IgnoreTM: Opportunistically Ignoring Timing Violations for Energy Savings using HTM

Dimitra Papagiannopoulou*, Sungseob Whang†, Tali Moreshet‡, R. Iris Bahar†
*Dept. of Electrical and Computer Engineering, University of Massachusetts, Lowell, MA 01854
†School of Engineering, Brown University, Providence, RI 02912
‡Dept. of Electrical and Computer Engineering, Boston University, Boston, MA 02215

Abstract—**Energy consumption is the dominant factor in many computing systems. Voltage scaling is a widely used technique to lower energy consumption, which exploits supply voltage margins to ensure reliable circuit operation. Aggressive voltage scaling will slow signal propagation; without coherent frequency relaxation, timing violations may be generated. *Hardware Transactional Memory* (HTM) offers an error recovery mechanism that allows reliable execution and power savings with modest overhead. We propose *IgnoreTM*, an adaptive error management framework, that tolerates (i.e., opportunistically ignores) timing violations, allowing for more aggressive voltage scaling. Our experimental results show that *IgnoreTM* allows up to 47% total energy savings with negligible impact on runtime.**

I. Introduction

Combining high performance with reduced energy consumption remains critically important for computing systems at all levels, from improving the battery life of embedded systems to reducing the power demands of servers. However, in recent years, the rapid growth of information that needs to be processed and the evolution of cloud computing, big-data, and always-on connectivity have increased the computational demands, making high performance along with energy efficient computing a pressing challenge. At the same time, transistor scaling has left devices more susceptible to the effects of static and dynamic variability [2], making fault-free computation more challenging. Maintaining fault-free operation without adding considerable overheads on performance and energy is difficult. *Approximate computing* has emerged as a promising solution to these dilemmas for applications that can sustain a slightly reduced accuracy for increases in performance and energy efficiency. Examples include machine learning, signal processing, image processing, scientific computing, data mining and analysis.

This paper proposes *IgnoreTM*, a new framework for approximate computation. IgnoreTM provides a novel error management scheme that utilizes aggressive voltage scaling to improve energy efficiency. The key insight is that recovery from critical errors, ones that cannot be tolerated, can be facilitated by lightweight mechanisms adapted from hardware transactional memory (HTM) [7]. We show that IgnoreTM achieves up to 47% total energy savings without impacting runtime. Moreover, it allows for an additional 13-18% energy savings compared to existing voltage scaling techniques that do not apply approximation but always correct timing errors, while either improving or having a negligible impact on runtime.

II. Background

Various approximate computing techniques have been proposed in literature. At the software level, early approaches focused on coarse-grained tasks [5], [10], but came with a high recovery cycle overhead, and lacked generality. Other software approaches (e.g. Green [1], Paraprox [14]) propose approximate programming models. Rahimi *et al.* [12] reuses the result of an error-free instruction to spatially correct errant instructions and thus avoid the high timing error recovery overhead. Such software-based approaches focus on code modifications for approximation, which is orthogonal to what we are proposing.

At the hardware level, there have been proposals employing voltage scaling for approximation [9], [16]. ISA extensions are used in [15] and [6] to support approximate instructions that are executed on approximate functional units and storage. All above approaches require special approximate hardware versions for the actual computation and are again orthogonal to what we are proposing. Rather, in this paper we propose an opportunistic way of dealing with errors that arise at runtime such that some errors may pass through without the need to waste runtime and energy to correct them. The hardware implementation itself remains exact; however, error recovery, when needed, is handled through our HTM-based scheme.

Other works borrow the recovery mechanism from transactional memory to manage runtime errors, but do not propose solutions that target energy savings ([17]-[20]). In addition, [11] proposes a HTM technique for error recovery from timing errors, but does not consider approximation by ignoring timing errors. By contrast, our goal is to provide a detailed HTM-based implementation for approximate execution that specifically targets energy savings.

III. Architecture

A. Target Platform

Our target architecture is based on *PULP* (Parallel Ultra Low Power platform), a scalable computing platform that leverages tightly-coupled shared-memory *clusters* as a main building block [13]. Each cluster features multiple cores (Processing Elements, PEs) that share a single instruction cache (*I$*). The cores do not have private data caches; they all share a multi-banked tightly coupled data memory (*TCDM*) configured as a shared data scratchpad. Each core is composed of the main data path that handles all fixed point instructions and a Floating Point Unit (FPU) that handles floating point operations. The main data path and the FPU are decoupled and have separate voltage islands, so their voltage can be adjusted independently. Each PE has a its own private FPU. Intra-cluster communication is based on a high bandwidth *low-latency interconnect*, implementing a word-level interleaving scheme to reduce access contention to the TCDM.

B. HTM Infrastructure

An underlying runtime system (RTS) transparently manages the transactions with a core-level policy that optimistically lowers the voltage in small steps. A snapshot of the system state is taken before each transaction is started so that if the allowed error threshold is exceeded during transaction execution, this safe state can be restored. For timing error detection, we assume that each core is equipped with runtime error-detection circuitry, such as error-detection sequential (EDS) [3]. We handle these timing errors at the granularity of a transaction. If the acceptable error threshold is exceeded while executing a transaction, the system aborts that transaction, takes the required countermeasures in terms of voltage settings, and restarts the transaction.

Our HTM design is based on two key components: (1) a check-pointing and rollback mechanism to recover from errors and retry aborted transactions when necessary and (2) a data versioning technique to keep track of original data versions to recover them when needed.

a) Checkpointing and Rollback: We enclose within a transaction each program block containing instructions to be monitored for timing errors, thereby protecting all sections of a program where voltage scaling will be applied, regardless of whether it will be approximated or not. At the beginning of each transaction, we save the core's internal state. If the error threshold is not exceeded, the transaction *commits*, the checkpoint is discarded, and speculative changes to the data become permanent. If the error threshold is exceeded, the transaction *aborts*, and a *rollback* mechanism restores the internal core state. In addition, data are restored to their original values and speculative copies are discarded.

b) Data Versioning: We use a distributed logging scheme to enable data versioning [11]. Logs are distributed among the TCDM banks and each bank keeps a fixed-size log space for each core in the system. Note that only the first time an address is written does its original value need to be saved in the log, so the log size is quite modest. The log saving and restoration process is done independently at each memory bank and does not require interaction with other banks, which makes it very fast and efficient.

IV. Implementation

Next, we describe *IgnoreTM*, an approximation-aware error management Dynamic Voltage Scaling (DVS) policy built on top of the above HTM infrastructure, which opportunistically ignores errors as a result of timing violations in order to save energy. Our approach works by allowing a certain rate of errors to be left uncorrected and aborting only when the acceptable error threshold has been reached.

For a DVS policy to be able to approximate instructions by ignoring timing errors, first it is necessary to consider which instructions can be approximated. We place instructions into two categories: *"approximatable"* and *"non-approximatable"*. For our experiments, we define approximatable instructions as floating point add, multiply, copy, negate (but not load/store, CPU to/from FPU register moves), and non-approximatable instructions as any other instructions that if approximated would result in significant loss in output quality, hence they must be kept accurate. In general, users may annotate approximatable instructions. We use HTM transaction boundaries over all parts of the program. The program regions that are not approximated must have HTM protection so that if a timing violation occurs there, the HTM support mechanism can correct it. The instructions that are annotated as approximatable and cause

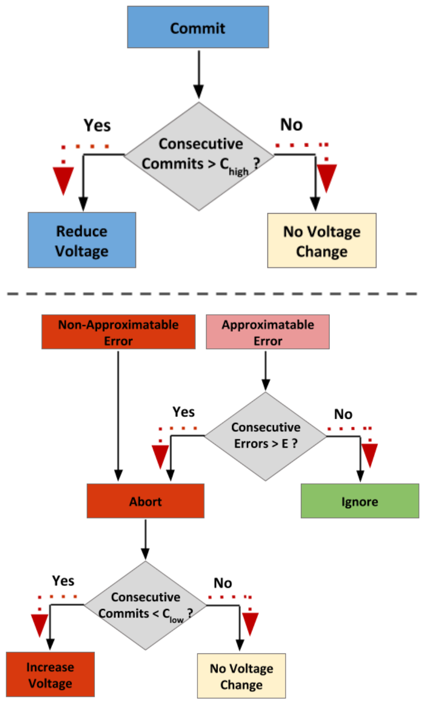

Fig. 1: Flow diagrams showing voltage adjustment policy on commit (top) and adjustment for timing violations (bottom).

timing violations are potentially ignored and left uncorrected. But, if the error threshold tolerance is exceeded, recovery and voltage level adjustment is possible.

Voltage adjustments are determined based on the number of completed transactions between aborts, i.e., the number of *consecutive commits*. In our *IgnoreTM* policy, we take into account the frequency of a timing violation during a transaction that has to be corrected, and its effect on the total system energy consumption (analyzed in Section V-B). When the frequency of a timing violation is too high (thus increasing the number of aborts and total energy consumption), we increase our operating voltage in order to save energy by lowering the abort rate. The *IgnoreTM* policy balances ignoring timing violations, correcting timing violations, and changing voltage levels to optimize energy savings while maintaining acceptable program output accuracy.

Fig. 1 shows the execution flow of our proposed policy. The supply voltage is gradually scaled down in steps until errors start to occur. A non-approximate policy would immediately abort the ongoing transaction where the error occurred and would re-execute it at a higher voltage, in order to correct it. Instead, *IgnoreTM* does not abort the ongoing transaction, but ignores the error if it has occurred within an approximatable instruction. A transaction is aborted only if i) the error occurred in a non-approximatable instruction, or ii) the error occurred in an approximatable instruction consecutively for a number of times exceeding a certain threshold. We call this the *error threshold* E, or the maximum number of ignored timing errors in a single run of a transaction.

The policy makes decisions on voltage adjustment on two occasions, i) when a transaction successfully commits, and ii) when a transaction aborts, before it is re-executed. A voltage adjustment decision is based on the number of consecutive commits that are experienced. We set two thresholds for consecutive commits, C_{high} and C_{low}, which are the decision factors for decreasing or increasing the voltage, respectively. The C_{high} threshold determines how many consecutive commits are necessary in order to safely scale down the operating voltage by one step. The C_{low} threshold determines how many consecutive commits must at least happen between two aborts. If the C_{low} threshold is not reached, then the policy realizes that the current operating voltage is dangerous and increases the voltage by one step. The choice of C_{high} and C_{low} threshold values is discussed in Section V-B.

The error threshold E (i.e., the maximum number of ignored timing errors in a single run of a transaction), determines the level of approximation the transaction will tolerate. Since every benchmark and every use case can have different error thresholds, we implement E as a user input: a parameter set by a specially encoded instruction. The user can set a threshold for the number of ignored timing errors *per each transaction*. While running each transaction, *IgnoreTM* will accumulate the number of ignored timing errors, and abort to retry when the threshold E is crossed. We assume that compilers never mark system calls as approximatable, so any timing violation in a system call will be caught and reverted via the HTM recovery mechanism explained below. The E threshold value choice is based on the target output quality and is discussed in detail in Section V-B.

V. Results

A. Experimental Setup

We use a cycle-level SystemC simulator of the PULP architecture described in Section III. To isolate the benefits of *IgnoreTM*, we run all experiments with one processor core since all cores operate independently from each other with regards to our policy, and as we will see, the total possible energy savings are heavily dependent on the benchmark we are running. We target the processor, including its paired Floating Point Unit (FPU), to be voltage-scaled. The main data path and the FPU in each processor are decoupled and have different voltage islands, hence their voltage can be scaled independently. This allows for more aggressive voltage scaling in the FPU, where approximations are applied. We use a voltage scaling step of 20 mV, which is in line with modern voltage regulators [23]. The dynamic and static power consumption of each module is characterized by extrapolation from an implementation of the platform in STMicroelectronics 28nm UTB FD-SOI technology under operating points considered in our experiments, and back-annotated in the simulator. Timing errors are generated on a cycle-basis, with exponentially increasing probability as voltage is scaled down. That is, for every step reduction in voltage (20mV), the error rate is increased by 10X. Specific behavior of FPUs under timing violations is implemented using the FPU error model proposed in [21]. Support for error correction is done through the HTM infrastructure (as described in Section III-B). As benchmarks, we consider floating point implementations of *Gaussian Filter* [4], *Fast Fourier Transform* (FFT) [22] and *Matrix Multiplication* [8]. We analyze their energy savings, runtime overhead, and quality of output loss from various DVS policies. For Gaussian Filter and Matrix Multiplication, we define quality of output as the Peak Signal to Noise Ratio (*PSNR*). For FFT, we define quality of output as the Average Relative Error (*ARE*).

To evaluate the capability of our approximation-aware *IgnoreTM* policy to save energy and improve performance, we compare it to two existing error-management policies that are also based on HTM but do not apply approximations and instead correct all occurring errors. These are the Point of First Failure (POFF) and Thrifty Uncle/Reckless Nephew (TURN) policies proposed in [11]. The POFF policy is a pessimistic approach where the supply voltage is gradually scaled down until the first failure occurs, in which case the voltage is immediately scaled up a step. Hence, with POFF any detected timing error results in the ongoing transaction being immediately aborted and re-executed after the voltage is increased. The *Thrifty Uncle-Reckless Nephew* or *TURN* policy optimistically scales the voltage beyond the point of first failure with the goal of achieving better energy savings. If a number of consecutive successful commits is reached, the voltage is scaled down further. However, if the number of aborts exceeds a certain threshold, the voltage is increased again to avoid further errors and wasted energy due to excessive re-execution.

B. Tuning Error Threshold and Commit Parameters

Here we discuss how we tune the Error Threshold E and the Commit Parameters, C_{high} and C_{low} that were introduced in Section IV. We set C_{high} to 100 and C_{low} to 10. This choice was based on an empirical analysis in which this setup yielded the best energy savings. As discussed in Section IV, E is a user-determined threshold of how many timing violations and approximations each transaction can tolerate. We performed a set of experiments for all benchmarks to characterize how E correlates with the output accuracy in each case and extract the value of E that yields this level of accuracy. The accuracy ranges and the average E values to which they corresponded are summarized in Table I. Depending on the application, different levels of accuracy might be acceptable. E_{low}, E_{medium} and E_{high} correspond to low, medium and high level of accuracy respectively. We next use these values in our experiments to see how *IgnoreTM* performs in terms of energy and runtime compared to POFF and TURN.

C. Overhead, Runtime and Energy Consumption

The area overhead of the error detection and the HTM-based error correction hardware is small. The EDS 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 bulk of the hardware overhead is therefore due to the data versioning scheme (i.e., logging), which accounts for a 2.65% area overhead. The runtime overhead introduced by HTM is also very small (1.1% to 1.8% relative to no voltage scaling).

We carried out a set of experiments to see how IgnoreTM performs in terms of energy consumption and runtime compared to POFF and TURN policies for different levels of target accuracy. In Figure 2 we show the runtime and energy consumption results for our simulations, *all compared to the more conservative POFF policy*. Specifically, in Figure 2(a) we see that with IgnoreTM the runtime improves over the TURN policy for all benchmarks and all levels of accuracy. We observe on average 7% improvement in runtime for Gaussian and FFT and 4% for Matrix Multiplication. This improvement

Gaussian Filter	E_{low} = 1 , PSNR = 33.6 dB	E_{medium} = 5 , PSNR = 23.4 dB	E_{high} = 255 , PSNR = 8.7 dB
FFT	E_{low} = 1 , ARE = 0.088 %	E_{medium} = 5 , ARE = 0.77 %	E_{high} = 25 , ARE = 9.53 %
Matrix Multiplication	E_{low} = 1 , PSNR = 75.6 dB	E_{medium} = 10 , PSNR = 63.7 dB	E_{high} = 25 , PSNR = 59.4 dB

TABLE I: Error threshold E and output accuracy correlation.

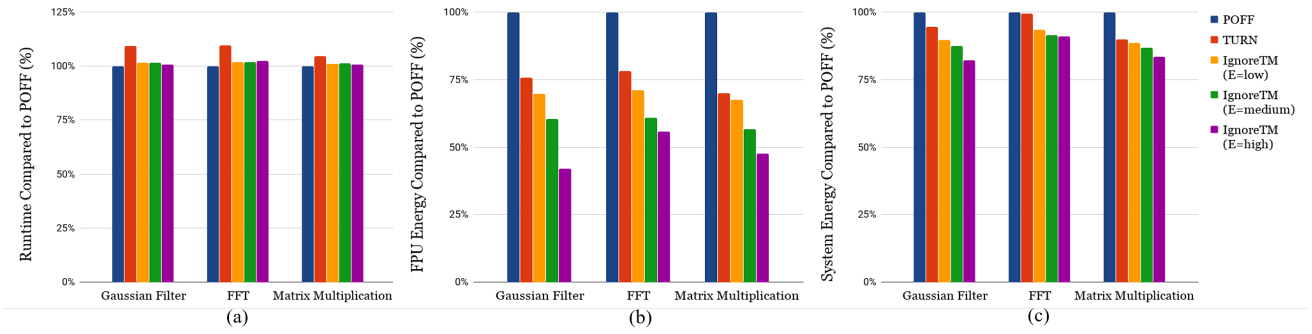

Fig. 2: (a) Runtime, (b) FPU and (c) System Energy Consumption for different DVS policies (results shown relative to POFF).

is due to our new policy of opportunistically ignoring timing errors in approximate regions while for TURN errors are always being corrected, thus wasting time for transaction recovery and re-execution. The POFF policy shows slightly better runtime than IgnoreTM since it operates at a safer voltage level just above the edge of failure, hence it does not encounter timing errors; however the difference is negligible (1-2%). Also, while not shown in the figure, note that our IgnoreTM policy has negligible impact on runtime compared to a baseline configuration that never applies voltage scaling (and thereby never encounters timing errors).

Figure 2(b) shows the energy consumption of the FPU for TURN, POFF and IgnoreTM policies. The FPU is where approximations are applied in IgnoreTM, so this is where we expect the most significant energy savings. Indeed, we observe that compared to POFF, IgnoreTM improves energy consumption by 30%-58% for Gaussian, 29%-44% for FFT and 33%-52% for Matrix Multiplication. The energy savings of IgnoreTM over TURN range between 8%-45% for Gaussian, 9%-29% for FFT and 4%-32% for Matrix Multiplication. Comparing IgnoreTM with a baseline policy that never applies voltage scaling, the respective savings are 66%-80% for Gaussian, 65%-72% for FFT and 71%-80% for Matrix Multiplication (not shown in the figure). As expected, IgnoreTM achieves significant energy savings where approximations are applied and these savings depend heavily on the target accuracy level. We get better energy savings as we increase the error threshold E from low to high, hence decreasing the level of accuracy. The overall system energy, including system components that are not voltage scaled, follows similar trends (Figure 2(c)). As expected, when looking into the total system energy that includes components that operate at 100% accuracy, the energy savings diminish compared to the FPU savings; however, they are still significant. IgnoreTM yields energy savings up to 18% over POFF and up to 13% over TURN. Compared to a baseline policy which does not apply voltage scaling, IgnoreTM yields system energy savings up to 47%.

VI. Conclusions

In this paper, we propose *IgnoreTM*, an adaptive error management scheme that allows bounded approximate computation through aggressive voltage scaling, using a HTM-based infrastructure to recover from errors when necessary. Our results show that *IgnoreTM* can yield up to 47% system energy savings while incurring only a 1% overhead in runtime compared to no voltage scaling. Moreover, it yields up to 18% additional system energy savings compared to other prominent voltage scaling policies that exist in literature due to its ability to opportunistically ignore some timing errors.

References

[1] W. Baek and T. Chilimbi. Green: A Framework for Supporting Energy-conscious Programming Using Controlled Approximation. *SIGPLAN Not.* 2010.
[2] S. Borkar, et al., Parameter variations and impact on circuits and microarchitecture. *DAC* 2003.
[3] K. Bowman, et al., A 45nm resilient microprocessor core for dynamic variation tolerance. *JSSC* 2011.
[4] G. Bradski, et al., The OpenCV Library. *Dr. Dobb's Journal of Software Tools* 2000.
[5] S. Dighe, et al., Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *JSSC* 2011.
[6] H. Esmaeilzadeh, et al., Architecture Support for Disciplined Approximate Programming. *SIGPLAN Not.* 2012.
[7] M. Herlihy and J. E. B. Moss. Transactional Memory: Architectural support for lock-free data structures. *ISCA* 1993.
[8] S. Hong, et al., Eigenbench: A simple exploration tool for orthogonal TM characteristics. *IISWC* 2010.
[9] P. K. Krause, et al., Adaptive voltage over-scaling for resilient applications. *DATE* 2011.
[10] L. Leem, et al., ERSA: Error resilient system architecture for probabilistic applications. In *DATE*, March 2010.
[11] D. Papagiannopoulou, et al., Edge-TM: Exploiting Transactional Memory for Error Tolerance and Energy Efficiency. *TECS* 2017.
[12] A. Rahimi, et al., Spatial Memoization: Concurrent Instruction Reuse to Correct Timing Errors in SIMD Architectures. *TCAS-II* 2013.
[13] D. Rossi, et al., PULP: A parallel ultra low power platform for next generation IoT applications. *HCS* 2015.
[14] M. Samadi, et al., PARAPROX: Pattern-based Approximation for Data Parallel Applications. *ASPLOS* 2014.
[15] A. Sampson, et al., EnerJ: Approximate Data Types for Safe and General Low-power Computation. *PLDI* 2011.
[16] G. V. Varatkar, et al., Error-Resilient Motion Estimation Architecture. *TVLSI* 2008.
[17] H. Q. Le, et al., Transactional Memory support in the IBM POWER8 processor. *IBM Journal of Research and Development* 2015.
[18] J.-T. Wamhoff, et al., Transactional encoding for tolerating transient hardware errors. *SSS*, volume 8255 of *LNCS* 2013.
[19] G. Yalcin, et al., Fault tolerance for multi-threaded applications by leveraging hardware transactional memory. *Computing Frontiers* 2013.
[20] G. Yalcin, et al., Combining error detection and transactional memory for energy-efficient computing below safe operation margins. *PDP* 2014.
[21] S. Whang, et al., Evaluating Critical Bits in Arithmetic Operations due to Timing Violations. *HPEC* 2017.
[22] A. Yazdanbakhsh, et al., AxBench: A Multiplatform Benchmark Suite for Approximate Computing. *IEEE Design & Test* 2017.
[23] Intel Corporation, Voltage Regulator Module and Enterprise Voltage Regulator-Down 11.1. http://www.intel.com/Assets/en_US/PDF/design-guide/321736.pdf 2009.