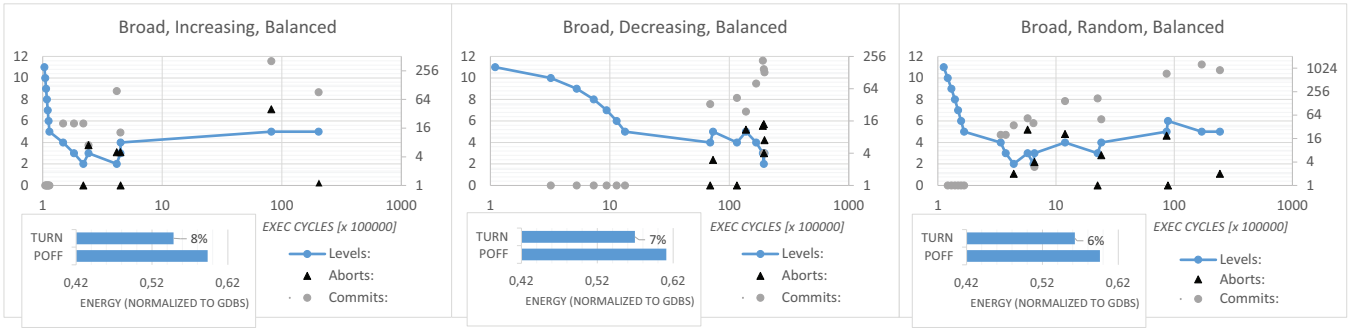

Figure 9: Execution cycles (x-axis), voltage level (primary y-axis/blue curve), number of aborts and commits (secondary y-axis/circular and triangular cloud points) for different transaction size patterns.

resilient circuits for dynamic variation tolerance. *IEEE JSSC*, 44(1):49–63, Jan 2009.

[3] K. Bowman, J. Tschanz, S. Lu, P. Aseron, M. Khellah, A. Raychowdhury, B. Geuskens, C. Tokunaga, C. Wilkerson, T. Karnik, and V. De. A 45nm resilient microprocessor core for dynamic variation tolerance. *IEEE JSSC*, 46(1):194–208, Jan 2011.

[4] F. Chaix, G. Bizot, M. Nicolaidis, and N. E. Zergainoh. Variability-aware task mapping strategies for many-cores processor chips. In *IOLTS*, pages 55–60, July 2011.

[5] C. Constantinescu. Intermittent faults and effects on reliability of integrated circuits. In *RAMS*, pages 370–374, 2008.

[6] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning dvs processor using delay-error detection and correction. *IEEE JSSC*, 41(4):792–804, April 2006.

[7] S. Das, C. Tokunaga, S. Pant, W. H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw. Razorii: In situ error detection and correction for PVT and SER tolerance. *IEEE JSSC*, 44(1):32–48, Jan 2009.

[8] S. Dighe, S. Vangal, P. Aseron, S. Kumar, T. Jacob, K. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. De, and S. Borkar. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *JSSC*, 46(1):184–193, Jan 2011.

[9] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *MICRO*, pages 7–, 2003.

[10] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw, and D. Sylvester. Bubble razor: Eliminating timing margins in an arm cortex-m3 processor in 45 nm cmos using architecturally independent error detection and correction. *IEEE JSSC*, 48(1):66–81, Jan 2013.

[11] M. Herlihy and J. E. B. Moss. Transactional memory: architectural support for lock-free data structures. In *ISCA*, pages 289–300, 1993.

[12] S. Hong, T. Oguntebi, J. Casper, N. Bronson, C. Kozyrakis, and K. Olukotun. Eigenbench: A simple exploration tool for orthogonal tm characteristics. In *IISWC*, pages 1–11, 2010.

[13] Intel. Voltage regulator module and enterprise voltage regulator-down 11.1, 2009. http://www.intel.com/Assets/en_US/PDF/designguide/321736.pdf.

[14] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori. Slack redistribution for graceful degradation under voltage overscaling. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 825–831, Jan 2010.

[15] V. B. Kleeberger, P. R. Maier, and U. Schlichtmann. Workload- and instruction-aware timing analysis: The missing link between technology and system-level resilience. In *DAC*, 2014.

[16] L. Leem, H. Cho, J. Bau, Q. Jacobson, and S. Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *DATE*, pages 1560–1565, March 2010.

[17] L. Liangzhen and P. Gupta. A case study of logic delay fault behaviors on general-purpose embedded processor under voltage overscaling. Technical report, University of California, Aug. 2014. Retrieved from http://escholarship.org/uc/item/3967v8hw.

[18] S. Narayanan, G. Lyle, R. Kumar, and D. Jones. Testing the critical operating point (cop) hypothesis using FPGA emulation of timing errors in over-scaled soft-processors. In *SELSE*, 2009.

[19] openMP. The OpenMP application program interface v.3.0. available through www.openmp.org, 2017.

[20] D. Papagiannopoulou, A. Marongiu, T. Moreshet, L. Benini, M. Herlihy, and I. Bahar. Playing with fire: Transactional memory revisited for error-resilient and energy-efficient MPSoC execution. In *GLSVLSI*, pages 9–14, 2015.

[21] D. Papagiannopoulou, T. Moreshet, A. Marongiu, L. Benini, M. Herlihy, and R. I. Bahar. Speculative synchronization for coherence-free embedded NUMA architectures. In *SAMOS*, pages 99–106, July 2014.

[22] J. Patel. CMOS process variations: A critical operation point hypothesis. web.stanford.edu/class/ee380/Abstracts/080402-jhpatel.pdf, 2008.

[23] F. Paterna, A. Acquaviva, A. Caprara, F. Papariello, G. Desoli, and L. Benini. Variability-aware task allocation for energy-efficient quality of service provisioning in embedded streaming multimedia applications. *IEEE TOC*, 61(7):939–953, 2012.

[24] A. Rahimi, D. Cesarini, A. Marongiu, R. K. Gupta, and L. Benini. Improving resilience to timing errors by exposing variability effects to software in tightly-coupled processor clusters. *JETCAS*, 4(2):216–229, 2014.

[25] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini. PULP: A parallel ultra low power platform for next generation IoT applications. In *Hot Chips*, Aug 2015.

[26] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gurkaynak, A. Teman, J. Constantin, A. Burg, I. Miro-Panades, E. BeignÃĺ, F. Clermidy, F. Abouzeid, P. Flatresse, and L. Benini. 193 MOPS/mW @ 162 MOPS, 0.32v to 1.15v voltage range multi-core accelerator for energy efficient parallel and sequential digital processing. In *COOL CHIPS*, Apr. 2016.

[27] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. VARIUS: A model of process variation and resulting timing errors for microarchitects. *IEEE TSM*, 21(1):3–13, Feb 2008.

[28] J. Sartori and R. Kumar. Overscaling-friendly timing speculation architectures. In *GLSVLSI*, pages 209–214, 2010.

[29] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De. Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance. In *SVC*, pages 112–113, June 2009.

[30] J.-T. Wamhoff, M. Schwalbe, R. Faqeh, C. Fetzer, and P. Felber. Transactional encoding for tolerating transient hardware errors. In *Stabilization, Safety, and Security of Distributed Systems*, volume 8255, pages 1–16. Springer Intl. Pub., 2013.

[31] P. M. Wells, K. Chakraborty, and G. S. Sohi. Adapting to intermittent faults in multicore systems. In *ASPLOS*, 2008.

[32] G. Yalcin, A. Cristal, O. Unsal, A. Sobe, D. Harmanci, P. Felber, A. Voronin, J.-T. Wamhoff, and C. Fetzer. Combining error detection and transactional memory for energy-efficient computing below safe operation margins. In *PDP*, pages 248–255, Feb. 2014.

[33] G. Yalcin, O. Unsal, and A. Cristal. FaulTM: Error detection and recovery using hardware transactional memory. In *DATE*, pages 220–225, 2013.

[34] G. Yalcin, O. S. Unsal, and A. Cristal. Fault tolerance for multi-threaded applications by leveraging hardware transactional memory. In *Computing Frontiers*, pages 4:1–4:9, 2013.