Parichute: Generalized Turbocode-Based Error Correction for Near-Threshold Caches*

Timothy N. Miller, Renji Thomas, James Dinan, Bruce Adcock, Radu Teodorescu
Department of Computer Science and Engineering
The Ohio State University
{millerti, thomasr, dinan, adcockb, teodores}@cse.ohio-state.edu

Abstract

Energy efficiency is a primary concern for microprocessor designers. A very effective approach to improving the energy efficiency of a chip is to lower its supply voltage to very close to the transistor's threshold voltage, into what is called the near-threshold region. This reduces power consumption dramatically but also decreases reliability by orders of magnitude, especially for SRAM structures such as caches.

This paper presents Parichute, a novel and powerful error correction technique based on turbo product codes that allows caches to continue to operate in near-threshold, while trading off some cache capacity to store error correction information. Our Parichute-based cache implementation is flexible, allowing protection to be disabled in error-free high voltage operation and selectively enabled as the voltage is lowered and the error rate increases. Parichute is also self-testing and variation-aware, allowing selective protection of cache sections that exhibit errors at higher supply voltages because of process variation. Parichute achieves significantly stronger error correction compared to prior cache protection techniques, enabling 2× to 4× higher cache capacity at low voltages. Our results also show that a system with a Parichute-protected L2 cache can achieve a 34% reduction in system energy (processor and DRAM) compared to a system operating at nominal voltage.

1 Introduction

The computing devices of tomorrow, from smartphones to desktops to servers, are expected to become increasingly energy-conscious. In portable devices, the demand for lower power is driven by expectations for lighter devices with longer battery life. In the desktop and server markets, hard limits on thermal design power [22] of chips will constrain the growth in the number of cores in future general-purpose CMPs [37]. Maintaining the expected growth in performance while keeping power under control will require dramatic improvements in the energy efficiency of microprocessors.

A very effective approach to improving the energy efficiency of a chip is to lower its supply voltage (V_{dd}) to very close to the transitor's threshold voltage (V_{th}), into what is called the near-threshold (NT) region [8, 13, 21]. This is significantly lower than what is used in standard dynamic voltage and frequency scaling (DVFS), resulting in dramatic reductions in power consumption (up to $100\times$) with about a $10\times$ loss in maximum frequency. Even with the loss in performance, chips running in near-threshold often achieve significant improvements in energy efficiency. In fact, prior work has shown that the lowest energy per instruction is often achieved in the sub-threshold or near-threshold regions [8, 13]. In a power-constrained CMP, near-threshold operation will allow more cores to be powered on (albeit at much lower frequency) than in a CMP at nominal V_{dd}. Despite lower individual core throughput, aggregate throughput can be much higher, especially for highly parallel workloads. This makes NT CMPs very attractive for systems ranging from portable devices to energy-efficient servers.

The energy reduction in NT, however, comes at the cost of severe degradation in reliability. Large on-chip SRAM structures such as the L2 and L3 caches are the most vulnerable to errors [9, 13]. They are optimized for area and power and therefore built with the smallest transistors. In near-threshold, such a cache can experience error rates that exceed 4%, rendering an unprotected structure virtually useless. Thus, in order to harness the energy savings of NT operation, the high error rates of large SRAM structures must be addressed.

This paper proposes *Parichute*, a novel forward error correction (FEC) technique based on a generalization of turbo product codes that is powerful enough to allow caches to continue to operate reliably in near threshold with error rates exceeding 7%. *Parichute* leverages the power of iterative decoding to achieve very strong correction ability while having a relatively low impact on cache access latency. Our *Parichute*-based cache implementation dynamically trades off some cache capacity to store error correction information. It is flexible and adaptive, allowing protection to be disabled in error-free high voltage operation and selectively enabled as the voltage is lowered to near-threshold and the error rate increases. *Parichute* is self-testing and variation-aware, allowing selective protection of cache sections that exhibit errors at higher supply voltages due to process variation.

Compared to previous cache error protection solutions targeting high error rates [10, 19], *Parichute* provides significantly stronger error correction for the same parity storage overhead, with similar decoding hardware costs and only slightly higher decoding latency. We demonstrate that *Parichute*'s error correction is significantly more effective than a state of the art solution based on Orthogonal Latin

*This work was supported in part by the National Science Foundation under grant CNS-0403342.

Square Codes (OLSC) [10]. At near-threshold voltages a *Parichute*-protected cache has between $2\times$ and $4\times$ the capacity of a cache protected by OLSC.

We also show that a processor with a *Parichute*-protected L2 cache running in near-threshold achieves a 34% reduction in system energy (processor and DRAM) compared to a system operating at nominal voltage. If the same system uses standard SECDED (Single Error Correction, Double Error Detection) protection for its cache, it achieves almost no reduction in energy due to the much lower cache capacity and the resulting increase in miss rates.

We synthesized a prototype implementation of the *Parichute* hardware for 45nm CMOS to show that it can be implemented with low hardware overhead and would add little additional access latency to an L2 cache. The *Parichute* encoder and decoder logic occupies less than 0.06mm^2 of die area and uses less than 12mW, negligible overhead for a state of the art processor.

Overall, this paper makes the following contributions:

- Introduces *Parichute ECC*, a novel error correction technique based on a generalization of turbo product codes that is very powerful, adaptive, lightweight, and amenable to efficient hardware implementation.
- Presents a *Parichute*-enabled adaptive cache architecture, designed to operate efficiently in very high error rate environments at near-threshold and in error-free environments at nominal voltage.
- Evaluates the power and area overheads of a *Parichute ECC* prototype synthesized for 45nm CMOS.
- Demonstrates the energy benefits of near threshold operation of a processor with *Parichute*-protected caches.

Section 2 provides an overview of near-threshold operation and its reliability implications, as well as an overview of other error correction schemes. Section 3 describes the *Parichute* error correction solution. Section 4 details the architecture of a *Parichute*-protected cache. Section 5 presents a prototype design of the *Parichute* hardware. An experimental evaluation is presented in Sections 6 and 7. Finally, Section 8 details related work, and Section 9 concludes.

2 Background

This section provides background on near-threshold operation, focusing on the energy benefits and reliability challenges of this technique. It also presents a few existing error correction techniques relevant to this work.

2.1 Near Threshold Computing

Standard dynamic voltage and frequency scaling typically reduces V_{dd} to no lower than 70% of the nominal level in order to guarantee reliable chip operation. In near-threshold operation, the V_{dd} is scaled more aggresively to $25 - 35\%$ of nominal levels, close to the transistor threshold voltage (V_{th}) [13]. With V_{dd} this low, transistors no longer operate in the saturation region. As a result, chip power consumption is around $100\times$ lower than at nominal V_{dd}. These power savings, however, come at a cost of decreased switching speeds (about $10\times$) and decreased reliability.

Transistors in near threshold are much more affected by process variation. Process variation is caused by manufacturing difficulties in very small feature technologies [4] and refers to deviations in transistor parameters beyond their nominal values. Several parameters are impacted by variation. Of key importance is the threshold voltage because it directly impacts a transistor's switching speed and leakage power. Process variation has both random and systematic effects. The systematic component of variation is spatially correlated, meaning that transistors close to each other on the die tend to have similar characteristics. In near-threshold, because V_{dd} is very close to V_{th}, variation in V_{th} will have a much more pronounced effect on transistor speed and leakage current compared to nominal voltage. At nominal V_{dd}, variance in transistor delay tends to be relatively small, but it increases significantly as V_{dd} is lowered towards V_{th}.

Large SRAM arrays are especially vulnerable to variation at low V_{dd} [1, 6, 7, 25, 41]. They are optimized for area and power and therefore built using the smallest transistors, which are the most affected by random variation. Random variation among the transistors in an SRAM cell can create imbalance between the back-to-back inverters, and as the voltage is lowered the cell may become unable to reliably hold a value. Variation can also make the cell too slow to access; although it may hold a value, one or both access transistors may pull down its bit-line so slowly that the cell cannot be read in a reasonable time.

2.2 Error Correcting Codes

Error-correcting codes (ECC) are functions for encoding data in a way that allows errors to be detected and corrected. Simple ECCs are used in server memory [12, 28, 33] to tolerate bit upsets that occur as a result of faulty RAM cells, single event upsets, and other disruptive factors such as voltage fluctuations [24]. More sophisticated codes are used in everything from digital communications to error-prone storage media such as flash drives and hard disks.

All ECCs require redundant bits, or *parity*, to be added to the data bits being protected. A grouping of data bits and their corresponding parity bits form a *code word*. The number of bits that differ between two error-free code words (called the *Hamming* distance) dictates how many errors the ECC can correct in each codeword. An ECC with minimum Hamming distance of d can correct $\lfloor (d-1)/2 \rfloor$ errors.

2.2.1 Single-Bit Error Correcting Codes

The most common ECCs correct single bit errors and are refered to as *SEC* (Single Error Correction) codes. They have a minimum Hamming distance of 3, where a 1-bit error creates a code word that is nearest to only one error-free code word. Adding one additional parity bit, for a minimum Hamming distance of 4, allows correction of 1-bit errors and detection of 2-bit errors. This is called a *SECDED* code (Single Error Correction, Double Error Detection). *Parichute* uses SECDED codes as part of a more sophisticated protection scheme.

2.2.2 Orthogonal Latin Square Codes

A more powerful ECC based on Orthogonal Latin Square Codes (OLSC) [17] was used for cache protection in [10].

OLSC creates a parity matrix from orthogonal groupings ("latin squares") of data bits and associates parity to them. Given a word with m^2 data bits, up to $m/2$ errors can be corrected. OLSC correction involves a single-step majority voting across multiple parity encodings for each data bit. This requires a significant amount of hardware but can be implemented with low latency [10]. Although OLSCs are suitable for moderate error rates, they have limited performance in the presence of the high error rates observed at deep NT.

2.2.3 Turbo Product Codes

A *Product Code* [15] is an ECC made from a composition of multiple *short codes* that make up a *long code*. Data is typically arranged in a 2D matrix, and *short code* words are computed from the data in each column and each row. This creates a much more powerful ECC by applying a simpler ECC to two orthogonal *permutations* of the same data. Various ECCs have been used for the short code, as in [3, 16, 29].

A product code is called a *Turbo Product Code (TPC)* if iterative decoding for the long code word is performed by arranging short code decoders in a cycle; corrections are computed for rows and columns separately, and decoders iteratively exchange intermediate results. The orthogonal data layout allows each bit to receive protection twice, once in its column and once in its row. Errors uncorrectable with respect to one permutation may be correctable with respect to the other. Moreover, an error that is uncorrectable if each permutation is considered independently may be correctable through iterative decoding, where each correction builds on the results of the previous.

Parichute generalizes block TPC by using more data permutations to make more flexible use of the storage space used to hold parity. TPCs are typically used for signal communications, where processing power is much greater than the channel bit rate, making it practical to use probablistic (soft-decision) decoding. To minimize latency and logic overhead, *Parichute* correction is entirely binary (hard-decision).

3 The Parichute ECC

Parichute defines two mechanisms: *Parichute ECC*, a novel error correction algorithm, and the *Parichute Cache* architecture which applies *Parichute ECC* to maximize cache capacity at ultra-low voltages. *Parichute ECC* is an enhancement to turbo product codes that has strong correction ability, efficient use of parity bits, and low decode latency.

3.1 Generalized Turbo Product Codes

The *Parichute Cache* protects data by storing parity bits in other cache lines, using either whole lines or fractional lines. Under these constraints a straight-forward application of TPC as in [16] is too rigid and suboptimal because it requires a fixed number of parity bits that may not match well with the cache line size. For instance, given N data bits, a TPC that uses SECDED as a short code, requires $2H\lceil\sqrt{N}\rceil$ parity bits, where H is the number of parity bits for $\lceil\sqrt{N}\rceil$ data bits. For a 512-bit cache line, the data is arranged roughly into a 23×23 matrix, where each column and each row requires 6 parity bits, for a total of 276. These do not fit in half of a cache line and would waste space in a full line.

Considering these inefficiencies, we have designed *Parichute ECC* to offer a more flexible layout. *Parichute ECC* allows the selection of an arbitrary number of data permutations and much greater flexibility in the number of parity bits mapped to the long data word. These make *Parichute ECC* a more space-efficient and stronger ECC than standard TPC.

For a given data word size and available parity space, multiple configurations are possible under certain constraints. *Parichute ECC* uses SECDED as its short code to protect data slices. The short data slice length is $S \leq 2^{H-1} - H$, where H is the number of SECDED parity bits. With *Parichute*, we can impose an additional constraint on the total number of parity bits for the long code. For a long data word of size N, let M_{max} be the budget for parity bits, M the actual number of parity bits used, and P the number of permutations. The following relation must hold:

$$\lceil N/S \rceil \times P \times H = M \leq M_{max}$$

There are typically multiple valid combinations of values for H, P, and S. For instance, if cache lines are 512 bits, and the goal is to fit parity into half of a cache line ($M_{max} = 256$), one option is to use 7-bit parity words (57-bit data slices) and 4 data permutations. Another would be to use $H = 6$ ($S = 26$), which would require that $P = 2$. In our implementation we evaluate all feasible configurations and pick the best one.

3.2 Optimization of the Parity-Data Association

With *Parichute ECC*, data permutations are no longer trivially orthogonal, making a good mapping of parity to data very important for maximizing correction ability. Each permutation offers an opportunity for correcting a subset of all possible errors, and the correction ability improves when the possibility of uncorrectable errors in multiple permutations is minimized. To that end, the number of times any two data bits are protected by the same parity word in multiple permutations should be minimized. This is because if both bits fail and are protected by the same parity word in multiple permutations, they will be uncorrectable in all of these permutations.

Finding the set of *optimal* permutations is NP-hard. We therefore use a greedy randomized search of the solution space to find a good solution.

3.3 Parichute Error Correction Example

Figure 1 shows an example of correcting a corrupted data line protected by *Parichute ECC*. In this example, a 512-bit data line is protected by 252 parity bits in 9 slices. There are 4 permutations, and each is decoded by a dedicated corrector. There are five corrupted bits: a, b, c, d, and e. Because each permutation arranges data differently, the errors end up in different slices in each corrector. For instance bit a is in slice 0 for corrector 0 (C_0^0) and in slice 8 for corrector 1 (C_8^1).

On **cycle 0**, corrector 0 can only correct bit a in C_0^0 because the rest are in multi-bit errors in their slices. Since SECDED is used, only single-bit errors can be corrected in each slice. Corrector 1 can correct bit d in C_8^1, and corrector 2 can correct bit b in C_8^2. While corrector 3 can fix both e and b, the 3-bit error in C_1^3 is mistaken for a 1-bit error, resulting in the

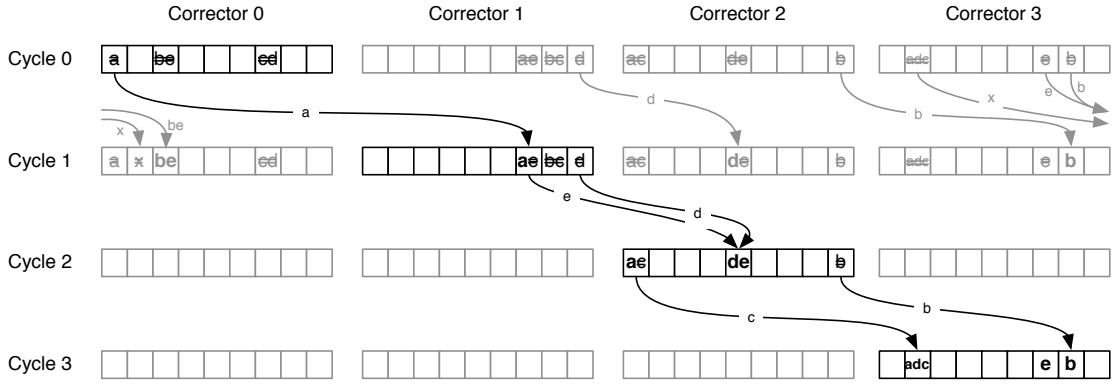

Figure 1: *Parichute* error correction example. Bits a, b, c, d, and e are corrupted, and arrows indicate the propagation of corrected bits. The successful correction path is emphasized.

additional corruption of bit x. Following the successful correction path, on **cycle 1**, corrector 1 corrects bits e and d, and on **cycle 2** corrector 2 corrects bits b and c. Finally, on **cycle 3**, the correction is complete, and the data is sent to its destination in **cycle 4**. Note that the vast majority of 5-bit errors will require only one correction cycle.

4 Parichute Cache Architecture

We use *Parichute ECC* to design an L2 cache that is resilient to the high error rates encountered in near-threshold operation. The *Parichute Cache* is variation-aware and adaptive, allowing protection to be disabled in error-free high voltage operation and selectively enabled in near-threshold as errors increase. Hardware support for encoding and correction is added to the cache controller, and all reads and writes to cache lines that need protection will go through this hardware. A high-level overview is shown in Figure 2.

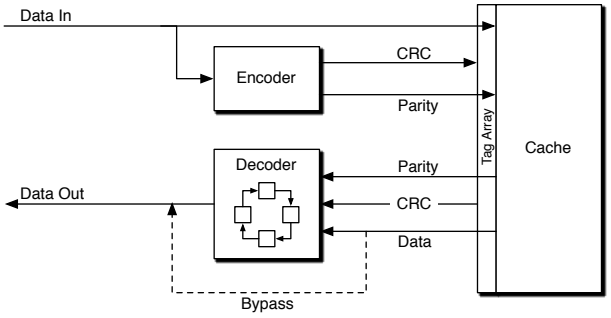

Figure 2: High-level overview of *Parichute* cache architecture. For lines requiring no protection, decoding is bypassed, which reduces access latency.

4.1 Hardware for Parichute Encoding and Correction

Parichute uses a hardware encoder, shown in Figure 3(a), to generate multiple permutations of data through a hard-wired *permutation network*. For each permutation, a *parity encoder* computes the SECDED parity for each data slice in the permutation. The parity bits of all slices are concatenated into a *parity group*, shown in Figure 3(b). Finally, the concatenation of all parity groups constitutes a *Parichute parity block*.

In parallel with parity generation, the encoder also generates a CRC for the data to be written to the cache and stores it in the tag array. This CRC is used by the decoder to determine if the data was successfully corrected and to verify potential corruption of unprotected data.

The *Parichute* decoder is responsible for correcting corrupted data. It is composed of P correctors, arranged in a circular path, illustrated in Figure 4. Each corrector loads its own copy of the data and parity bits. A corrector is hardwired to decode one data permutation utilizing M/P parity bits. For each S-bit data slice and its corresponding H-bit parity word, the corrector indicates either that a specific bit out of the $S + H$ is corrupt or that two unknown bits are corrupt. After correction is applied, data and parity propagate to the next corrector.

A CRC is generated for the current data in parallel with the next correction cycle. This CRC is compared to the CRC from the tag to determine when the correction is complete. When there is a CRC match, correction stops, and the correct data is taken from the registers in the following corrector.

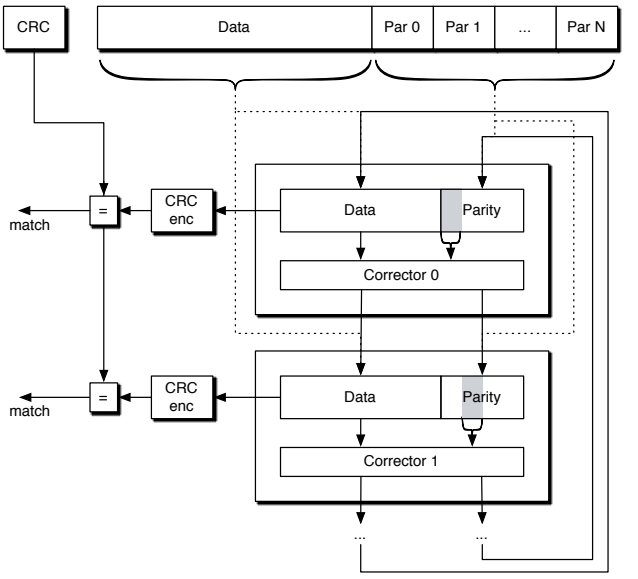

Figure 4: Diagram of full decoder circuit, with multiple parallel correctors in a cycle. Each corrector applies corrections based on its own parity group (indicated in gray) and then passes data and parity to the next corrector. Data is also validated against a CRC to determine if correction has succeeded.

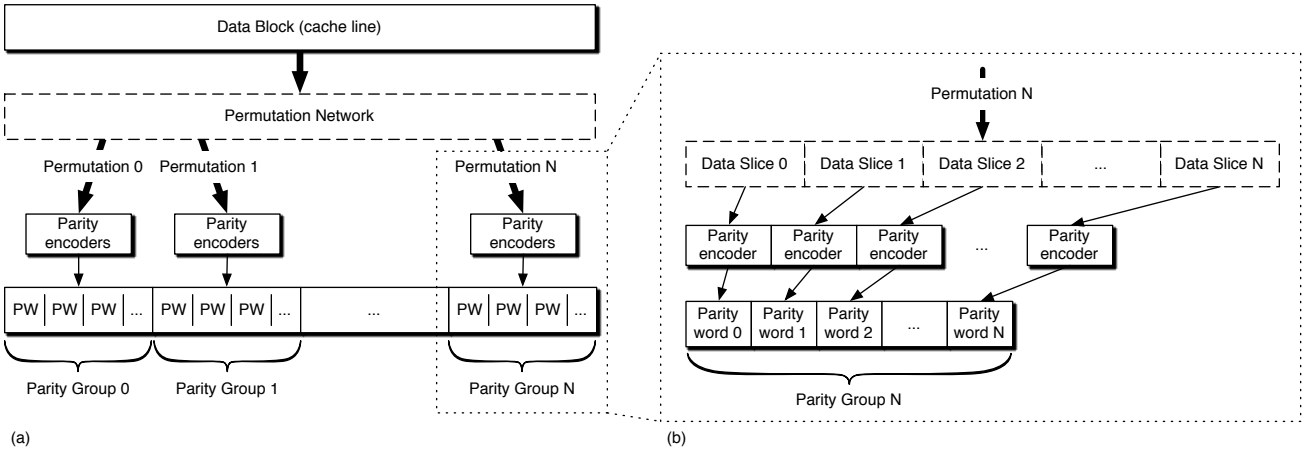

Figure 3: (a) Complete parity encoding data-path. The permutation network generates multiple data permutations that are sent to a set of parity encoders, which produce sections of the complete parity block. (b) Detail on parity encoders for one permutation.

For uncorrectable errors, decode terminates through a time-out, and a correction exception is reported. Uncorrectable lines are flagged through testing and will not be used when the cache is at a voltage level they cannot support (Section 4.3.2).

4.2 Parity Storage and Access

Parichute parity for a line is stored in a different way (of associativity) of the same cache set as the data. As protection is added to more data lines, the associativity of each set decreases. To allow concurrent access to the data line and its associated parity, we organize the cache such that each way of a set is in a separate bank, and we assume an access bus wide enough to accommodate both data and parity, as in [10].

For each line, the tag is extended to store the CRC as well as a pointer to indicate where the corresponding parity, if any, is stored. For an 8-way cache with half-line parity, a 4-bit pointer is required; for whole-line parity, 3 bits are sufficient.

The tag array also needs some form of protection. Since the tag array occupies a small fraction of the total area of the cache, we choose to harden it by using larger transistors or more robust 8T or 10T SRAM cells [9]. Alternatively *Parichute ECC* could be used to protect tag entries.

4.3 Dynamic Cache Reconfiguration

Parichute protection is dynamically enabled when the supply voltage is lowered. The naïve approach is to enable protection for all lines when the voltage drops below a certain point. However, because of process variation, errors have a non-uniform distribution, which makes some lines more vulnerable than others. We therefore propose a variation-aware protection algorithm that considers the relative vulnerability of cache lines in the assignment of protection.

Lightweight testing is performed either post-manufacturing or at boot time to determine the relative vulnerability of cache lines in near-threshold operation. Using these rankings, a cache configuration that maximizes capacity can be selected. These configurations are stored in on-chip ROM or main memory and loaded before the processor transitions into near-threshold.

4.3.1 Classifying Cache Lines

Testing is performed with simple built-in self test (BIST) circuitry [11] that writes and reads each line. Two test patterns are written to each line: one containing all 0's and one with all 1's. The patterns are read through the correctors, which also receive the precomputed parity for those bit patterns. Lines are classified based on the correction outcome. Those with no errors are marked as *Good*. *Bad* lines are ones that have correctable errors. *Ugly* lines have errors in number and position that render them completely unable to reliably store data, even with protection, and therefore should be disabled.

Testing can also be performed on-line. A *tested* bit is added to the tag for each cache line. On power-up, all *tested* flags are cleared. Before writing to an untested line, a test sequence is initiated, the line is classified, and its tested bit is set. Line classification is refined throughout execution to account for transistor aging and other effects. A *Good* line can start to experience errors, which we detect through CRC checks but cannot correct. The line is then reclassified as *Bad*. A *Bad* line can also be downgraded to *Ugly* if correction fails.

4.3.2 Variation-aware Protection

We examine multiple levels of variation-awareness that can inform decisions about which lines need protection. A variation-unaware solution will protect all cache lines. Identifying which lines are *Good* allows them to not have associated parity, which increases cache capacity. Adding the ability to distinguish *Ugly* lines allows these to be completely disabled, which avoids wasting cache capacity by adding protection to lines that cannot be corrected. Figure 5 illustrates an example of variation-aware data and parity mapping.

We have also considered ranking *Bad* lines according to the their relative "quality." *Parichute* parity bits are more vulnerable than data bits, because while each data bit is protected once for each permutation, parity bits are only indirectly protected through the data. If parity is stored in relatively "better" lines, overall capacity should increase because more lines should be correctable. However, in practice we find little increase in cache capacity as a result of this optimization.

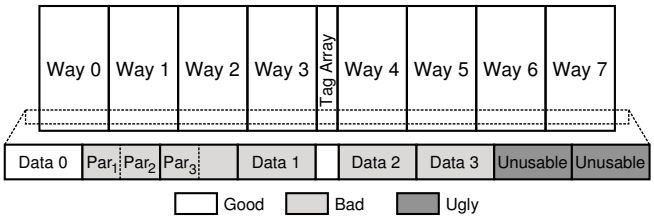

Figure 5: Example data and parity assignment for a cache set of an 8-way set associative cache. Data 0 is assigned to a *Good* line without parity. Data 1–3 are and their associated parity are assigned to *Bad* lines (parity 1 and 2 share a line). *Ugly* lines are disabled.

4.4 Cache Access Latency

Parichute adds some additional latency to cache accesses. All writes incur one cycle of additional latency because of CRC and/or parity generation. For writes to *Good* lines, only the CRC is generated and stored in the tag. For writes to *Bad* lines, parity is also generated and stored simultaneously with data in a different cache way.

Reads from *Good* lines are validated against the CRC, which incurs one cycle additional latency. Reads from *Bad* lines require that both data and parity be sent to the decoding hardware for correction. *Parichute* correction introduces variable access latency, although latency tends to be small even for significant numbers of errors (average of 4 decode cycles).

5 Prototype of Parichute Hardware

We designed and synthesized a circuit that implements a *Parichute* encoder and decoder. This prototype was coded in synthesizable Verilog and implements one of the *Parichute* configurations that yielded the best results in terms of cache capacity and is the same configuration used in our experimental evaluation. Using the notations in Section 3.1, the encoder and decoder target cache lines with $N = 512$ data bits, has $P = 4$ permutations, and utilizes data slices of $S = 57$ bits with $H = 7$ parity bits. The total number of parity bits $M = 252$ fits well in half of a cache line.

The 4-permutation *Parichute* encoder is comprised of 36 SECDED encoders. Each SECDED encoder takes in $S = 57$ bits and outputs $H = 7$ parity bits. Each output of the SECDED encoder is an XOR of 31 data bits. Parity bits are computed such that if one or two of the 64 data or parity bits is inverted, a "syndrome" can be computed that indicates either which bit is wrong or that there are two incorrect bits.

The decoder consists of four correctors, one for each permutation. Each corrector takes as input $N = 512$ data bits and the $M/P = H \cdot 9 = 63$ parity bits that correspond to its permutation. Each corrector is comprised of 9 syndrome generators that determine incorrect data or parity bits. Each syndrome generator takes as input $S = 57$ data bits and $H = 7$ parity bits and computes a 7-bit syndrome.

The *Parichute* encoder and decoder also use CRC generators to calculate data CRCs according to the CRC-16-CCITT [20] standard, which has a generator polynomial of $x^{16}+x^{12}+x^5+1$. We generated Verilog code that computes the entire CRC in one step, where each bit of the resulting CRC is an even parity (XOR reduction) of a particular subset of the data bits. The *Parichute* encoder uses one CRC generator to compute the CRC in parallel with parity generation. The *Parichute* decoder uses four CRC generators (one for each corrector) to compute the CRC for the data being decoded. At each correction step CRC is computed and compared to the correct CRC for that data. The CRC generator plus comparator require 14 levels of logic that create a long critical path. To break this path, we register the output of the CRC encoder and perform the comparison against the correct CRC in the following cycle. On a CRC match the data sent to the CPU is taken from the register in the subsequent corrector.

Parichute's hardware was synthesized for 45nm CMOS technology. First, Formality [35] was used to check the Verilog HDL description. Once verified, the Verilog HDL was linked to Nangate's Open Cell Library [27]. To work with Synopsys, the Liberty standard delay format library was converted to a Synopsys compatible database. This database was used to synthesize the logic with a clock constraint of 1ns. The design was compiled using Synopsys Design Compiler [36] iteratively to achieve timing closure. The resulting compiled design was then verified post-synthesis using Formality for functional correctness. The synthesized chip is used to determine critical path details, gate count, area, and power estimate. Section 7.5.1 provides synthesis results.

6 Evaluation Methodology

To evaluate *Parichute*, we use the following infrastructure. An SRAM model at near-threshold and a process variation model are used to estimate error rates and distributions for an L2 cache. Multiple correction models, including *Parichute*, OLSC and SECDED, are applied to the cache model to determine cache capacity and correction latency under various error conditions and configurations. The resulting cache capacities and access latencies are used by a multicore processor model to evaluate the impact of the cache protection techniques on the performance and energy of a state of the art processor and memory. The *Parichute* prototype is used to determine critical path delay, area, and power consumption for the *Parichute* hardware, included in the processor model.

6.1 SRAM Model at Near Threshold with Variation

To model SRAM cells in near threshold and in the presence of process variation, we performed SPICE simulations of an SRAM block implemented in 32nm CMOS. We used the Cadence Spectre Circuit Simulator (IC611 package), with the BSIM4 V2.1 32nm PTM HP transistor model [42]. Read, write and access failure tests were conducted for a single 6T SRAM cell, along with a bit-line conditioning circuit for precharging the bit lines and a sense circuit for reading the memory cell. This model is used to determine the dynamic and leakage power and read and write delays for different supply and threshold voltage values. The parameters of NMOS and PMOS transistors were simultaneously swept between 0.15-0.9V for the V_{dd} and between $\pm 30\%$ of nominal V_{th}.

We use VARIUS [32] to model process variation and we consider both random and systematic effects. When modeling systematic variation we make the simplifying assumption that transistors are fully correlated within a cache bank

Shared L2	8-way 2 MB, 10-16 cycle access
L1 data cache	8-way 16K, 1-cycle access
L1 instruction cache	2-way 16K, 1-cycle access
Branch prediction	2K-entry BTB, 12-cycle penalty
Fetch/issue/commit width	3/3/3
Register file size	40 entry
Technology	32nm, 3GHz (nominal)
Nominal V_{dd}	0.9V
Near threshold V_{dd}	0.3375-0.375V
V_{th} μ,	150mV
V_{th} σ	$\sigma_{ran} = 4.8\%$ and $\sigma_{sys} = 1.8\%$

Table 1: Summary of the architectural configuration.

and uncorrelated across banks. Random variation is modeled as a normal distribution across all bit cells. For systematic and random components of variation, $\sigma_{ran} = 4.8\%$ and $\sigma_{sys} = 1.8\%$. Variation parameter values are taken from ITRS [18] predictions and [32].

We use the variation model to generate distributions of threshold voltages (V_{th}). Using the SRAM access failure data (reads and writes) as a function of V_{dd} and V_{th} from the SPICE simulations, we generate error distributions, which we use to model cache bit failures.

6.2 Cache Error Correction Models

We built models for the three cache protection schemes we evaluate: *Parichute*, SECDED and OLSC. We use these models in Monte Carlo simulations of 100 cache profiles with variation, at multiple voltages. We modeled 8-way 2 MB caches, with 512-bit lines. For each line a random data pattern is assigned, and parity is computed. Bits are corrupted randomly for each line according to a probability distribution given by the variation pattern and error model. Error correction is attempted with each cache protection scheme. Only corrections of entire lines are considered successful.

6.3 Near-Threshold Processor Model

To evaluate the impact of *Parichute*-enabled caches on performance and energy, we use a modified version of the SESC simulator [31]. SESC is configured to simulate a system similar to the Intel Core 2, with a *Parichute*-protected L2 cache. For the performance and energy simulations, we use the SPEC CPU2000 benchmarks, SPECint (*crafty, mcf, parser, gzip, bzip2, vortex,* and *twolf*) and SPECfp (*wupwise, swim, mgrid, applu, apsi, equake,* and *art*). We decrease the sizes of the L1 and L2 caches to 16KB and 2MB respectively to increase the pressure on the L2 cache from the SPEC benchmarks. Table 1 summarizes the architecture configuration.

We use SESC, augmented with dynamic power models from Wattch [5] and CACTI [26], to estimate dynamic power at a reference technology and frequency. We scale these numbers using our own model for near-threshold, based on SPICE simulations of SRAM and logic cells. We use the variation-aware leakage model from [32] to estimate leakage power. We also model main memory dynamic power with an approach similar to that in [43].

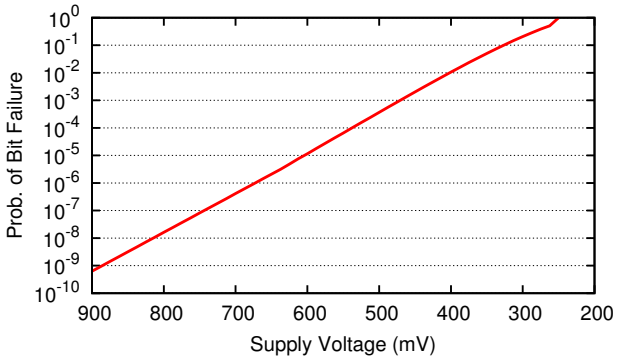

Figure 6: Voltage versus probability of bit failure (log scale). As V_{dd} is lowered, the probability of failure increases exponentially, exceeding 2% at near-threshold.

Name	*Description*
No Protection	No error correction
SECDED	Extended Hamming, 64 parity bits ($H = 8$)
Parichute 126	*Parichute*, 126 parity bits ($H = 7, P = 2$)
Parichute 252	*Parichute*, 252 parity bits ($H = 7, P = 4$)
Parichute 504	*Parichute*, 504 parity bits ($H = 7, P = 8$)
OLSC 128	Orthogonal Latin Square Codes, 128 parity bits
OLSC 256	Orthogonal Latin Square Codes, 256 parity bits
OLSC 512	Orthogonal Latin Square Codes, 512 parity bits

Table 2: Summary of the error correction techniques used.

7 Evaluation

This section evaluates *Parichute* and compares it to other state of the art correction solutions. We show that its superior correction ability results in very good cache capacity in near-threshold. We also examine the implications of near-threshold operation on system energy and show that *Parichute* enables significantly lower energy operation. Finally, we examine the area, power, and delay overheads of the *Parichute* prototype implementation.

7.1 Error Rates in SRAM Structures

We first examine the probability of SRAM bit failure as a function of V_{dd}. Figure 6 shows results from Monte Carlo simulations based on our error model and SPICE simulations. Error rate increases rapidly as V_{dd} is lowered. Our experiments focus on three near-threshold voltage levels: 375mV, where the error rate is 2.3%; 350mV, with 5% error; and 337.5mV, with 7.3% error.

7.2 Parichute Error Correction Ability

To evaluate the correction ability of *Parichute ECC*, we perform Monte Carlo simulations over a large number of cache lines with different numbers of bad bits. Errors have a uniform distribution. We compare *Parichute* to OLSC [10] for different numbers of parity bits. We also compare to SECDED error correction which uses 8 bits of parity to protect for 64 bits of data, for a total of 64 parity bits per line. Table 2 lists the error correction schemes examined. All experiments assume a cache line size of 512 bits.

Figures 7 and 8 show the fraction of successfully corrected lines as a function of the number of bad bits per 512-bit data line. Figure 7 shows the case where errors are confined to the data bits, and the parity bits are error-free. This repre-

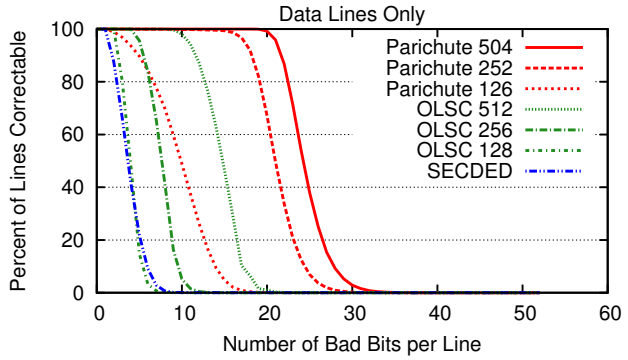

Figure 7: The probability of successful correction versus the number of bit errors per data line, where parity is error-free.

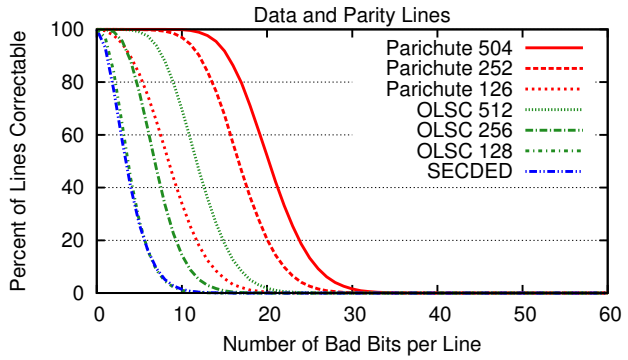

Figure 8: The probability of successful correction versus the number of bit errors per cache line; both data and parity experience errors.

sents scenarios where parity is stored in more protected or less vulnerable media than data. Note that *Parichute* consistently outperforms OLSC for the same number of parity bits.

Figure 8 shows the same experiment for the case when both data and parity bits may be corrupted. Data is assigned to one cache line, while parity is assigned to a portion of another, both with the same number of bad bits per line. Codes with more parity bits will therefore be subjected to more errors. This represents scenarios where data and parity are subject to a similar distribution of errors, as is the case with the *Parichute* cache. In this case, the tolerance for errors is decreased somewhat; however, the relative correction ability of the correction techniques is unchanged.

7.3 Parichute Cache Capacity

We now examine the effect that each error correction technique has on cache capacity at different voltages with different error rates. All correction solutions disable lines that cannot be corrected. For the no-protection case, a line is disabled if it has a single error. Both SECDED and OLSC store their parity bits in cache ways, similarly to *Parichute*.

Figure 9 shows cache capacity versus V_{dd} for all three correction algorithms. For *Parichute* and OLSC, we only show the 252 and 256 cases respectively. For *Parichute*, 252 parity bits out-performs 126 and 504 for maximizing capacity. Although *Parichute* 504 can correct more bad bits, the extra space required for parity offsets the gains. OLSC faces a similar tradeoff. The *Parichute* protected cache has significantly higher capacity than OLSC and SECDED as soon as V_{dd} is lowered sufficiently to cause a large number of failed bits.

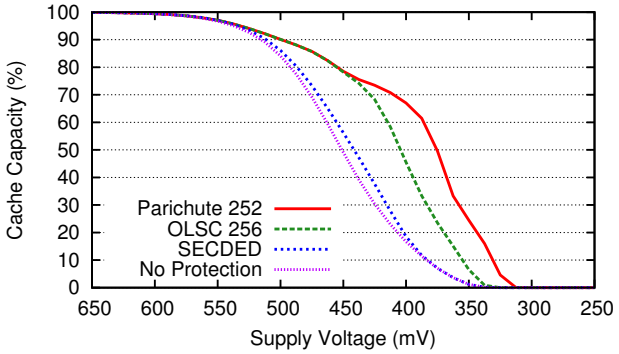

Figure 9: Cache capacity versus supply voltage.

	Cache capacity			*Latency*	
V_{dd}	*Unprot.*	*SECDED*	*OLSC 256*	*Parichute 252*	
900mV	100%	100%	100%	100%	0 cycles
375mV	7.05%	7.06%	23.5%	49.5%	4.11 cycles
350mV	1.22%	1.22%	6.54%	24.5%	3.75 cycles
337.5mV	0.187%	0.187%	0.863%	16.0%	5.74 cycles

Table 3: Cache capacity at nominal and NT supply voltages; average *Parichute* decode latency at nominal and NT supply voltages.

	Capacity at voltage		
Variation awareness	*375mV*	*350mV*	*337.5mV*
None	44%	18%	9%
Good lines	45%	18%	9%
Good & Ugly lines	49%	24%	16%
Good, Ugly, sort Bad lines	50%	24%	16%

Table 4: The effects of increased variation awareness on cache capacity at three voltages in near-threshold.

Table 3 summarizes cache capacity for four V_{dd} values (one nominal, three in near threshold) and shows the corresponding error rates. At 375mV, *Parichute* has 50% of the nominal cache capacity. This is double the cache capacity of OLSC 256 at the same voltage. At 350mV, *Parichute* has about 3.7× the capacity of OLSC, while SECDED leaves the cache virtually useless. Finally, at 337.5mV, only *Parichute* has a usable cache, with 16% of the nominal capacity.

Table 3 also shows average *Parichute* decode latency at nominal and NT voltages. The decode latency is the number of cycles required to correct a line. The latency at the different voltages is averaged across all lines and all cache profiles. At 900mV down to a safe V_{ccmin}, we assume that the cache requires no protection and therefore bypasses the corrector. At NT, the corrector must be used for a large fraction of the lines, and the CRC check must be performed for all lines, incurring a minimum latency of 1 cycle. *Ugly* lines are disabled and do not contribute to line count or decode latency.

7.3.1 Impact of Variation-awareness on Cache Capacity

Table 4 shows cache capacity at three voltages in near-threshold for different levels of variation awareness. No awareness means all lines need protection. Adding the ability to detect *Good* lines allows them to not receive parity protection, saving space and increasing capacity. Detecting *Ugly* lines avoids wasting space on parity for uncorrectable lines. Finally, placing parity in slightly better *Bad* lines improves capacity marginally in moderate error conditions.

	900 mV	*375 mV*			*350 mV*			*337.5 mV*		
Configuration	*All*	*SECDED*	*OLSC*	*Parichute*	*SECDED*	*OLSC*	*Parichute*	*SECDED*	*OLSC*	*Parichute*
Frequency	3000 MHz	463 MHz	463 MHz	463 MHz	355 MHz	355 MHz	355 MHz	305 MHz	305 MHz	305 MHz
L2 capacity	2048 kB	128 kB	512 kB	1024 kB	0 kB	128 kB	512 kB	0 kB	0 kB	256 kB
L2 associativity	8	1	2	4		1	2			1
L2 hit (cycles)	10	11	11	14		11	14			16
DRAM access (cyc)	300	46	46	46	35	35	35	30	30	30

Table 5: Simulation parameters for nominal and near threshold configurations with L2 cache protected by SECDED, OLSC, and *Parichute*.

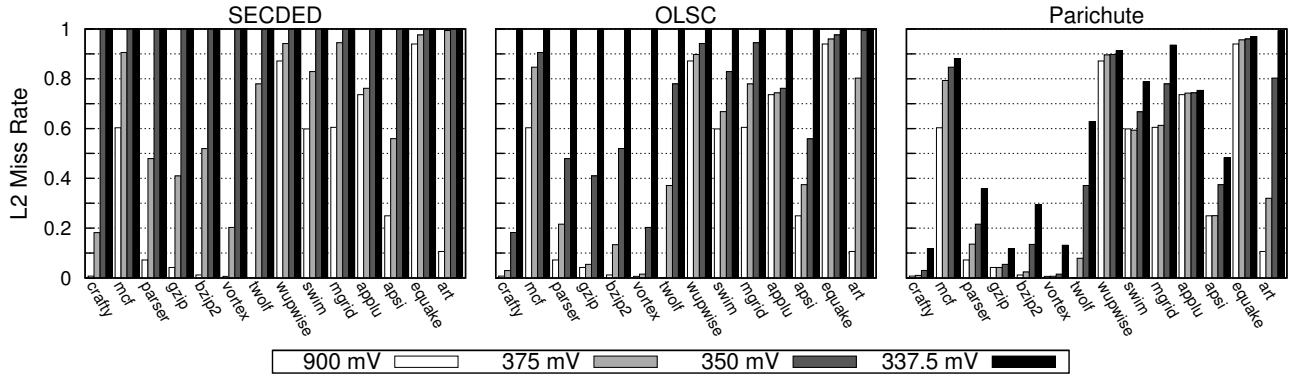

Figure 10: L2 cache miss rates for SECDED, OLSC, and Parichute protected caches in nominal and near threshold configurations.

We find that awareness of *Good* and *Ugly* lines brings the greatest improvement in cache capacity, almost doubling the capacity of a *Parichute* cache at 337mV. Sorting *Bad* lines and assigning parity to the better ones brings only a marginal improvement in capacity, so we do not use this optimization.

7.4 Energy Reduction with Parichute Caches

In this section we evaluate the impact of *Parichute* on performance and energy efficiency of a processor and memory system. We consider all three cache correction algorithms and examine the same three voltage levels in near-threshold. The L2 cache capacity and associativity are scaled according to the effective capacity for each correction algorithm. The average decode latency for *Parichute* at the three voltage levels is factored into the L2 hit time. DRAM access time is assumed to be constant, making relative DRAM latency lower for lower clock frequency. The configurations are summarized in Table 5, along with key simulation parameters.

We first examine the effects of each correction scheme on L2 miss rates (ratio of misses to total accesses). Figure 11 shows the average L2 miss rate over the set of SPEC CPU2000 benchmarks for SECDED, OLSC and *Parichute*. As expected, the higher cache capacity afforded by *Parichute* translates into a significantly lower L2 miss rate. At 350mV, the average L2 miss rate for OLSC is double that of *Parichute*. At the same voltage, SECDED has virtually no cache, so its miss rate is 100% compared to 28% for *Parichute*. Figure 10 shows the breakdown of L2 misses by benchmark, for SECDED, OLSC and *Parichute*.

The ability to lower supply voltage to near threshold and still have a usable cache has a significant impact on energy efficiency. We examine energy required for the entire execution of the benchmarks (average power times execution time). We factor in the power cost of accessing main memory to account for the energy implications of the higher L2 miss rate in near-

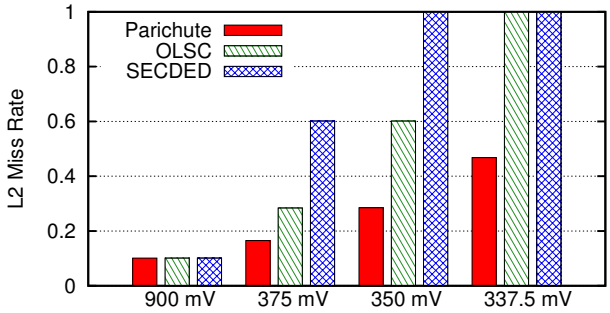

Figure 11: Geometric mean of L2 cache miss rate across all benchmarks, for each error correction scheme, at multiple voltages.

threshold. Figure 12 shows the energy at the three voltage levels in near threshold relative to the energy at nominal for SECDED, OLSC and *Parichute*. Although in near-threshold power consumption is significantly lower, with SECDED protection, there is no reduction in energy. This is caused by the very high number of L2 misses, which increase both execution time and average power due to the higher latency and power consumption of accessing main memory.

OLSC does see a reduction in energy of about 20% at 375mV. However, as V_{dd} is lowered and the L2 cache size decreases rapidly, the energy for OLSC starts to increase again. *Parichute* is significantly more energy-efficient due to its higher cache capacity, in spite of a slightly longer cache access latency. It achieves a 30% energy reduction at 375mV, and, as the voltage is lowered, its energy continues to decrease. At 350mV, *Parichute* achieves a 34% energy reduction compared to nominal V_{dd}, which is roughly 20% better than what can be achieved with OLSC at that voltage. At 337mV, the energy with *Parichute* starts to increase again, but it is still 25% lower than nominal, while OLSC and SECDED are actually 11% higher than nominal.

Figures 12(b) and (c) show the energy behavior for two

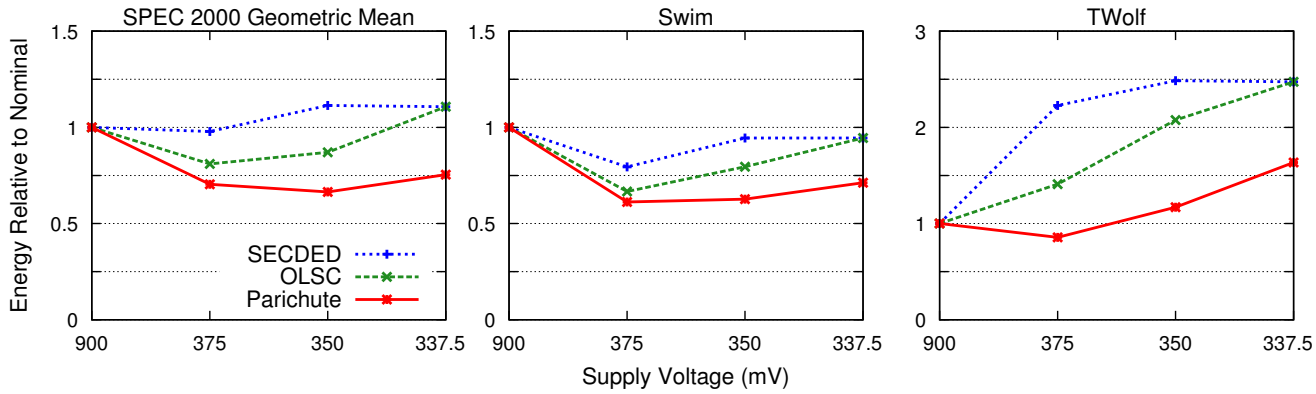

Figure 12: Geometric mean of total energy across all benchmarks relative to nominal (900 mV) and relative energy for *swim*, *twolf* for L2 caches protected by SECDED, OLSC, and *Parichute*.

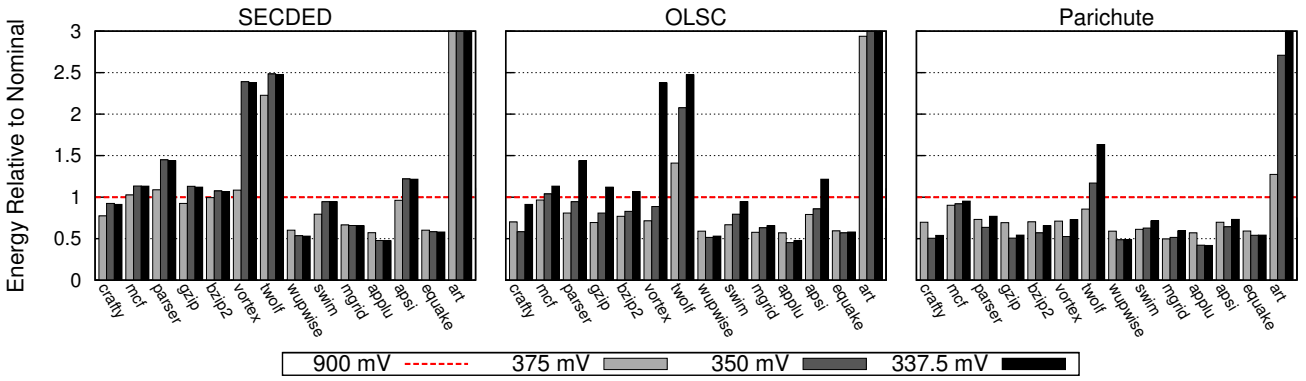

Figure 13: Total energy for each voltage relative to nominal (900 mV) for L2 caches protected by SECDED, OLSC, and *Parichute*.

of the SPEC benchmarks: *swim* and *twolf* respectively. *Swim*'s energy behavior is typical of most benchmarks. For *Parichute*, it shows an energy reduction of about 38% relative to nominal for both 375mV and 350mV cases. At 350mV, energy is also almost 20% lower than OLSC. The difference in energy is due to the lower L2 miss rate with *Parichute*. From Figure 10, we can see that, with OLSC, the L2 miss rate for *swim* increases from 60% in nominal V_{dd} to about 80% at 350mV. With *Parichute*, on the other hand, the miss rate at 350mV is only about 8% higher than at nominal V_{dd}.

Twolf is much more sensitive to the decrease in cache size. Its working set fits very well in the L2 cache at nominal voltage and therefore experiences a negligible miss rate. At lower voltages, the L2 miss rate increases rapidly with OLSC to 80%, while it remains below 10% with *Parichute* at 350mV. This difference in miss rates allows *Parichute* to have a 15% lower energy compared to nominal V_{dd}, while OLSC experiences a 40% increase in energy. As the voltage is lowered further and the L2 miss rate increases significantly, *Parichute* no longer achieves energy savings compared to nominal V_{dd} even though it continues to fare much better than both OLSC and SECDED. Figure 13 shows the total energy relative to nominal V_{dd} for all the benchmarks.

7.5 Overheads

This section presents the area and power overheads obtained from the synthesis of the *Parichute* prototype for 45nm CMOS. We also examine cache tag overhead and the testing time required for the variation-aware cache line classification.

7.5.1 Area and Power Overheads of Parichute Prototype

Table 6 provides a breakdown of the components and subcomponents of the complete *Parichute* implementation. For synthesis, place, and route, timing was constrained to 1ns, and the design successfully met that constraint with a positive slack of 0.05ns. Based on synthesis results, we report standard cell count and die area. The decoder has the greatest overhead, with a cell count roughly 3× that of the encoder.

The area for the entire design is very small at 0.056456 mm^2, which is roughly 0.02% of the area of an Intel Core 2 Duo. Power consumption is also very small at 11mW.

Parichute hardware component	*Synthesized standard cell count*	*Area (μm^2)*
Parichute encoder:	7384	10828
CRC encoder	1938	2802
SECDED encoder block	5436	8149
Parichute decoder:	20244	45628
Parichute corrector:	3419	3739
Syndrome generator	183	251
Slice corrector	177	192
Total	27628	56456

Table 6: Subcomponents and components that make up *Parichute* hardware. A *Parichute* encoder is made up of a CRC encoder and a SECDED encoder block. A *Parichute* corrector is made up of 9 syndrome generators, 9 slice correctors, a CRC checker, and 780 flip-flops. A *Parichute* decoder is made up of 4 *Parichute* correctors.

7.5.2 Cache Tag Overhead

Parichute requires some information to be stored in the cache tags. For an 8MB cache with 64-byte lines and a 48-bit physical address space, the tag array occupies about 5.9% of the cache area. Each tag entry requires 28 address bits to which *Parichute* adds 4 parity location bits (for half-line parity) and 16 CRC bits, for a total of 20 additional bits. While this increases the size of the tag array by 65%, the total cache area is only increased by about 4%. If cache entries are also protected by *Parichute ECC*, 1% is added to the total cache area to store the parity for the tag.

7.5.3 Cache Testing for Variation-awareness

Classifying cache lines as *Good, Bad* or *Ugly* requires testing their correctability. Based on our cache model, we compute an estimate of how long it will take to fully test the entire cache for reliable operation at NT. Assuming that the cache operates error-free at voltages from nominal down to a safe V_{ccmin} [39], proactive error testing is required only at NT voltage levels that will be used. At each voltage, each line must be accessed 4 times (two writes, two reads), and we consider only the three NT voltage levels, 375mV, 350mV and 337.5mV. Based on our model of access time, complete testing requires about 0.06 seconds. Without introducing significant overhead, this testing could be performed during die testing after manufacturing and/or at boot time in the field.

8 Related Work

Prior approaches to cache error correction have generally focused on much lower error rates than those expected in near-threshold. Some existing microprocessors use simple SECDED-based ECC techniques [2, 23, 30], mostly intended to deal with soft errors at high supply voltages.

Researchers have proposed a few cache-based error correction approaches targeted at higher error rates. Sun et al. [34] developed a cache protection solution based on multi-bit ECC (DECTED, Dual Error Correction Triple Error Detection). To reduce the space overhead, they do not maintain parity bits for each cache line but instead maintain a fully associative cache that holds parity information for select lines that are deemed to need protection. The size of the parity cache is fixed, and, as a result, the number of lines that receive DECTED protection is limited. Overall, the error rate that can be tolerated is about 0.5%, significantly below the error rates in near-threshold.

Very recently, Wilkerson et al. [38] developed a technique for coping with embedded DRAM errors that occur as a result of variation-induced cell leakage. They use both SECDED and BCH codes to protect data lines. When a cache line is accessed that has too many errors for SECDED to correct, a high-latency BCH correction is performed. Since this is a rare occurrence, the contribution to average latency is small. While this technique works very well for eDRAM, it would be unsuitable for SRAMs at NT. At NT, the multi-bit error rate is so high that the high-latency BCH correction would add far too much to the average cache access latency.

Yoon et al. [40] devised a method for correcting SRAM errors without storing ECC in dedicated SRAM cells. Instead, ECC bits are stored in cacheable DRAM memory space. In the place of ECC, a much less expensive error *detection* code is stored in dedicated SRAM. This approach is very efficient because only in the rare event that a line is both dirty and suffers an error must the ECC code be fetched from DRAM or elsewhere in the cache. Unfortunately, the very high error rates at NT would impose too high of a demand for DRAM access for this to be applicable to our needs.

Kim et al. [19] propose a 2D encoding scheme in which rows and columns of an SRAM array are simultaneously protected by ECC codes of different strengths. By design, their technique works very well for clustered errors with the ability to correct error rates $> 1\%$ if the bad bits are fully clustered. The technique is less effective if the errors are more evenly distributed. The area overhead for parity storage is fixed and cannot be scaled down in error-free operation.

Other works [9, 14] have proposed circuit solutions for improving the reliability of SRAM cells in near-threshold. They propose replacing the standard 6T SRAM cell with a more error-resilient 8T cell. Our solution relies on error correction and has the advantage that at nominal voltage the *Parichute* cache acts as a regular cache without the power and area overhead of the larger SRAM cells.

9 Conclusions

In this paper we have introduced *Parichute ECC*, a novel forward error correction scheme, and demonstrated its robustness to error rates as high as 7% that occur at near-threshold supply voltages. We have shown that a *Parichute*-protected cache can maintain high storage capacity at error rates that would render less protected caches virtually useless. We have also shown that a system with a *Parichute*-enabled L2 cache can achieve a 34% reduction in energy compared to a system operating at nominal voltage.

References

[1] A. Agarwal, B. Paul, S. Mukhopadhyay, and K. Roy, "Process variation in embedded memories: failure analysis and variation aware architecture," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 9, pp. 1804–1814, September 2005.

[2] H. Ando, K. Seki, S. Sakashita, M. Aihara, Kan, and K. Imada, "Accelerated testing of a 90nm sparc64 v microprocessor for neutron ser," *IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, 2007.

[3] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-Codes," *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261–1271, 1996.

[4] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Design Automation Conference*, June 2003.

[5] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *International Symposium on Computer Architecture*, June 2000.

[6] D. Burnett, J. Higman, A. Hoefler, B. Li, and P. Kuhn, "Variation in natural threshold voltage of NVM circuits due to dopant fluctuations and its impact on reliability," in *International Electron Devices Meeting*, 2002, pp. 529–534.

[7] B. Calhoun and A. Chandrakasan, "A 256-kb 65-nm subthreshold SRAM design for ultra-low-voltage operation," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 3, pp. 680–688, February 2007.

[8] A. Chandrakasan, D. Daly, D. Finchelstein, J. Kwong, Y. Ramadass, M. Sinangil, V. Sze, and N. Verma, "Technologies for ultradynamic voltage scaling," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 191–214, February 2010.

[9] G. K. Chen, D. Blaauw, T. Mudge, D. Sylvester, and N. S. Kim, "Yield-driven near-threshold SRAM design," in *International Conference on Computer-aided Design*, 2007, pp. 660–666.

[10] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages," in *International Symposium on Microarchitecture*, December 2009.

[11] K. Constantinides, O. Mutlu, and T. Austin, "Online design bug detection: RTL analysis, flexible mechanisms, and evaluation," in *International Symposium on Microarchitecture*, November 2008, pp. 282–293.

[12] T. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," *IBM Microelectronics Division Whitepaper*, 1997.

[13] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, February 2010.

[14] R. G. Dreslinski, G. K. Chen, T. Mudge, D. Blaauw, D. Sylvester, and K. Flautner, "Reconfigurable energy efficient near threshold cache architectures," in *International Symposium on Microarchitecture*, December 2008, pp. 459–470.

[15] P. Elias, "Error-free coding," *IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 29–37, 1954.

[16] Y. He and P. Ching, "Performance evaluation of adaptive two-dimensional turbo product codes composed of hamming codes," in *International Conference on Integration Technology*, March 2007, pp. 103–107.

[17] H. Y. Hsiao, D. Bossen, and R. Chien, "Orthogonal latin square codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 390–394, July 1970.

[18] "International Technology Roadmap for Semiconductors (2009)," http://www.itrs.net.

[19] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," in *International Symposium on Microarchitecture*, December 2007, pp. 197–209.

[20] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in *International Conference on Dependable Systems and Networks*, June 2004, pp. 145–154.

[21] D. Markovic, C. Wang, L. Alarcon, T.-T. Liu, and J. Rabaey, "Ultralow-power design in near-threshold region," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 237–252, February 2010.

[22] R. McGowen, C. Poirier, C. Bostak, J. Ignowski, M. Millican, W. Parks, and S. Naffziger, "Power and temperature control on a 90-nm Itanium family processor," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 229–237, January 2006.

[23] J. Mitchell, D. Henderson, and G. Ahrens, "IBM POWER5 processor-based servers: A highly available design for business-critical applications," *IBM Technical Report*, 2006.

[24] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005.

[25] S. Mukhopadhyay, H. Mahmoodi, and K. Roy, "Statistical design and optimization of SRAM cell for yield enhancement," in *International Conference on Computer-aided Design*, Washington, DC, USA, 2004, pp. 10–13.

[26] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Labs, Tech. Rep. HPL-2009-85, 2009.

[27] Nangate, "Nangate open cell library," http://www.nangate.com/.

[28] E. Normand, "Single event upset at ground level," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2742–2750, 1996.

[29] R. Pyndiah, "Near-optimum decoding of product codes: Block turbo codes," *IEEE Transactions on Communications*, vol. 46, no. 8, pp. 1003–1010, 1998.

[30] N. Quach, "High availability and reliability in the Itanium processor," *IEEE Micro*, vol. 20, no. 5, pp. 61–69, 2000.

[31] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, K. Strauss, S. Sarangi, P. Sack, and P. Montesinos, "SESC Simulator," http://sesc.sourceforge.net.

[32] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A model of parameter variation and resulting timing errors for microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, no. 1, February 2008.

[33] L. Spainhower and T. A. Gregg, "IBM S/390 parallel enterprise server G5 fault tolerance: A historical perspective," *IBM Journal of Research and Development*, vol. 43, no. 5, pp. 863–873, 1999.

[34] H. Sun, N. Zheng, and T. Zhang, "Realization of L2 cache defect tolerance using multi-bit ECC," in *Defect and Fault Tolerance of VLSI Systems*, October 2008, pp. 254–262.

[35] Synopsys, "Formality," http://synopsys.com.

[36] Synopsys, "Synopsys design compiler," http://synopsys.com.

[37] J. Torrellas, "Architectures for extreme-scale computing," *IEEE Computer*, vol. 42, pp. 28–35, November 2009.

[38] 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 *International Symposium on Computer Architecture*, June 2010.

[39] 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 *International Symposium on Computer Architecture*, June 2008, pp. 203–214.

[40] D. Yoon and M. Erez, "Memory mapped ECC: Low-cost error protection for last level caches," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 116–127, 2009.

[41] B. Zhai, D. Blaauw, D. Sylvester, and S. Hanson, "A sub-200mV 6T SRAM in 0.13μm CMOS," in *International Solid-State Circuits Conference*, 2007.

[42] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration," in *International Symposium on Quality Electronic Design*, 2006, pp. 585–590.

[43] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *International Symposium on Computer Architecture*, June 2009, pp. 14–23.