Center for Embedded and Cyber-physical Systems
University of California, Irvine

Relaxing Manufacturing Guard-bands in Memories for Energy Saving

Majid Shoushtari, Abbas Banaiyan, Nikil Dutt

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-2620, USA

{anamakis,abanaiya,dutt}@uci.edu

CECS Technical Report 14-04
May 2, 2014
(Revised August 18. 2014)

Relaxed Cache: Relaxing Manufacturing Guard-bands in Memories for Energy Saving

Majid Shoushtari, Abbas Banaiyan, Nikil Dutt
Dept. of Computer Science, UC Irvine, Irvine, CA 92697-3435
{anamakis,abanaiya,dutt}@ics.uci.edu

ABSTRACT

Memories occupy a significant area of chip real estate and are major contributors to total system energy consumption. Furthermore, the guard-banding required for masking the effects of ever-increasing manufacturing variability in memories imposes significant power overhead. In this Technical Report, we explore the opportunity for exploiting the intrinsic tolerance of a vast class of applications to some level of error in order to relax this guard-banding and build partially-forgetful memories that may lose some data. We introduce *Relaxed Cache* as an exemplar of this partially-forgetful memories. Unlike classic approaches which incorporate some type of uniform static guard-banding (e.g., higher voltage and/or sophisticated error correcting codes) for the entire cache at manufacturing-time, we elevate the decision-making to application-level by allowing the software programmer to dynamically adjust the required level of guard-banding at run-time. The programmer can relax the guard-bands for majority of the cache ways using two controlling knobs: (1) number of acceptable faulty bits in a cache block (AFB), and (2) supply voltage for SRAM array (VDD). The programmer also tags non-critical regions of the application's virtual address space to be potentially mapped to relaxed ways. The data tagging along with knobs adjustment enable the programmer to guide the system to dynamically alternate between different settings during run-time, seeking the optimal energy-performance-fidelity point in that specific phase of execution. Experimental results for sample error-tolerant benchmarks show up to 74% energy saving w.r.t. a uniformly guard-banded cache.

1. INTRODUCTION

As the semiconductor industry continues to push the limits of sub-micron technology, the ITRS expects hardware variations to continue increasing over the next decade [28]. The memory subsystem is one of the largest components in today's computing systems, a main contributor to the overall power consumption of the system, and therefore one of the most vulnerable components to the effects of variations. Device manufacturers have partially masked the presence of variability by guard-banding mechanisms.

1.1 Guard-banding Overhead

Guard-banding leads to over-design with less than optimal power and/or lifetime for different memory technologies. [31] shows that 10^3 leaky cells in a 32GB DRAM module force the use of 64ms refresh interval while other cells can work with a refresh interval that is four times longer. [46] also reports a 96% increase in the PCM programming power and 50X endurance degradation due to process variation.

In SRAM-based on-chip memories, the desire to operate at low voltages necessitates the need for guard-banding because of significant threshold voltage variations in those regimes. This guard-banding is usually a combination of using (1) higher supply voltage levels, (2) larger transistors, (3) complex logic for error detection/correction, and (4) spare cells. We focus here more on the first technique. Findings of [43] show that masking all variabilities in 90nm SRAM requires an increase in data retention voltage from nominal value of 0.35V to 0.7V. [37] shows that as feature size becomes smaller, more guard-band is required. While 20% voltage guard-band can improve error rate in 45nm by an order of magnitude, it can improve error rate of 16nm SRAM cell only by 3X.

There is a large body of work on fault-tolerant voltage-scalable SRAM cache designs that attempt to use the best combination of aforementioned guard-banding techniques which minimize power overhead while satisfying a yield threshold [3, 44, 45, 2, 8]. All of these approaches are application-agnostic and do not adapt the guard-banding to the requirements of application.

1.2 Approximate Computing

Recently, researchers have begun to explore energy-accuracy trade-offs in general-purpose applications [39, 29, 19]. A key observation is that systems spend a significant amount of energy for guaranteeing correctness. Consequently, a system can save energy by exposing faults to a variety of applications that are resilient to hardware and software errors during their execution [20]. In particular, embedded, multimedia and Recognition, Mining and Synthesis (RMS) applications could still process information usefully with unreliable or error-prone elements [16]. This motivates the opportunity to build partially-forgetful memories by doing application-aware adaptive guard-banding (i.e., relaxing the guard-bands when possible and strict guard-banding when necessary).

1.3 Challenges for Adaptive Guard-banding in Memories

In order to utilize adaptive guard-banding, some challenges should be addressed. Here we present the steps that we have identified should be taken in order to effectively employ

partially-forgetful memories.

1.3.1 Data Partitioning

Even in error-tolerant applications, there are certain data objects that are critical for the application's functionality. Critical data structures are defined as parts of the application's virtual address space in which any corruption leads to catastrophic failure or has a significant impact on the quality of output for that application. All other data are considered to be non-critical. This concept has been used before in [32]. A simplistic partitioning, as in [32], partitions the data only in two categories of critical and non-critical. But the non-critical part can be further sub-categorized for better exploitation of different regions of forgetful memory. For example: (1) integer data can be partitioned based on the magnitude of tolerable error; (2) floating point data can be partitioned based on a limit on precision. These categorizations need to be reflected in the program by annotating different data structures. One approach [39, 14] defines type qualifiers and dedicated assembly-level store and load instructions for this purpose. The other approach uses dynamic declarations that are enforced by a run-time system. We will show an example of this approach later in Section 3.

1.3.2 Data Mapping

The data partitioning annotations in the previous step should be used to map each category of data an appropriate part of memory based on its reliability characteristics. Depending on the type of memory, this mapping can be done by the hardware, application/compiler, or operating system. For caches, the cache controller decides where to map a new incoming cache block. In case of software-controlled memories, the compiler aggregates data with the same reliability requirement in tagged groups, and then a run-time system (having knowledge about underlying hardware) does the actual mapping based on this tagging. For main memories, the operating system should do the mapping during logical-to-physical mapping.

1.3.3 Controlling Exposed Error Rate to Program

Previous attempts at utilizing relaxed-reliability memories [40, 32] lack of the ability to dynamically adjust adjust the exposed error-rate to the program. We believe this is necessary because different applications have different level of tolerance to errors. Even within an application, one execution phase may have more tolerance to errors (i.e., less criticality) than another phase. Therefore, it is important to have the ability to adjust the guard-banding knobs based on the characteristics of application. Examples of these knobs are: DRAM Refresh Rate, SRAM Voltage, STT-RAM Retention Time, I-RESET Current in PCM, etc. If it is the compiler or operating system who adjusts these knobs, then they can work with the actual values for each knob. But these knobs could be also abstracted out for a novice programmer. An example of this abstraction will be shown in Section 3.

1.3.4 Adaptation to Phasic Behavior of Applications

An application can have phasic behavior in its computation because it alternates between memory and compute-bound phases [21]. Similarly, it has phasic behavior in criticality of its processing if it has *critical* and *non-critical* regions of code which have different impacts on the fidelity of the application's output [15]. Accordingly, there are three main reasons to change memory guard-banding knobs during run-time: 1) Depending on the application or set of applications that are executed, we might be able to tolerate different levels of error; 2) We may need to change the ratio of reliable to unreliable memory capacity to prevent performance degradations for specific situations (e.g., where most of the working set objects should be mapped to reliable parts of memory); 3) The programmer can save power by disabling parts of memory for computation-bound application phases (this works even for applications that do not tolerate any errors). On the other hand, even for applications that can not tolerate any error, the programmer by adjusting knobs to appropriate values can disable part of memory for computation-bound phases of application and hence save power.

1.4 Contributions

This paper proposes *Relaxed Cache* which exploits the application's behavior for adaptive relaxation of guard-bands in cache memories to save energy. Unlike previous efforts on memories for approximate computing [40, 32], here this relaxation is done in a *disciplined* manner. To the best of our knowledge, this is the first work that explicitly allows the programmer to have control on the guard-banding in underlying memory. Using language extensions, Relaxed Cache provides two knobs to the software programmer, which then (s)he uses for controlling the amount of guard-banding during different phases of execution. These knobs are: (1) SRAM array voltage (VDD), and (2) number of Acceptable Faulty Bits in a cache block (AFB). The application developer also guides a run-time system to dynamically tag the data blocks according to their criticality. Later, based on this tagging, the cache controller knows if a piece of data is critical and should be protected, or if relaxed caching is acceptable and accordingly performs the block replacement. The knob adjustment along with this data tagging enable the programmer to guide the system to dynamically alternate between different cache operational points during run-time seeking the optimal point in each phase of execution. This optimality can be defined based on various metrics including energy, performance, output fidelity or a combination thereof.

Accordingly, this paper makes the following contributions:

1. We elevate the decision making about memory hardware guard-bands to the application-level.
2. We define knobs for *disciplined* relaxation of cache guard-banding.
3. We describe how a software programmer, having knowledge about application characteristics and semantics, can adjust these knobs for saving power.
4. We provide means to the software programmer for *dynamically* declaring and undeclaring regions of application's virtual address space as candidates for relaxed caching and a run-time system that enforces this criticality declarations.

The rest of this report is organized as follows. Section 2

PSNR	inf	37 dB	32 dB	28 dB	26 dB
VDD	600 mV	560 mV	520 mV	480 mV	440 mV
AFB	0	Very Low (1)	Low (2)	Mild (3)	High (4)
Norm. Leakage Energy	0.78	0.71	0.64	0.58	0.53
Norm. Exec. Time	1.00008	1.0001	1.0001	1.0003	1.03

Figure 1: Exploring Performance-Energy-Fidelity Space by Adjusting Relaxed Cache Controlling Knobs (Leakage Energies and Execution Times are Normalized to a Baseline that Uses 700mV for SRAM Array Supply Voltage)

motivates the need for Relaxed Cache. Section 3 presents details of Relaxed Cache and its hardware and software components. Section 4 shows experimental results and their related discussions. Section 5 surveys related work and differentiates our work. Section 6 concludes this paper and gives directions for future work.

2. MOTIVATION FOR RELAXED CACHE

As mentioned earlier, Relaxed Cache provides controlling knobs to the programmer which (s)he can play with to save energy. Here, using the Susan/Image-Smoothing benchmark from MiBench [26], we show how these knobs can be used to save leakage energy in the underlying cache with minimal impact on the performance and output fidelity of the application. This benchmark is commonly used for smoothing images in order to remove specks of dust and artifacts from scanning. Figure 1 shows the output of smoothing the Lena test image with different levels of AFB and VDD. Figure 1 also shows normalized leakage energy and execution time w.r.t. the baseline system that uses a cache with 700mV supply voltage. This example shows that depending on the acceptable quality for the output of this benchmark, the programmer can achieve up to 47% leakage energy saving by adjusting controlling knobs at the application-level.

Before detailing the Relaxed Cache approach, we present the rationale behind the performance-energy-fidelity tradeoff offered by adjusting controlling knobs in caches. To increase memory density, memory bit-cells are typically scaled to reduce their area. High density SRAM bit-cells use the smallest devices in a technology, making SRAMs more vulnerable to manufacturing variations. On the other hand, static power, dominated by sub-threshold leakage current, is the primary contributor of power in memories and has an exponential dependence on supply voltage [24]. Reducing supply voltage for saving power makes the SRAM even more vulnerable to variations, especially in 65nm and below, which results in an exponential increase in the probability of cell failure [42].

Figure 2 shows that the capacity of a cache with traditional

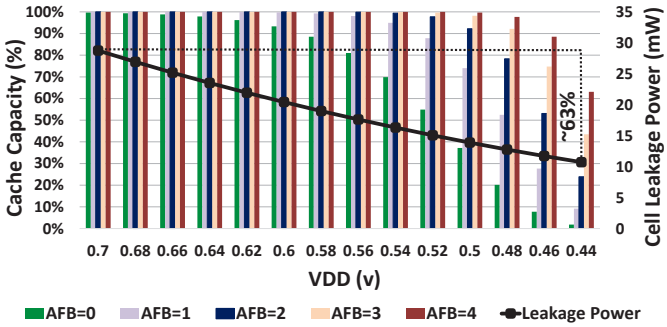

Figure 2: Tradeoff Between Cache Capacity, VDD, AFB and Bit-cell Leakage Power (Cache Block Size=64-byte, Technology=45nm).

assumption of 100% fault-free behavior (i.e., AFB=0), drops exponentially as we lower the voltage, which can dramatically degrade the performance of execution due to too many misses. This prevents the cache manufacturers to set the Vdd-min below a certain threshold. But if we consciously allow the SRAM to violate data integrity by increasing AFB to a value more than zero (as we did in the example shown in Figure 1), we can effectively lower this threshold depending on the level of tolerable errors in the application. Therefore, it's advantageous to let the application determine the (VDD, AFB) operational points of the cache. Even when no error is tolerable, the programmer can exploit the application's phasic behavior, e.g. lowering the cache voltage for an application's phase that will not require a large cache capacity. Note that moving from a voltage level to a lower voltage, even with same AFB level, helps the system to save energy.

3. RELAXED CACHE

The observations reported in Section 2, clearly makes it appealing to rely on the software programmer's ability to control the cache operational settings, and therefore letting him/her to explicitly specify the trade-off between energy consumption, output fidelity, and performance. Relaxed Cache requires the software programmer to identify different computational and criticality phases of the application. We believe it is straightforward for the software programmer,

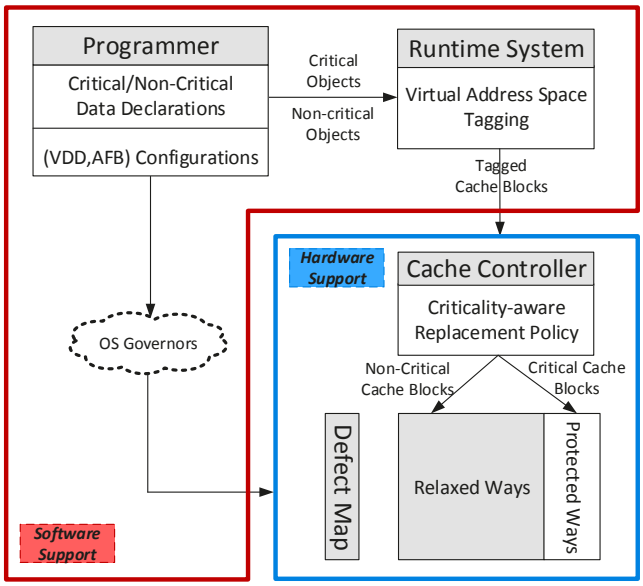

Figure 3: High-level Diagram Showing HW/SW Components of Relaxed Cache and their Interactions.

knowing the application semantics, to make the distinction between critical and non-critical parts of the application. Furthermore, using phase detection techniques [22], the programmer can determine different computational phases in the program. Relaxed Cache is a hardware/software collaborative approach that enables the programmer to exploit application semantics and domain knowledge toward saving energy.

Figure 3 shows the high-level overview of different components involved in the design of Relaxed Cache scheme and their interactions with each other. Our scheme requires some small modifications in both traditional hardware and software components of a system, as described in the following subsections detailing each shaded component of Figure 3.

3.1 Hardware Support

Here, we present the hardware support for Relaxed Cache that requires the following minor changes to traditional caches (shaded components in the hardware blocks of Figure 3):

1. Tuning *Relaxed Ways* based on architectural knob settings (AFB and VDD).
2. Modifying the *Defect Map* to store special defect information required by Relaxed Cache.
3. Making the *Cache Controller* aware of critical vs. non-critical blocks.

We describe each in more detail below.

3.1.1 Architectural Knobs for Tuning Relaxed Ways

We consider a SRAM cache that has N ways. A fixed small number (e.g., $\log N$) of these ways are protected, either by using a fixed high voltage that assures their fault-free storage or through the use of complex error correction techniques. These ways are called *Protected Ways*. The remaining cache ways in a cache set work with a lower dynamically adjustable

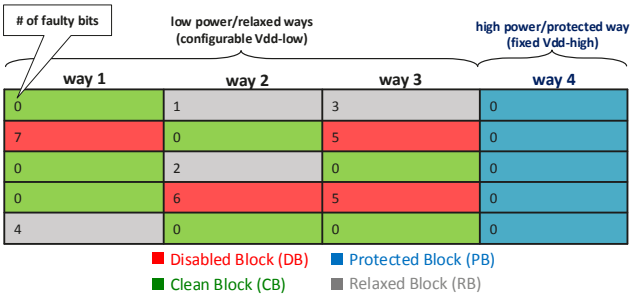

Figure 4: A Sample 4-way Cache with VDD=580mV and AFB=4.

voltage that could result in some faulty bits. We refer to these ways as *Relaxed Ways*. The operation of SRAM array for relaxed ways is controlled by two architectural knobs: (1) Supply voltage (VDD), and (2) Number of acceptable faulty bits (AFB) per each cache block. The definition of AFB allows us to relax the guard-bands for majority of cache ways while controlling the level of errors that are exposed to the program. According to this definition, we can have four types of cache blocks for each (VDD, AFB) setting:

- **Protected Block (PB)**: All blocks in protected ways.
- **Clean Block (CB)**: All fault-free blocks in relaxed ways.
- **Relaxed Block (RB)**: All blocks in relaxed ways that have at least one but no more than AFB number of faults.
- **Disabled Block (DB)**: All blocks in relaxed ways that have more than AFB number of faults.

Figure 4 demonstrates these terminologies using a sample 4-way cache when AFB=4.

3.1.2 Defect Map Generation and Storage

In order to control the level of errors that are exposed to the application in each (VDD, AFB) pair, we need to disable those blocks that have more than AFB number of faults in each VDD. For that we require to know how many cells fail at each VDD. Previous FTVS approaches have used defect maps for this purpose [8, 41]. To populate the cache defect maps, we can use any standard Built-In Self-Test (BIST) routine that can detect memory faults (e.g., March Tests [27]). If BIST is done once at test-time, then a non-volatile (NV) storage will be needed to store the defect map for the duration of the device lifetime. If it is done at every power-up of the chip, then no extra NV storage is necessary.

The defect map bookkeeping requires a few additional bits in the tag array of each cache block. However, if the number of AFB and VDD levels are limited, this overhead is negligible. For instance, assume that we have four levels of AFB and three levels of VDD. We can show that in this case the defect map overhead is about 1.5% for a cache with block size of 64-byte. Assume that our VDD levels are V0 > V1 > V2. Similarly we have A0 < A1 < A2 < A3 for AFB levels. Figure 5a shows the result of running a march test on four blocks of a set in a 4-way cache for three different VDD levels. The digits inside blocks represent number of faulty bits for each block at that VDD. Note that blue blocks belong to a protected way, therefore they are fault-free for

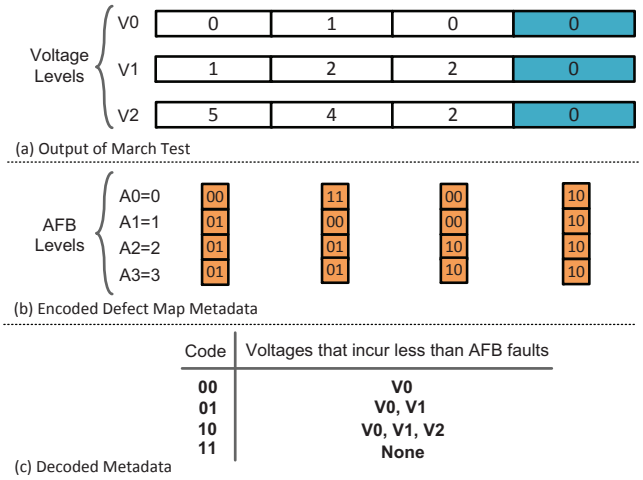

Figure 5: Encoding and Decoding Defect Map Info in Relaxed Cache

all VDDs. The fault inclusion property [25] states that the bits that fail at some supply voltage level will also fail at all lower voltages. Therefore by decreasing the voltage, number of faulty bits increases monotonically. We use this property and for each AFB=Ai encode the lowest voltage level that results in Ai number of faults or less (Figure 5b). In this manner, for each 64-byte block we use 8 bits to capture defect status of that block in all available (VDD, AFB) pairs, resulting in a 1.5% area overhead for a traditional cache.

3.1.3 Cache Controller

The last modification in the cache hardware needs to be made in the cache controller. Relaxed Cache's replacement policy should discriminate between critical and non-critical blocks. As stated before, there are a *fixed* number of relaxed ways. Whenever a miss occurs and the missed data block is tagged as a non-critical block (meaning it contains only non-critical data), the replacement policy should select a victim block from the relaxed ways of the corresponding cache set. On the other hand, a critical block is always allocated in a protected way. Note that for very aggressively-scaled voltage levels, it's possible that all of the blocks in the relaxed ways become disabled based on the specified AFB. In this case, the replacement policy uses a block in one of the protected ways for bringing a non-critical block into the cache. This allows us to operate at very low voltage levels and trade cache capacity for power saving for applications that are not memory-bound in certain phases of their execution. Because block replacement logic is not in the critical path for hit read/write accesses, this minor modification does not affect cache access latency.

3.2 Software Support

Here, we describe changes to software components in Figure 3 to support Relaxed Cache by programmer.

3.2.1 Programmer-Driven Application Modifications

Data Criticality Declaration: In order to have the program relax the guard-bands for part of the cache and simultaneously utilize that relaxed part, the software programmer should be able to identify non-critical data structures in the program. These data structures will be candidates for relaxed caching. Critical data structures are defined as parts of the application's virtual address space in which any corruption leads to catastrophic failure. They also include any data that has a significant impact on the output of the application. All other data are considered to be non-critical. We believe a software programmer can easily make this distinction since (s)he is aware of the functionality and semantics of different parts of the application and related data structures.

The memory footprint of an application has four segments: Code, Global, Stack, and Heap. The code segment usually can not tolerate any errors. Global, stack and heap can contain both critical and non-critical data. In our experiments we found most of the non-critical data to be allocated on heap. However, our implementation supports non-critical data in all memory segments.

Type qualifiers have been used before for data criticality annotation in [32]. That approach makes a data region to be statically declared as either critical or non-critical for the *entire* execution time. Instead in Relaxed Cache, the programmer uses `DECLARE_APPROXIMATE(Start-VA, End-VA)` and `UNDECLARE_APPROXIMATE(Start-VA, End-VA)` to declare and undeclare data non-criticality dynamically within code. `Start-VA` and `End-VA` are the virtual addresses of the boundaries of the region declared/undeclared to be non-critical. The additional capability of undeclaring a data region becomes important in two cases: (1) It is common in the area of approximate computing that the programmer utilizes understanding of functionality and semantics of the program to bound the errors introduced to certain data structures before passing them to the next stage of program. The purpose is to reduce the magnitude of error that is propagated between different stages. In such a case it's usually desirable to stop introducing any further errors in data after error bounding. (2) When a critical procedure should be done on a piece of non-critical data, it's desirable to undeclare that region of data. An example of this case is at the end of image compression, when the checksum of pixels should be calculated and stored in the file.

Cache Configuration: The intuition behind letting the programmer configure the cache settings, is a key insight in today's energy management techniques for systems and applications: *they respond to phases*. Accordingly, our scheme leverages the programmer's knowledge for managing energy consumption. The programmer, with in-depth knowledge of the software is in the best position to identify different phases of application and based on that manages the settings of the underlying hardware. On the other hand, a programmer can respond to the system's operational mode by devising provisions that adapt the execution to these modes [18]. For example, a programmer can decide to render a high-fidelity image when the user's smart-phone is fully charged, and only a low-fidelity image when the battery level of the phone falls below 25%.

Depending on the software programmer's familiarity with underlying hardware, the hardware controlling knobs can be abstracted. For example, an embedded system programmer,

with fairly good understanding of hardware details, can explicitly set Relaxed Cache's knobs based on their low-level meanings to fully exploit the Relaxed Cache capabilities. However, for a traditional software developer (with little knowledge of hardware), we can abstract the hardware knobs settings into discritized "levels" allowing the programmer to qualitatively set hardware knobs based on their desired effects (e.g., low power setting, or high-fidelity setting). Figure 6 shows such an abstraction for Relaxed Cache knobs.

Nevertheless, to utilize the capabilities offered by Relaxed Cache, the software programmer should dynamically control VDD and AFB knobs to adjust the guard-banding to the current phase of execution. Language extensions let the programmer use `CONFIG_CACHE(VDD-LEVEL, AFB-LEVEL)` for configuring the cache's operational points. Note that this incurs several penalty cycles for flushing the cache. Therefore, this reconfiguration should be done infrequently, i.e., only in certain important stages of each application. The request for this reconfiguration should be sent to the operating system to be handled by its governor which has the privilege to change the hardware settings.

Figure 7 shows a piece of code in an image editing program that uses declarations and cache configuration to save leakage energy during scaling up an image. In this example, two large regions of data that can tolerate some level of errors are the data structure that holds values of original image (pointed to by `src`), and the data structure for storing up-scaled image (pointed to by `dest`). As can be seen, the programmer has declared both regions as candidates for relaxed caching. After returning back from `scale()` function, and before moving to the next phase of the program, (s)he has undeclared both regions. Source data structure is undeclared because it is not required anymore and is going to be freed. The data structure holding the up-scaled image has been also undeclared because the next phase of program may not tolerate any error during processing image.

3.2.2 *Run-time System*

We assume a run-time system (e.g., the operating system), uses the programer's declarations to keep a table of addresses that contain non-critical data per each application updated. Once a miss happens, while the data is fetched from the lower level of memory, the run-time system searches this table for the address of the missed block. If the address is already within the table, the block is tagged as non-critical. Otherwise, the block is tagged as critical. As mentioned before, this tagging is used by cache controller's replacement mechanism to find an appropriate cache way for replacement.

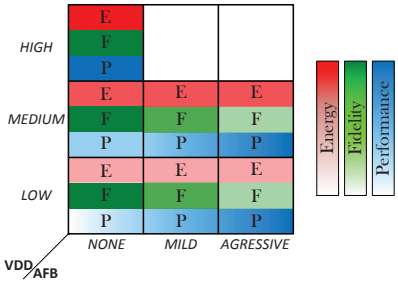

Figure 6: Abstracting Relaxed Cache Knobs up to Metrics Familiar to a Software Programmer (i.e., Performance, Fidelity, and Energy Consumption). Note that (VDD = High, AFB = Mild) and (VDD = High, AFB = Aggressive) Combinations are Sub-optimal, Hence not Applicable.

```
#include "Approximations.h" // Enables approximation declarations

#define INT_SIZE sizeof(int);

int main(int argc, char**argv){

// Capturing Raw Image on Camera
// Compress Raw Image

/*------------------------- Image Scaling -------------------------*/
CONFIG_CACHE(MEDIUM VDD, MILD AFB);

int *src, *dest;
dest = (int*) malloc(*num_elmnts*INT_SIZE);

#ifdef SRC
  DECLARE_APPROXIMATE((uint64_t)&src[0],
                      (uint64_t)(&src[*num_elmnts]+INT_SIZE-1));
#endif

#ifdef DEST
  DECLARE_APPROXIMATE((uint64_t)&dest[0],
                      (uint64_t)(&dest[*num_elmnts]+INT_SIZE-1));
#endif

src = read_image(in_filename,&sw,&sh,&src_dim);
dest = allocate_transform_image(scale_factor,sw,sh,&dw,&dh,&dest_dim);

scale(scale_factor, src, sw, sh, dest, dw, dh);

#ifdef SRC
  UNDECLARE_APPROXIMATE((uint64_t)&src[0],
                        (uint64_t)(&src[src_dim]+INT_SIZE-1));
#endif

#ifdef DEST
  UNDECLARE_APPROXIMATE((uint64_t)&dest[0],
                        (uint64_t)(&dest[dest_dim]+INT_SIZE-1));
#endif

free((void *) src);
/*-----------------------------------------------------------------*/

// Other Image Transformations/Editing

}
```

Figure 7: A Code Sample Showing Programmer's Data Criticality Declarations and Cache Configurations to be Used by Relaxed Cache.

4. EXPERIMENTAL EVALUATION

4.1 Experimental Setup

To simulate the behavior of Relaxed Cache, we modified the cache architecture in the gem5 framework [12] to implement our scheme in detail. We used gem5's pseudo-instruction capability for implementing required language extensions in order to enable the software to interact with the rest of system. The common gem5 parameters used in all our simulations are summarized in Table 1. Note that we have used *Random* replacement policy which is commonly used in embedded domain processors [4, 6, 5] due to its minimal hardware and power cost.

Table 1: gem5 Common Parameter Settings

Parameter	Value
ISA	Alpha
CPU Model	Detailed (OoO)
No. Cores	1
Simulation Mode	Syscall Emul.
Cache Config	L1 (Split), L2
Replacement Policy	Random
Cache Block Size	64 B
L1$ Size, Assoc.	32KB, 4-way
L2$ Size, Assoc.	256KB, 8-way
Main Memory	LPDDR3, 512MB

Our SRAM bit error rates (BER) are shown in Figure 8. These were computed by [25] using the data and models from [42], which performed a detailed analysis of SRAM noise margins and error rates for a commercial 45nm technology. Using these BERs, we found the distribution for number

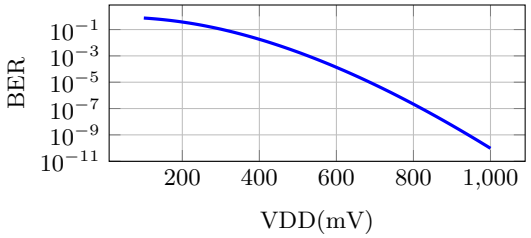

Figure 8: SRAM Bit Error Rates (BER) for 45nm Using Models and Data from [42]

of faults for each data array voltage, thus allowing us to compute the expected cache capacity in each AFB.

Cache power consumption is estimated based on the model in [25]. CACTI 6.5 [35] is modified to extract static power for the baseline and Relaxed Cache. Note that baseline cache doesn't have any fault tolerance and doesn't support voltage scalability. NFET and PFET on/off current parameters in CACTI were set directly from SPICE data using commercial 45nm SOI MOSFET models.

4.2 Benchmarks

We selected a number of RMS benchmarks to examine the energy savings that Relaxed Cache enables us to obtain. Below are descriptions of each benchmark and their fidelity metrics.

4.2.1 Susan

Susan is an image recognition package from MiBench [26]. It was developed for recognizing corners and edges in Magnetic Resonance Images of the brain. It is typical of a real world program that would be employed for a vision based quality assurance application.

Image-Smoothing: Part of the Susan package can smooth an image and has adjustments for threshold, brightness, and spatial control. For Image-Smoothing, we use Peak-Signal-to-Noise Ratio (PSNR), which is a common quality metric in the image processing domain. Table 2 shows the relation between PSNR and perceptual quality.

Table 2: Relation Between PSNR and Perceptual Quality in Image Processing Domain

PSNR	Perceptual Quality
< 28.0	Low
$28.0 \leq \cdots < 30.0$	Acceptable
$30.0 \leq \cdots < 33.0$	Good
≥ 33.0	Excellent

Edge-Detection: Susan package can also perform edge detection which is one of the most commonly used operations in image analysis. Smart-phone apps use edge detection and perspective correction to clean up digital shots of printed photos. [33] defines Accuracy metric as the fraction of detections that are true positives rather than false positives. A value of 0.8 or more is usually considered acceptable. Even though there are also some other methods to quantitatively measure the visual quality of edge detectors, none are widely accepted by researchers. Therefore, aside from using this metric to measure the fidelity of output, we will subjectively compare the output of edge detection in different settings to determine acceptable quality.

4.2.2 Scale

Scale is an implementation of the core part of an algorithm to scale an image to a larger size. It calculates the value of each pixel in the final result by mapping the pixel's location back to the original source image and then taking a weighted average of the neighboring pixels. We use the same fidelity metric as Image-Smoothing for Scale benchmark.

4.2.3 x264

This application is an H.264/AVC (Advanced Video Coding) video encoder [11]. PSNR of 32dB can provide satisfactory video quality for many types of videos.

4.3 Results

4.3.1 Leakage Energy Savings

Using the derived static power numbers from CACTI and our analytical fault models, we are able to analytically compare our proposed mechanism with a baseline cache. Figure 9 summarizes the achievable leakage energy savings in different settings. The baseline cache is assumed to use 700mV for supply voltage. We assume power-gating for disabled blocks. Because of that, if we keep voltage constant and increase AFB (meaning that we reclaim faulty memory blocks), the leakage energy savings decreases. Also note that, in voltages $\geq$ 540mV increasing AFB does not reduce energy savings. This is because of the low number of faulty blocks at those voltages. This analysis shows that depending on the acceptable knobs setting we can decrease leakage energy up to 74%.

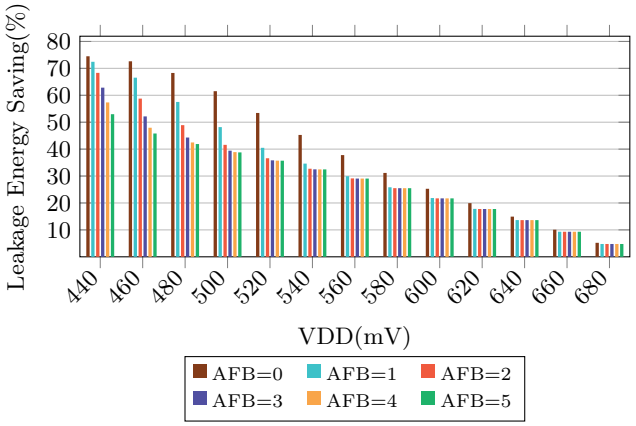

Figure 9: Leakage Energy Savings for a 4-Way L1 Cache with 3 Relaxed Ways. (Energy Savings are Normalized to a Baseline Cache that Uses 700mV.)

4.3.2 Fidelity Analysis

Figure 10 shows the PSNR difference between the images up-scaled using Scale benchmark run on a system with energy-efficient Relaxed Cache versus the ones that are processed with a system that uses guard-banded and energy-hungry baseline cache. The reported PSNR values are averaged for different image sizes. Comparing PSNR values with the reference PSNRs in Table 2 makes it clear that we can reduce the voltage to as low as 440 mV and tolerate 3 faults in each cache block, but still generate an acceptable up-scaled image.

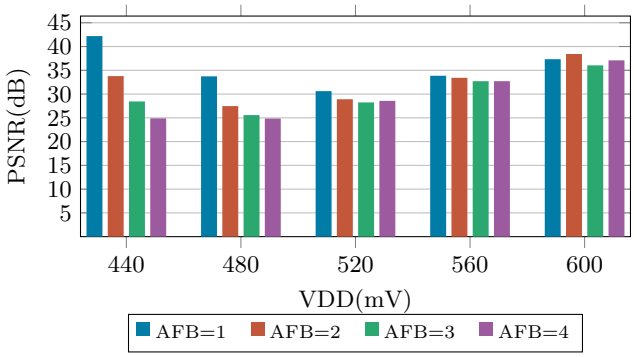

Figure 10: Fidelity Results for Scale Benchmark.

The PSNR values for Image Smoothing kernel of Susan package are reported in Figure 11. Same as Scale benchmark, many of the (VDD, AFB) settings result in an acceptable output. However, it is interesting to note that on average the PSNR values are larger than the ones reported for Scale. This shows that Image Smoothing benchmark is more error-tolerable and therefore the programmer can use a more aggressive policy for that kernel. This observation confirms our motivation for letting the programmer adapt the underlying cache's setting to the application that is using it, resulting in further energy saving.

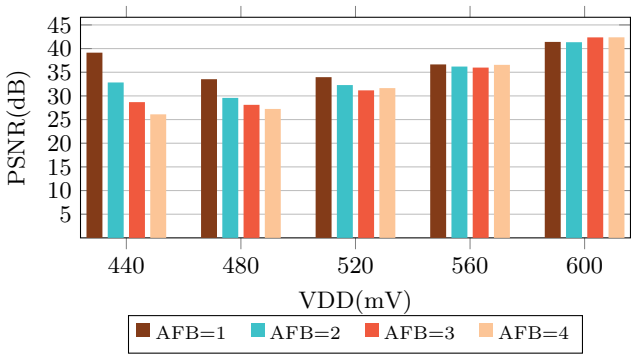

Figure 11: Fidelity Results for Susan/Image-Smoothing Benchmark.

Susan/Edge-Detection is the most error-tolerant application that can produce good result even at high error rates. Figure 12 shows the result of Susan/Edge-Detection on the Lena test image with different AFBs while VDD is set to 480mV. We can see that even setting AFB to 6, which reclaims all the faulty blocks at that voltage, can result in an acceptable output according to the Accuracy metric in [33].

4.3.3 *Performance Analysis*

Operating at low voltages makes part of the cache disabled. Reduced cache capacity has minor impact on performance of programs with small working set size. The degradations in total execution time are bounded by 3% for Scale and Edge-Detection benchmarks and 2% for Image-Smoothing benchmark over all (VDD, AFB) settings in our experiments. But for applications with big working set size, this reduced cache capacity considerably slows down the performance. However, using the AFB knob we are able to reclaim part of cache thus increase performance of the application. This is especially useful for applications in which generating a partial/less accurate result by deadline

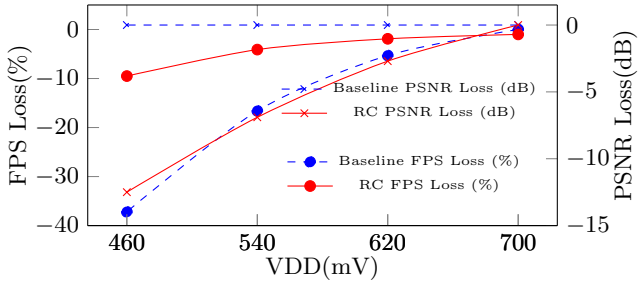

Figure 13: FPS-PSNR Trade-offs w/ and w/o Relaxed Cache Scheme (AFB=4)

is more valuable than a late, perfect result. Relaxed Cache provides the means for a programmer to explore this trade-off space. It's up to the programmer to decide when higher performance is desirable and when higher output fidelity is preferable. Figure 13 shows how Relaxed Cache prevents frame per second (FPS) of H.264 encoder to degrade severely in low cache voltages. This is possible by trading quality for increased FPS. However in all the experiments, PSNR of degraded-quality video was still above 32dB threshold, yielding acceptable quality for the user.

5. RELATED WORK

This work represents a convergence of three main bodies of research: voltage-scaled cache design, variability-aware memory management, and approximate computing.

Voltage-scaled Cache Design: There is a rich body of work on fault-tolerant voltage-scaled cache design at different design abstraction levels. A number of these works use circuit-level techniques to improve the reliability of each SRAM cell in near-threshold voltages. Apart from the familiar 6T SRAM cell, 8T SRAM cell and 10T SRAM cell [13] have been proposed. Most of these designs have a large area overhead which again poses a significant limitation for performance and power consumption of caches. Several schemes have been also proposed at the architecture level to save the energy by lowering the voltage. All of these approaches try to reduce the minimum operable cache Vdd-min with yield constraints by employing relatively sophisticated fault tolerance mechanisms, such as block/set pairing [45, 38], address remapping [2], block/set-level replication [8], etc. In another set of schemes, various complex error correcting codes (ECC) have been used to protect against both permanent and transient errors while reducing the Vdd-min [17, 44, 1, 36]. A recent work [25] proposes an architectural solution which dynamically lowers SRAM array voltage to sacrifice cache capacity for power saving. Our approach is different from the above techniques as we elevate the decision making about memory guard-bands to the application-level and allow the programmer to exploit the application's behavior for adaptive relaxation of guard-bands to push for more energy saving.

Exploiting Memory Variability: There are several efforts on exploiting variations in both off-chip and on-chip memories for power saving. Some of these efforts elevate exploiting memory variability through the software stack and let the programmer to manage variability but none

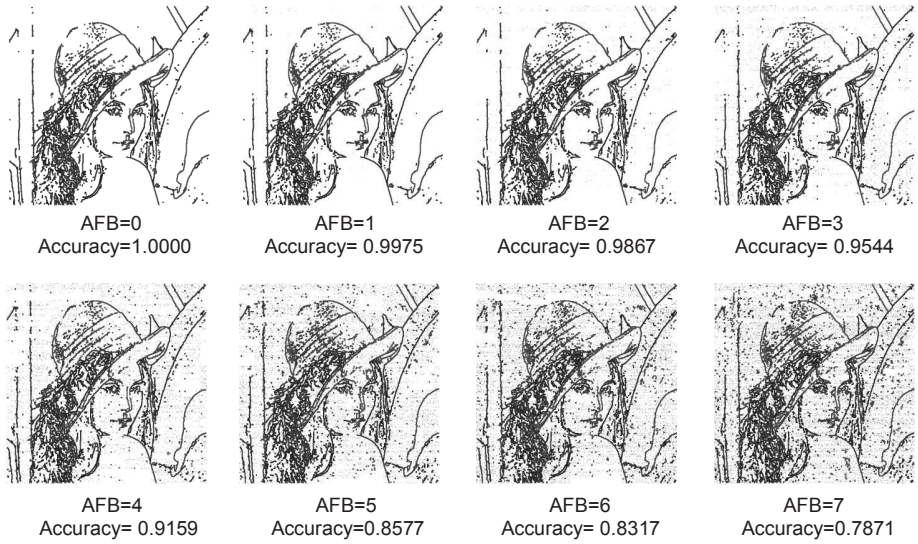

Figure 12: Fidelity Results for Susan/Edge-Detection Benchmark.

of them considers phasic behavior of an application. The work in [9] virtualizes on-chip and off-chip memory space to exploit the variation in the memory subsystem. They exploit this variability through a Variability-aware Memory Virtualization (VaMV) layer that allows programmers to partition their application's address space into regions and create mapping policies for each region. Bathen et al. proposed ViPZone [10], an OS-level solution to exploit DRAM power variation through physical address zoning for Energy Savings. ViPZonE is composed of a variability-aware software stack that allows developers to indicate to the OS the expected dominant usage patterns as well as level of utilization through high-level APIs. While Relaxed Cache is similar to such approaches in the sense of exploiting the memory variability, our proposal differs in two aspects. First, Relaxed Cache considers also phase behavior of application and criticality of data in its mapping policy. Second, Relaxed Cache intentionally relaxes the guard-bands under programmer control to gain more energy saving through matching different phases of application to the underlying hardware.

Approximate Computing: A significant amount of work has proposed different hardware and software techniques that trade-off application fidelity or correctness to gain benefits in energy, performance, lifetime, or yield. Many studies have shown that a variety of applications have a high tolerance to errors [20, 16, 29, 34, 30]. ERSA proposes collaboration between discrete reliable and unreliable cores for executing error resilient applications [29]. Esmaeilzadeh et al. proposed Truffle [23], a dual-voltage microarchitecture to support mapping of disciplined approximate programming onto hardware. Some software-based approaches use language extensions to give the ability to programmer to perform relaxations [39, 14]. EnerJ [39] uses type qualifiers to mark approximate variables. Using this type system, EnerJ automatically maps approximate variables to low power storage and uses low power operations to save energy. In contrast with EnerJ, Relaxed Cache lets the programmer change the criticality type of variables through the application to match its phasic behavior. Rely [14] is a language that allows specification of computations that execute on unreliable hardware and the analysis that ensures that the computation that executes on unreliable hardware satisfies its reliability specification. Rely considers an abstract model of the underlying hardware with a simple hardware reliability specification. It specifies the reliability of arithmetic/logical and memory operations by some probabilistic values. So it is limited to only soft errors.

Relax [19] is a compiler/architectural framework for exposing hardware errors to the software in specified regions of code for saving computational power. Relax allows programmers to mark certain regions of the code relaxed, and decrease the processor's voltage and frequency below the critical threshold when executing such regions. While Relax focuses on error recovery and hardware design simplicity, our Relaxed Cache approach emphasizes energy-efficiency over error detectability and supports a wider range of power-saving approximations. Moreover, Relax explores a code-centric approach, in which blocks of code are marked for failure and recovery while Relaxed Cache employs data-centric type annotations. Green [7] trades Quality-of-Service (QoS) for energy efficiency in server and high-performance applications respectively. Green allows programmers to specify regions of code in which the application can tolerate reduced precision. Based on this information, the Green system attempts to compute a principled approximation of the code-region (loop or function body) to reduce processor power.

While most of the previous efforts on approximate computing have focused on the computational part, there exist only two works that have addressed unreliable off-chip main memories. Liu et al. proposed Flikker [32], a software/hardware technique which utilizes the low-refresh rate part of DRAM for storing non-critical data. A programmer using Flikker doesn't have control on the refresh rate of the unreliable part of the DRAM. So s(he) won't be able to adjust this rate to the application's tolerable level of error. Sampson et al [40] proposed

two unreliable writes to solid-state main memories, based on Multilevel-cell (MLC) phase change memory (PCM) technology, for improving performance. Unlike prior work on memory, with Relaxed Cache, the programmer can adjust the dynamic knobs to the required capacity/reliability for each phase of program to achieve energy saving.

6. CONCLUSION AND FUTURE DIRECTIONS

We presented Relaxed Cache which exploits phasic and error-tolerance behavior of RMS applications to adapt the guard-banding in hardware to the software running on it. In Relaxed Cache scheme, we elevate decision making to the application programmer level, allowing the programmer to leverage semantic and domain knowledge of an application for energy saving. The application developer can use a set of language extensions for dynamic data criticality declarations and setting cache operational points in order to save energy. Our experimental results on sample RMS benchmarks show promising results. Our scheme can reduce leakage energy by up to 74%. We intend to explore adapting the guard-banding in the *entire* memory hierarchy of a system to the set of applications running of that system at a time.

References

[1] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu. Energy-efficient cache design using variable-strength error-correcting codes. In *ISCA*, 2011.

[2] A. Ansari, S. Feng, S. Gupta, and S. A. Mahlke. Archipelago: A polymorphic cache design for enabling robust near-threshold operation. In *HPCA*, 2011.

[3] A. Ansari, S. Gupta, S. Feng, and S. Mahlke. ZerehCache: Armoring cache architectures in high defect density technologies. In *MICRO*, 2009.

[4] ARM. *Cortex-A5 processor manual*. http://www.arm.com.

[5] ARM. *Cortex-A7 processor manual*. http://www.arm.com.

[6] ARM. *Cortex-R4 processor manual*. http://www.arm.com.

[7] W. Baek and T. M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, 2010.

[8] A. BanaiyanMofrad, H. Homayoun, and N. Dutt. FFT-cache: a flexible fault-tolerant cache architecture for ultra low voltage operation. In *CASES*, 2011.

[9] L. A. D. Bathen, N. D. Dutt, A. Nicolau, and P. Gupta. VaMV: Variability-aware memory virtualization. In *DATE*, 2012.

[10] L. A. D. Bathen, M. Gottscho, N. Dutt, A. Nicolau, and P. Gupta. ViPZonE: Os-level memory variability-driven physical address zoning for energy savings. In *CODES+ISSS*, 2012.

[11] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT*, 2008.

[12] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2), 2011.

[13] B. H. Calhoun and A. Chandrakasan. A 256kb sub-threshold SRAM in 65nm cmos. In *ISSCC*, 2006.

[14] M. Carbin, S. Misailovic, and M. C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *OOPSLA*, 2013.

[15] M. Carbin and M. C. Rinard. Automatically identifying critical input regions and code in applications. In *ISSTA*, 2010.

[16] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *DAC*, 2013.

[17] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu. Improving cache lifetime reliability at ultra-low voltages. In *MICRO*, 2009.

[18] M. Cohen, H. S. Zhu, E. E. Senem, and Y. D. Liu. Energy types. In *OOPSLA*, 2012.

[19] M. de Kruijf, S. Nomura, and K. Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *ISCA*, 2010.

[20] M. de Kruijf and K. Sankaralingam. Exploring the synergy of emerging workloads and silicon reliability trends. In *SELSE*, 2009.

[21] A. S. Dhodapkar and J. E. Smith. Comparing program phase detection techniques. In *MICRO*, 2003.

[22] C. Ding, S. Dwarkadas, M. Huang, K. Shen, and J. Carter. Program phase detection and exploitation. In *IPDPS*, 2006.

[23] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *ASPLOS*, 2012.

[24] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *ISCA*, 2002.

[25] M. Gottscho, A. BanaiyanMofrad, N. Dutt, A. Nicolau, and P. Gupta. Power/capacity scaling: energy savings with simple fault-tolerant caches. 2014.

[26] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *WWC*, 2001.

[27] S. Hamdioui, A. J. van de Goor, and M. Rodgers. March SS: A test for all static simple RAM faults. In *MTDT*, 2002.

[28] ITRS. http://www.itrs.net. 2013.

[29] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *DATE*, 2010.

[30] X. Li and D. Yeung. Application-level correctness and its impact on fault tolerance. In *HPCA*, 2007.

[31] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-aware intelligent DRAM refresh. In *ISCA*, 2012.

[32] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flikker: Saving DRAM refresh-power through critical data partitioning. In *ASPLOS*, 2011.

[33] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE TPAMI*, 2004.

[34] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. C. Rinard. Quality of service profiling. In *ICSE*, 2010.

[35] N. Muralimanohar et al. CACTI 6.5: A tool to model large caches. Technical report, HP Laboratories, 2009.

[36] M. K. Qureshi and Z. Chishti. Operating secded-based caches at ultra-low voltage with flair. In *DSN*, 2013.

[37] V. J. Reddi, D. Z. Pan, S. R. Nassif, and K. A. Bowman. Robust and resilient designs from the bottom-up: Technology, CAD, circuit, and system issues. In *ASP-DAC*, 2012.

[38] D. Roberts, N. S. Kim, and T. N. Mudge. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. *Microprocessors and Microsystems - Embedded Hardware Design*, 2008.

[39] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: Approximate data types for safe and general low-power computation. In *PLDI*, 2011.

[40] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate storage in solid-state memories. In *MICRO*, 2013.

[41] A. Sasan, H. Homayoun, A. Eltawil, and F. Kurdahi. Process variation aware SRAM/cache for aggressive voltage-frequency scaling. In *DATE*, 2009.

[42] J. Wang and B. Calhoun. Minimum supply voltage and yield estimation for large SRAMs under parametric variations. *TVLSI*, 2011.

[43] J. Wang and B. H. Calhoun. Standby supply voltage minimization for reliable nanoscale SRAMs. In *Solid State Circuits Technologies*. InTech, 2004.

[44] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu. Reducing cache power with low-cost, multi-bit error-correcting codes. In *ISCA*, 2010.

[45] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. Trading off cache capacity for reliability to enable low voltage operation. In *ISCA*, 2008.

[46] W. Zhang and T. Li. Characterizing and mitigating the impact of process variations on phase change based memory systems. In