Correction Prediction: Reducing Error Correction Latency for On-Chip Memories

Henry Duwe
University of Illinois
at Urbana-Champaign
Email: duweiii2@illinois.edu

Xun Jian
University of Illinois
at Urbana-Champaign
Email: xunjian1@illinois.edu

Rakesh Kumar
University of Illinois
at Urbana-Champaign
Email: rakeshk@illinois.edu

***Abstract*—The reliability of on-chip memories (*e.g.,* caches) determines their minimum operating voltage (V_{min}) and, therefore, the power these memories consume. A strong error correction mechanism can be used to tolerate the increasing memory cell failure rate as supply voltage is reduced. However, strong error correction often incurs a high latency relative to the on-chip memory access time. We propose correction prediction where a fast mechanism predicts the result of strong error correction to hide the long latency of correction. Subsequent pipeline stages execute using the predicted values while the long latency strong error correction attempts to verify the correctness of the predicted values in parallel. We present a simple correction prediction implementation, CP, which uses a fast, but weak error correction mechanism as the correction predictor. Our evaluations for a 32KB 4-way set associative SRAM L1 cache show that the proposed implementation, CP, reduces the average cache access latency by 38%-52% compared to using a strong error correction scheme alone. This reduces the energy of a 2-issue in-order core by 16%-21%.**

I. Introduction

One simple, yet effective, technique to reduce the power of on-chip memories (*e.g.,* caches) is voltage scaling [21], [20], [28], [32], [10]. Reducing the supply voltage results in significant reductions in static and dynamic power [32]. One major challenge of scaling the voltage of on-chip memories is maintaining the desired reliability. Process variations can cause a rapidly increasing fraction of memory cells to become faulty as the supply voltage decreases [32], [12].

A flurry of recent work has been devoted to providing the desired cache reliability at low supply voltages [32], [10], [2], [3], [23]. Some propose using larger, and therefore stronger, memory cells (*e.g.,* 8T and 10T SRAM cells or cells with up-sized transistors) to prevent errors from occurring in the first place. Unfortunately, these methods incur a high static overhead even for nominal voltage operation (see Section VI-A). As a result, many have instead proposed using error correction to correct the wrong values in memory cells as the cells become faulty due to voltage scaling. A large number of error correction techniques have been proposed, spanning from the use of error correction codes [10], [2], [23] to data remapping [32], [3].

However, error correction inevitably incurs a latency overhead, which may be significant relative to the cache access time for error correction strong enough to provide the desired reliability (see Section III). This increase to cache access time due to *strong error correction* may lead to a significant degradation in performance and energy (see Section VII) from either an increased clock cycle time [6] or increased pipeline depth. An alternative supported by some processors is to speculatively execute on the instruction or data accessed from the cache prior to performing error detection [11]. A detected error corresponds to a mis-speculation, which causes the appropriate instructions to be squashed and the pipeline to be stalled for correction. While this technique suffices for scenarios where the bit-failure probability is low, it incurs a high performance overhead for scenarios where the bit-failure probability is high (*e.g.,* when the supply voltage is low) due to rampant mis-speculation leading to frequent squashes and stalls (see Section IV-A).

We propose a novel scheme for hiding the latency overhead of a strong error correction scheme used to ensure reliability. Our scheme, correction prediction, uses a fast mechanism to predict the results of strong error correction. Subsequent pipeline stages execute using the predicted values. In parallel, the long latency strong error correction attempts to verify the correctness of the predicted values. On a mis-prediction, *i.e.,* when the value produced by the correction predictor is not the same as the result of the strong error correction, speculative instructions are squashed and the pipeline is restarted. By allowing the logic core to execute on the predicted data or instructions, one can effectively hide the latency of the slow, strong error correction, even at very low supply voltages where cell failure is prevalent. In the context of hard faults in voltage scaled SRAM L1 caches, we propose implementing the correction predictor using a fast, but weak error correction mechanism that produces the same result as strong error correction mechanisms for most but not all words. Our implementation, CP, is based on a Correction Prediction Table (CPT—details in Section V) that can correctly predict over 90% of cache word corrections.

We make the following contributions:

- We propose correction prediction, a scheme to reduce the latency of strong error correction by predicting its output. Long latency strong error correction verifies the correctness of prediction even as execution continues with the predicted value.
- We present CP, a simple implementation of correction prediction where a fast, weak correction precedes strong error correction and allows CP to limit the mis-prediction rate to <0.1%. CP adds <10% area overhead and <2.5% worst-case latency to a cache with strong error correction.

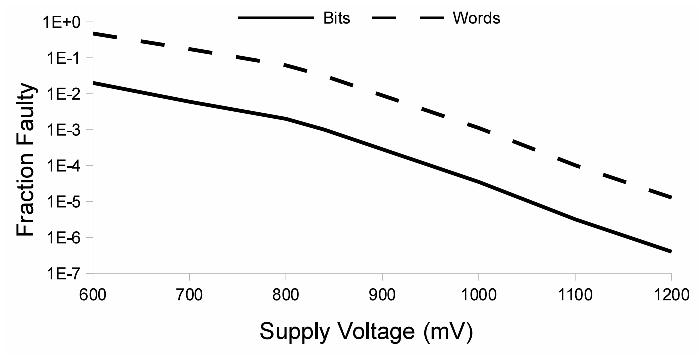


Fig. 1: **The average fraction of bits and 32-bit words that are faulty in a cache for 65nm SRAMs.**

- We evaluate CP applied to three recently proposed strong error correction schemes—Hi-ECC [31], VS-ECC [2], and Bit-Fix [32]. Compared to using the strong error correction technique alone, CP reduces L1 cache access latency by 38%, 38%, and 52%, respectively. For a 2-issue in-order core, this corresponds to a processor-wide energy savings of 16%, 17%, and 21%.

II. Background and Related Work

In this paper we apply correction prediction to SRAM-based L1 caches. Below we discuss the effect of voltage scaling on SRAM cache reliability and prior approaches to provide reliable cache operation.

A. SRAM Reliability at Low Voltages

Some SRAM cells in a cache are weaker than other cells in the cache due to process variations. Although practically every cell in a cache (*e.g.,* 99.999% plus) is functional at a high supply voltage, more and more of these weaker cells become faulty as the supply voltage is reduced. A faulty cell can experience both read failures, where the wrong value is returned or the stored value is toggled unintentionally, and write failures, where the value in the cell cannot be toggled [32]. Since these faulty cells are due to permanent defects (*e.g.,* dopant variation), they can be located using a number of built-in self test (BIST) routines [32], [2], [3]. Some faults at low voltages are due to soft errors [8] that cannot be detected by BIST routines. However, the fraction of such faults at low voltages is minuscule (by over 5 orders of magnitude at 650mV [10]).

Figure 1 shows the average fraction of the cells in a 65nm cache that are faulty as a function of the supply voltage. The calculation is based on the SRAM failure probability reported in [12] and assumes that the faulty cells are distributed randomly across the cache, which is consistent with the assumption made in prior works [32], [10], [2]. We also calculated the fraction of 32-bit words that are faulty; a word is faulty if it contains one or more faulty bits. The results in Figure 1 show that nearly 30% of all words require error correction when the supply voltage is scaled beyond 650mV, motivating the need for efficient error correction algorithms.

B. Error Resilience Techniques In Caches

One technique to improve reliability is using larger transistors to implement the SRAM cells [12]. Another technique is to use a more fault tolerant SRAM implementation, such as an 8T or a 10T SRAM cell [12]. The downside of these techniques is that they incur a significant area and power overhead even when the processor is operating at high supply voltage (see Table II).

There have been several recent attempts at using strong error correcting codes to implement a cache that operates reliably at low supply voltages while incurring a low overhead at a high supply voltage. For example, FLAIR [23] uses a combination of SECDED (single error correction double error detection) codes and dual modular redundancy to correct errors in a cache line. VS-ECC [2] proposes using a combination of SECDED and 4EC5ED (four error correct five error detect). MS-ECC [10] proposes trading off storage-overhead for decode latency by using Orthogonal Latin Square Codes (OLSC) for multi-bit error correction. The downside of these techniques is their high energy overhead at low voltages due to the high performance cost of detection and correction (Section VII).

Finally, several works observe that since the location of the faulty cells can be predetermined via offline testing (*e.g.,* BIST), one can remap the value of faulty cells elsewhere in the cache. PADded Cache [24], for example, uses a fast, programmable address decoder to remap cache lines into other sets and disables faulty physical lines. Unfortunately, for low supply voltage operation, most cache lines would need to be disabled (*e.g.,* >99% of cache lines at 650mV). Bit-Fix [32] proposes a simple remapping policy that uses dedicated bits per cache line to record the bad bits in each cache line and remap their values to a different cache line. Archipelago [3] proposes a more sophisticated remapping policy that uses a global fault map table to perform remapping with greater flexibility. The primary downside of these techniques is the unavoidable latency increase for every cache access due to data correction after data array access or map look-up before the data access. This latency increase may have significant performance and energy impact (Section VII).

C. Tolerating Error Detection and Correction Latency

One simple approach to account for the additional delay in the instruction-fetch and data-load stages due to error correction without stalling the processor pipeline is to slow down the overall core frequency; however, this can lead to a significant performance degradation (Section VII) since the error correction latency is often a significant fraction of the cache access latency. Instead of slowing down the core frequency, Bonnoit et al. [6] propose using additional pipeline stages to handle the error correction latency. However, using pipeline deepening to hide correction latency can also lead to a degradation in performance and energy efficiency (Section VII) because branch and data hazards become more expensive. Also, load-dependent instructions are stalled more frequently and for more cycles. Furthermore, since only low supply voltage operation requires error correction, the added pipeline stages result in unwanted overhead for high voltage operation.

Bonnoit et al. [6] also propose avoiding the additional pipeline stages by decoupling error detection from correction.

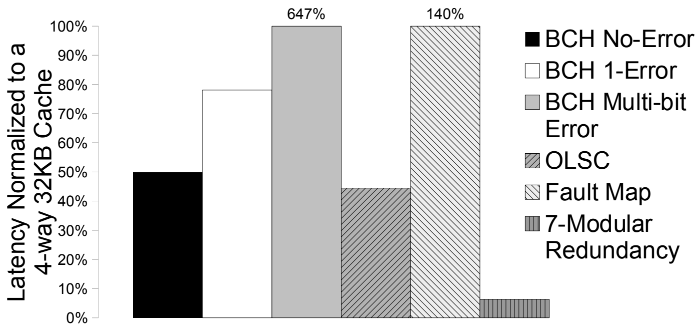


Fig. 2: **Latency of different error correction schemes normalized to the access time of a 4-way set-associative 32KB cache at 65nm.**

They observe that error detection typically incurs a shorter latency than error correction. Therefore, they propose reducing the clock frequency to accommodate only the error detection latency, and then stalling the pipeline when errors are detected to wait for the error correction. However, this technique may result in a significant reduction in clock frequency when error detection latency is still a large fraction of cache access latency. In addition, this technique leads to frequent stalling when the fraction of cache accesses with errors is high, which limits its effectiveness for aggressive voltage scaling.

Some processors [11] attempt to hide the latency of error detection by using speculation, whereby the word retrieved from the cache is sent directly to the subsequent pipeline stages, prior to performing any error detection; meanwhile, error detection takes place in the clock cycles following the cache access. If errors are detected, the pipeline is flushed, an exception handler performs the correction, and execution restarts from the erring instruction. However, this technique is also ineffective at low voltages, where flushing is frequent due to the large fraction of cache accesses with errors (see Figure 1).

III. Motivation

Error correction can be expensive when the number of errors that need to be corrected is large (as may be the case for low V_{min} L1 caches). Consider, for example, 5-bit BCH-based error correction at 650mV for the SRAM failure rates in Figure 1. BCH-based correction has been used for strong error correction in past works such as VS-ECC [2]. We calculate the decode latency of the BCH code for the three scenarios (no error, one error, multi-bit errors) assuming 32 data bits per codeword using the methodology in [27], [31] and using the FO4 delay for the 65nm technology reported in [30] (more modeling details in Section VI-A). Figure 2 shows the latency values normalized to the access latency of a 65nm 4-way set-associative 32KB cache with 64-bit output granularity (see Section VI-A for details). The figure shows that even in the most optimistic scenario (*i.e.,* codeword is error-free), the decode latency of the BCH code is a significant fraction (50%) of the cache access latency. The decode latency is 72% of the cache access latency for single-bit errors and 647% for multi-bit errors.

Other strong error correction techniques that provide comparable reliability are expensive as well. Figure 2 shows the decode latency of a 7-bit error-correcting OLSC code [10], and Bit-Fix, a fault-map-based technique [32]—the reported implementations provide the same reliability as the 5-bit BCH discussed above. Results show that the decode latency of OLSC and the decoding and shifting latencies of Bit-Fix are also significant (41% and 140% of the cache access latency respectively). Moreover, while the error correction latency of the OLSC code is shorter than that of the BCH code, it comes at a significant cost in terms of storage overhead. Similarly, while the error correction latency of 7-modular redundancy is only 6% of that of the cache access, the corresponding storage overhead is 600%.

Our goal is to develop a technique that allows strong error correction to be used for low V_{min} caches without the prohibitive latency or storage overhead. Towards this goal we employ correction prediction used in conjunction with a strong error correction scheme.

IV. Correction Prediction for L1 Caches

Correction prediction for L1 caches feeds predicted values to the pipeline while using strong error correction in parallel. When the predicted value is correct (*i.e.,* the word consumed by the subsequent pipeline stages is the same as the output of the strong error correction), the latency of strong error correction is avoided. On a mis-prediction (*i.e.,* the word consumed by the subsequent pipeline stages differs from the output of the strong error correction), the instructions dependent on the consumed predicted word are flushed and restarted using the output of the strong error correction. The mis-prediction penalty is the larger of the squash/restart and strong error correction penalties.

A. Key Idea

A correction predictor must be both accurate and fast. A correction predictor must have high accuracy to be effective since a large fraction of words are faulty at low voltages (*e.g.,* over 30% at 650mV, see Figure 3). A high mis-prediction rate will lead to frequent squashes. A correction predictor must be fast since even an added cycle of latency may be prohibitive for L1 cache accesses.

A high accuracy correction predictor implementation predicts correctly in high likelihood scenarios. Figure 3 shows that for voltage-scaled SRAMs, a high likelihood scenario is where a word has zero, one, or two faults. An error correction mechanism that can correct up to two errors would correct over 99% of words at 650mV. One fast implementation of such an error correction mechanism is storing information about up to two faults in a table. Since the table cannot store enough information to correct every word and since the table itself can suffer faults, the table cannot correctly predict every access. However, we show in Section V that it can correctly predict over 90% of all accesses. By allowing the pipeline to speculatively execute using instructions or data that have been predicted by the fast error correction mechanism, our proposed error correction implementation, CP, can effectively provide error correction latency similar to that of a fast error correction mechanism with the same level of reliability as a long latency error correction mechanism.

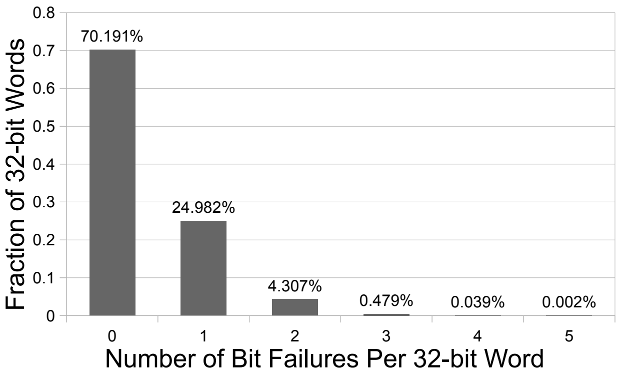


Fig. 3: **Word Error Distribution at 650mV.** 99% of words have two or fewer errors. This suggests that a weak error correction mechanism provides sufficient prediction for most cache accesses.

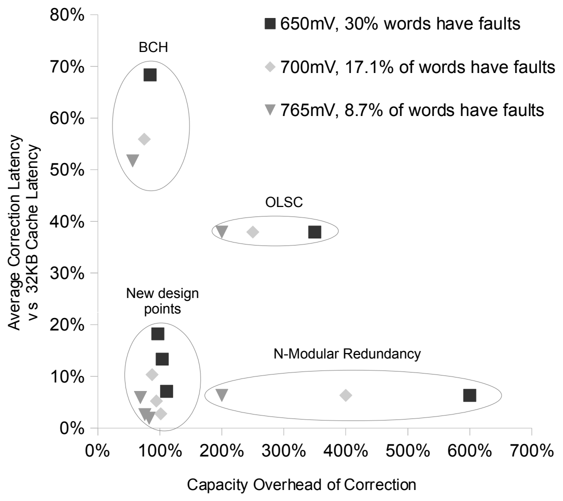
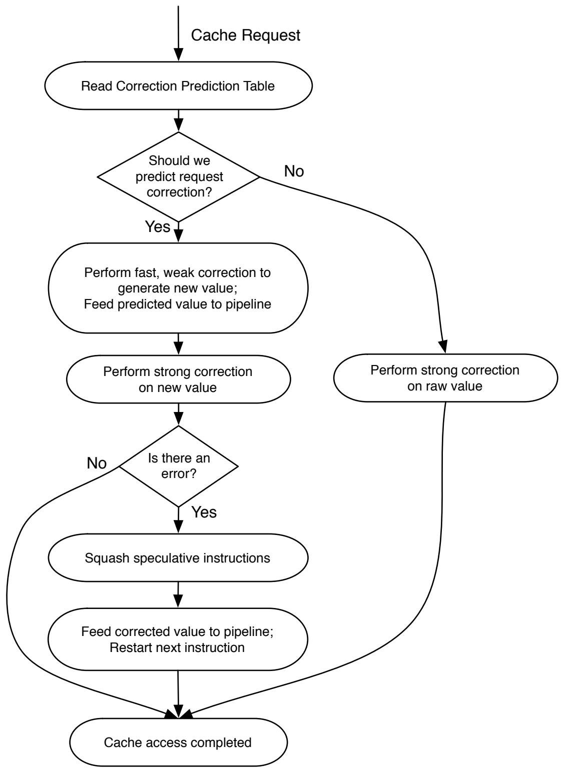


Fig. 4: **Capacity Overhead vs Latency Tradeoff.**

Figure 4 shows the capacity overhead and the average error correction latency of the CP implementation relative to other strong error correction implementations that meet the same reliability target (*i.e.,* implementation can correct 99.9987% of words in the cache for the word error rates shown in the legend[1]). Each data point corresponds to a particular implementation of a correction technique. For the BCH code, the average correction latency is the average of the correction latency for zero errors, one error, and more than one error weighted by the frequency of occurrence of these errors (Figure 3). Each CP design point uses BCH as the strong error correction mechanism while the fast correction mechanism is based on fault-map and can correct up to zero, one, and two errors (corresponds to zero, one, and two Map Units–see Section V). The results show that CP indeed provides latency similar to the low latency correction techniques (*e.g.,* N-modular redundancy) at the capacity overhead of capacity-optimized techniques (*e.g.,* BCH).

[1]This means that each implementation has barely enough ECC resources to provide the same reliability for the voltages shown in the legend as the cache has at 1.2V.



Cache Request
Read Correction Prediction Table
Should we predict request correction?
No
Yes
Perform fast, weak correction to generate new value; Feed predicted value to pipeline
Perform strong correction on new value
Perform strong correction on raw value
Is there an error?
No
Yes
Squash speculative instructions
Feed corrected value to pipeline; Restart next instruction
Cache access completed

Fig. 5: **Correction Prediction for an L1 Cache Access.**

B. Microarchitectural Support

Figure 5 details the logical flow of an L1 cache access using CP. When the pipeline requests an L1 cache access, the cache performs the normal cache tag and data array accesses. In parallel, the cache reads the *Correction Prediction Table* entry (see Section V) corresponding to the word being accessed. The entry indicates whether to perform correction prediction. To predict, the cache applies fast error correction and feeds the resulting predicted value to the pipeline. After speculative execution begins using the predicted value, the slow, strong error correction determines whether the predicted value contains an error. If the predicted value does contain an error, a mis-prediction has occurred and the pipeline squashes all dependent instructions. The cache then returns the corrected value to the pipeline and the pipeline restarts the instruction that initiated the cache request with the correct value. When prediction is not performed, the cache applies strong correction to the value returned from the data array and returns the correct value to the pipeline.

To support CP, the following changes need to be made to the processor pipeline:

Cache Support Figure 6 depicts the additions the CP cache module requires beyond a traditional cache. The fast, weak correction module contains the Correction Prediction Table and associated logic (see Section V). This module determines whether prediction should be triggered and also generates the predicted value. The slow, strong correction module uses the predicted value to provide sufficient error correction to meet the reliability requirement. It outputs both whether it detected an error within the predicted codeword and also returns the

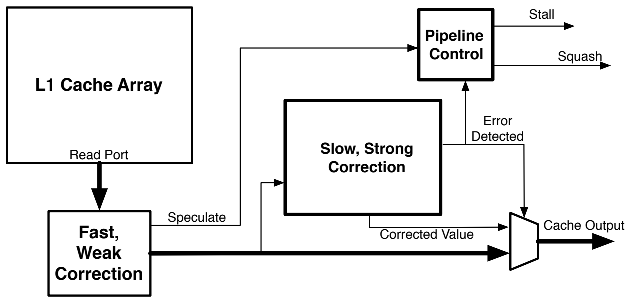


Fig. 6: **Modifications to L1 caches.** The critical path for the common case of correction prediction is in **bold**.

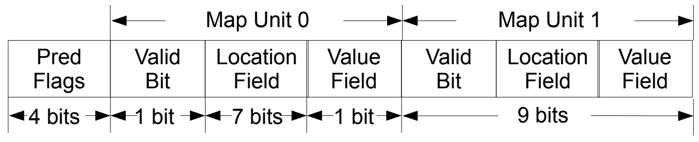


Fig. 7: **Correction Prediction Table Entry Format.** Each table entry corresponds to four 32-bit words.

corrected value (*i.e.,* the corrected data bits from the codeword) in case of an error. If the corrected value does not match the predicted value, then a mis-prediction occurred. The *Pipeline Control* module then indicates that a squash is required. The Pipeline Control module also stalls the pipeline when the fast, weak correction module does not predict a value and the strong error correction module must perform long-latency error correction.

We recognize that the additional logic inserted on the critical path of a cache access may result in a reduction in operating frequency. We quantify this overhead in Section V-B and study the sensitivity of benefits to this overhead in Section VII-C.

Instruction Fetch Applying CP to the instruction cache requires support for squashing each instruction dependent on a mis-predicted instruction and restarting the corrected instruction in the decode stage. Also, the predicted instructions may cause exceptions (*e.g.,* illegal opcode or divide by zero exceptions). Such exceptions must be suppressed until strong correction completes. For the core used in our evaluations (see Section VI), our strong correction scheme requires at least two cycles to detect a mis-prediction, resulting in potentially erroneous instruction bits propagating to the decode, execute, and initial fetch stages. For this core, all exceptions must be suppressed until after the execute stage, which would happen anyway to maintain precise exception handling.

Data Memory Load The core idea behind CP is allowing computation to continue successfully speculating during error correction. For a data memory load, this means that the predicted result must be forwarded to any dependent instructions within the pipeline. This avoids execution of dependent instructions being stalled by the additional latency of error correction. To allow continued forwarding of predicted data to dependent instructions, additional pipeline stages must be added after the data memory stage. The number of required additional stages is equal to the number of cycles it takes to perform strong correction. These additional stages are dummy stages through which instructions flow. These stages also support the forwarding of predicted values to earlier stages. Note that adding these forwarding stages to the data cache access has a significantly smaller impact on performance than adding pipeline stages for error correction (*Deep Pipelining*). This is because the dependent instructions are stalled waiting for the strong correction to complete in the latter case. For the core used in our evaluations (see Section VI), at least two additional pipeline stages are needed after the data memory stage. Mispredictions are detected during the write-back stage. On a misprediction, the value generated by strong error correction is written back to the register file; all following instructions are squashed.

C. Tag Array Protection

Note that the above discussion assumes that correction is performed only on the content of the data array, not the tag array. The tag access is assumed to be robust at low voltages similar to previous related work [3], [32]. Since the tag array is significantly smaller than the data array and is less latency constrained, we assume a 10-T SRAM-based [13] implementation for the tag array to guarantee robustness at low voltages.

V. Implementation, Coverage, and Overheads

For correction prediction to be beneficial, the underlying prediction mechanism must only add minimal latency to a regular cache access. As such, our design goal is to limit the latency of correction prediction to the delay of a single logic gate. Many prediction mechanisms with varying prediction accuracies, latencies, and storage overheads are possible. Here we present one such correction prediction mechanism and leave a full exploration of correction prediction mechanisms as future work.

The proposed correction prediction mechanism is a fast but weak error correction mechanism that duplicates a small number of faulty bits in the cache. The number of duplicate bits is kept small so that they can be stored in a small enough table, called the Correction Prediction Table or CPT, that accessing the table is much faster than accessing the L1 cache. The CPT is accessed in parallel with a L1 cache access. The fast CPT allows the duplicate bits to be accessed and then processed before a L1 cache access completes; as such, it allows the duplicate bits to correct the faulty bits in the cache word at the cost of the delay of a single MUX, which decides what bit—a regular data bit or a duplicate bit—to output per bit position to subsequent pipeline stages.

The CPT is organized as follows. There is a CPT entry corresponding to every consecutive four words (e.g., 128 bits) in the L1 cache. Each CPT entry contains four predFlags and two Map Units (Figure 7). There is a predFlag for each word that allows CP to avoid bad predictions, such as when a CPT entry does not have enough Map Units to correct all faults in the corresponding words or when a Map Unit is faulty. The predFlag indicates to the cache controller whether to perform correction prediction or to simply perform strong error correction and stall the pipeline when the word is accessed. A Map Unit has three fields—*valid*, *location*, and *value*. The

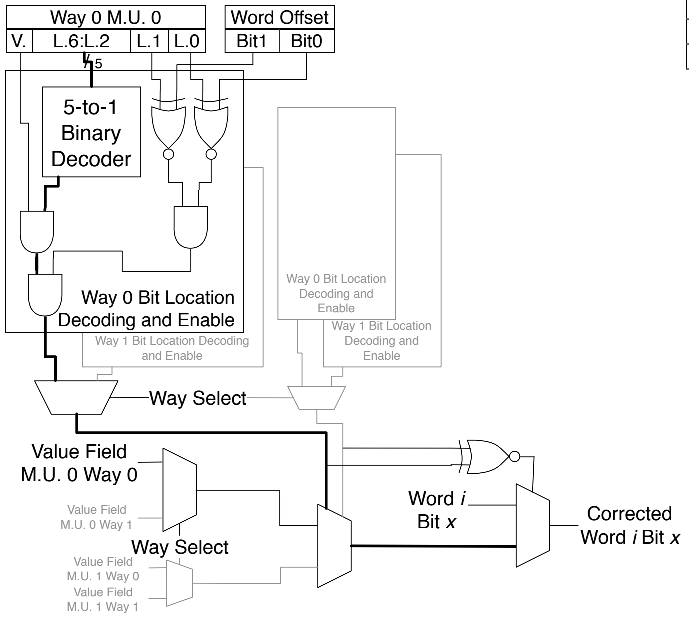


Fig. 8: **Fast, weak error correction circuit for bit x of word i.** *M.U.* stands for Map Unit, *V.* stands for valid bit, and *L.* stands for location field. Value propagation before data array access completes is **bolded**.

valid bit indicates that the corresponding location and value fields are error-free. The location field contains the location of one of the single-bit errors within the 128 bits. The value bit contains the correct current value for the corresponding cache bit. Note that both the valid bits and the predFlags are vulnerable to errors; however, errors in the CPT only affect prediction accuracy, not reliability.

Each CPT entry is populated as follows. Only a CPT entry's value bits are set and updated during regular cache accesses, while all other bits in the CPT are set at runtime by a built-in self test (BIST) routine that tests the L1 cache at the target low voltage. The BIST routine first tests the two Map Units of the CPT entry. If the routine finds any faulty bits in a Map Unit, the routine sets the valid bit of the Map Unit to false. The routine then tests the 128 cache bits that correspond to the CPT entry to identify as many faulty cache bit locations as there are valid Map Units in the CPT entry. These faulty cache bit locations are then recorded in the location field of the valid Map Units. Next, for each of the four predFlags in the CPT entry, the routine sets a predFlag to true if every faulty bit in the 32-bit cache word that corresponds to the predFlag has a corresponding valid Map Unit. Finally, for each write to the cache data array (data write or cache fill), the corresponding CPT entry of written cache word must be read to update the value bits.

Figure 8 shows the fast correction circuitry that uses entries from the CPT to fix an error in a bit of the data word. The location field bits of the two Map Units are decoded to determine which of the 128 bits are replaced by the values stored in the Map Units. The Map Unit valid bit enables this decoding. The least significant two bits of the word offset are used to determine if the Map Unit points to $Word_i$. For

Rate	Derived Formula	Value at 650mV
Prediction	$1-(p\cdot[1-P(error)]+(1-p)\cdot P(error))$	91%
Mis-prediction	$p\cdot P(error)$	0.089%

TABLE I: **Prediction and Mis-prediction Rates.**

every bit in the word accessed from the cache ($Word_iBit_x$), a multiplexer is used to select between the bit read out from the cache and the values stored in the Map Units. The valid bit of a Map Unit enables the selection of its corresponding value field.

A. *Detection and Correction Coverage*

In order to evaluate the fast error correction mechanism we must calculate what fraction of words it attempts to predict (the *prediction rate*) and of those words how many are incorrectly predicted (the *mis-prediction rate*).

To calculate the prediction and mis-prediction rates, we first calculate the probability of observing an error in the output of fast correction. An error may exist in the output of fast correction if the total number of fault bits among the four data words that share the same CPT entry exceeds the number of correct Map Units in the entry. Note that since the Map Units themselves have the same bit error probability as the words in the L1 cache, one or more of the two correction units in a CPT entry may be faulty. To calculate the probability that an error occurs at the output of the fast correction circuitry, we observe that when the total number of faulty bits among the four data words exceed the number of non-fault Map Units by one, an error will appear when one of these four words is accessed. Therefore, the probability that an error occurs when one of these four words is accessed given that the error correction capability of the fast correction entry is exceeded by one is $1/4$. Similarly, the probability that an error will occur when one of these four words are accessed given that the error correction capability of the fast correction entry is exceeded by two is $2/4$. The following equation summarizes the probabilities of all possible scenarios that cause an error in the output of fast correction:

$$P(error)=\sum_{i=0}^{2}\cdot\binom{2}{i}((1-p)^9)^{2-i}(1-(1-p)^9)^i\cdot[\sum_{j=2-i+1}^{128}\binom{128}{j}p^j(1-p)^{128-j}\cdot min((j-2+i)/4,1)] \quad (1)$$

In the formula above, p is the fault probability of a single bit, i is the number of faulty Map Units in the entry and j represents the total number of bad bits in the four words with a total of 128 bits.

To determine the mis-prediction rate, *i.e.,* the probability that the output of fast correction is faulty, but prediction is still triggered, we note that it is equal to the probability that the predFlag bit is faulty when the output of fast correction is faulty. This probability is $p\cdot P(error)$. Prediction is not attempted when the predFlag bit is fault-free while the output of fast correction is faulty, even though it would have been beneficial to trigger prediction. The probability of this occurring is $p\cdot[1-P(error)]$. Prediction is also not attempted when a predFlag bit is faulty while the output of fast correction is not faulty. In this case, the pipeline is correctly stalled until strong correction completes. The probability of this occurring is $(1-p) \cdot P(error)$. Table I lists calculated values for our cache operating at 650mV where the bit failure rate is 0.011 (Figure 1). We note that we could protect the CPT with Schmidt Trigger (ST) SRAM cells or increased supply voltage. However, only 0.089% of cache accesses are mis-predicted, resulting in an insignificant performance degradation. We argue that this is a good tradeoff for not requiring an additional voltage rail or a unnecessary increase in area (*i.e.,* 100%) for the CPT.

We also note that although our specific implementation of fast correction leverages characteristics of the fault distribution (*e.g.,* uncorrelated bit errors where single errors are most common), this is not a requirement of fast error correction. For example, if faults are correlated within a word, we could increase the size of our value fields in the Map Units to improve the correct speculation rate.

B. Latency Overhead

We model the CPT for our 4-way set associative 32KB cache using CACTI [29] to determine its latency. The table has a latency of 0.44ns in a 65nm technology node operating at 1.2V. As shown in Figure 8, the critical path of the fast correction circuit before the MUX that picks between a data bit and a duplicate bit consists of the following: a 5-to-1 binary decoder, 2 AND gates, a MUX to select between the outputs of the fast correction entries that correspond to the two ways in the set, a MUX to select between the two Map Units, and the three level of inverters required for every location to drive 32 bit slices. Using the FO4 delay of 65nm technology reported in [30], these equate to a total delay of 0.2ns. In comparison, the access latency of a 4-way set-associative 32KB L1 cache (the associativity/size of the L1 caches in our evaluations in Section VII) is 0.68ns. Therefore, the delay of fast error correction circuitry prior to the MUX gate can be effectively hidden by the latency difference between the L1 cache and the CPT. Consequently, the MUX gate used to select between the data bit and the duplicate bit is the only additions to the critical path of the cache access. Following the methodology in [14], we estimate the MUX to be 0.5 FO4. In our evaluations, we increase the clock period of the CP cores by 0.01ns at nominal Vdd (1.2V) and 0.026ns at 650mV to account for these delays.

C. Area Overhead

The area overhead of the fast correction technique is dominated by the Correction Prediction Table. We estimated using CACTI [29] that the area of a Correction Prediction Table with two Map Units is 8.6% of the size of a 4-way set associative 32KB L1 cache. Depending on the the desired level of voltage scaling, fewer Map Units may be needed. For design points with only a single Map Unit plus the four predFlag bits or with the four predFlag bits only, the area overhead due to the Correction Prediction Table normalized to cache area would be 5.1% and 1.6%, respectively.

We also estimated the area overhead of the fast correction circuit. As shown in Figure 8, the fast error correction circuit that corrects a single bit of the 32-bit word requires 3 MUXes, eight XOR gates, 12 AND gates, and four 5-to-1 binary decoders, where each requires at most four AND gates, for a total of 40 gates. The total number of gates required by the fast error correction circuit is $40 \cdot 32 = 1240$ gates. Assuming that two such decoders are needed to keep up with the issue width of the processor, this translates to a maximum of 2480 gates, which incurs an area overhead 1.3% compared to a 32KB L1 cache.

D. Energy Overhead

We model the access energy of the CPT using CACTI [29] and determined that an access to the table results in a 4% energy increase for every cache access. We also estimated the energy consumed by the correction circuitry (Figure 7) assuming an activity factor of 1 and the switching energy of a 65nm transistor given in [1]. This energy is 0.2% that of the access energy of the 32KB L1 cache. Static power overheads are no worse than the area overheads discussed in Section V-C.

VI. Methodology

In this section, we describe the methodology we used to evaluate CP for a 65nm technology node. Section VI-A describes the strong error correction baselines to which we apply CP. Section VI-B describes the different designs we evaluated. Section VI-C describes our experimental details.

A. Strong Error Correction

CP can be applied to any strong error correction technique to reduce its latency. In this paper, we use CP in conjunction with three recently proposed strong error correction schemes—Hi-ECC [31], VS-ECC [2], and Bit-Fix [32]. These strong error correction methods vary in their area, capacity, and latency overheads.

Our Hi-ECC implementation protects every word using a BCH code that is capable of correcting five erroneous bits within a 59-bit codeword. We employ an additional parity bit to detect a sixth error within the codeword. If zero or one errors are detected, the evaluated BCH decoder only incurs the single error correction latency, but not the much longer multi-bit correction latency. Single-bit correction is used instead of the multi-bit correction whenever applicable. The multi-bit BCH correction is used if single-bit correction fails. In our implementation, the ECC bits are stored in cache ways during low voltage operation such that half the ways store data bits, while the other half store ECC bits. The ECC cache ways store data during nominal operation as in [10]. Our VS-ECC implementation, based on VS-ECC-fixed from [2] uses seven bits for SECDED per cache word. At the same time, each cache line contains four additional 20-bit extended ECC fields to accommodate a 5EC6ED BCH code for up to four words within the line.

The slow, strong correction module in Figure 6 contains a BCH decoder implemented iteratively according to the modified Berlekamp-Massey Algorithm presented in [2]. With the addition of a modest amount of logic, this decoder can detect errors and correct single bit errors with a small latency relative to the latency of full 5-bit error correction [31]. Using the latency equations from [27] and the 65nm technology parameters from [30], we calculate the error detection latency for the 5EC6ED BCH code to be 0.34ns or 50% of the access

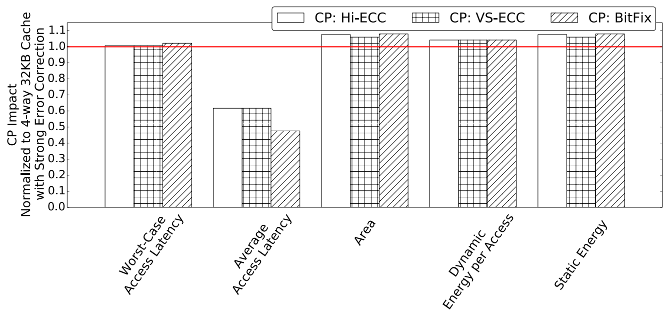


Fig. 9: **Impact of adding CP to different strong correction schemes at 650mV.**

Low Vdd Tolerance Technique	Nominal Voltage		Low Vdd (650mV)		
	Ave Lat Over.	Capacity Over.	Ave Lat Over.	Capacity Over.	Area Over.
CP + Hi-ECC	2%	0%	24%	100%	20%
CP + VS-ECC	2%	0%	24%	0%	50%
CP + BitFix	2%	0%	19%	33%	13%
10T ST Cell SRAM [17]	60%	0%	60%	0%	100%

TABLE II: **Latency, capacity, and area overheads normalized to an unprotected cache.**

latency of a 4-way set-associative 32KB L1 cache [29]. Single-bit correction, as calculated from [31], takes 0.53ns or 78% of an L1 cache access. Similarly, multi-bit correction requires 4.4ns or 648% of an L1 cache access. Given the iterative BCH decoder used in [2] and Schmidt Trigger 10-T cell protection for the tag array, Hi-ECC has a total area overhead of 11.9%. VS-ECC, requiring additional static storage overhead has a 41.4% area overhead compared with our L1 cache.

The energy consumption of the BCH code used by both our Hi-ECC and VS-ECC implementations depends on the number of errors in the input codeword (i.e., zero, one, or more than one). At 650mV, an average of 11 bits are bad per 1000 bits (Figure 1). At this bit error rate, the energy overhead of the BCH decoding is calculated to be 0.86% that of the L1 access assuming an activity factor of 1. The energy overhead of the BCH encoder is calculated to be 0.3% that of an L1 access. Static power overheads are no worse than the previously discussed area overheads.

Our Bit-Fix implementation is adapted from [32]—we assume access at word granularity. Each access takes three cycles [32]. The decoding circuitry has fewer than 26,000 transistors [32] or roughly 1.7% of a 32KB caches data array. Our Bit-Fix implementation also requires the 4.8% area overhead for robust tag cells (10T-ST) resulting in a total area overhead of 6.5% of a 32KB L1 cache. Using an activity factor of 1, our Bit-Fix circuitry has an energy overhead of 1.2% the access energy of an L1 cache. Static power overheads are no worse than the area overheads.

Figure 9 presents the latency, area, and energy impact of applying CP to the above strong error correction schemes. In the worst-case, CP increases the latency of a cache access by up to 2.2% (*e.g.,* when fast, weak correction attempts to predict a word, but strong error correction determines that the predicted word was incorrect). However, as shown in Table I, most errors can be predicted by fast error correction, allowing CP to reduce the *average* latency of a cache access by 38% to 52% depending on the strong error correction scheme. These benefits come at an area overhead of no more than 8%, a maximum dynamic energy overhead of 4.2%, and a maximum static energy overhead of 8% at low Vdd.

Table II compares the latency, capacity, and area overheads of complete CP schemes (including overheads from both CP and the specific strong error correction scheme) to those of a strong circuit-level technique. At nominal voltage, all CP schemes have significantly smaller latency and area overheads compared to a 10T ST SRAM cell. The 10T ST SRAM [16] has significantly better reliability at low voltage than 8T [9] and 10T [7], yet it still cannot provide sufficient reliability for a 32KB cache compared with a strong error correction technique such as Bit-Fix [32].

B. Design Points

As shown in Figure 9, applying CP reduces the average access latency of a reliable, low Vdd L1 cache. At the processor level, this can result in significant performance and energy benefits for those processors (*e.g.,* in-order cores) where the latency of cache access can significantly determine performance. To quantify the processor benefits, we evaluate CP against three design points presented in Table III. The first baseline, *Nominal Baseline*, is a 7-stage pipeline running at 2.68GHz at 1.2V. Note that cache accesses take two cycles for this baseline. The second baseline, *Deep Pipe*, has additional stages (for both the L1 ICache and L1 DCache) to accommodate the latency of error correction required during 650mV operation without affecting the frequency of the pipeline during nominal (1.2V) operation. For Hi-ECC and VS-ECC, error correction latency requires two cycles given the single-bit correction latency described in Section VI-A, while Bit-Fix requires three cycles. The frequency of the Deep Pipe baseline (and other baselines) at 650mV is determined using the operating point pairs from [32] and assuming a linear voltage-frequency scaling (this is similar to the methodology used in [18], [2]).

The third design point is Speculate on Every Access (*SEA*) where every cache access is speculated upon. *I.e.,* the uncorrected value is returned to the pipeline and to the strong correction circuitry at the same time. This design point represents the natural extension of [11] using hardware error correction. This design will suffer from frequent squashing of speculative instructions and more frequent stalls for long-latency correction. SEA also requires the additional pipeline stages that CP needs for forwarding speculative values. A fourth design point is *SECDED+Disable* where each word is protected by an ECC that can correct up to one error (this is the same SEC as used for our VS-ECC implementations). If more than one error is identified, that word is disabled, requiring an access to the next level in the cache hierarchy. Since SECDED requires more than one cycle of additional latency, implement it in an 11-stage pipeline. The final design point is *CP* . It has a slightly lower frequency at nominal voltage than the other

Parameter		**Nominal Baseline**	**Deep Pipe**	**SEA**	**CP**
Processor Frequency	Low Vdd	—	968 MHz	968 MHz	945 MHz
	Nom Vdd	2.68 GHz			2.56 GHz
L1 $ Latency	Low Vdd	—	2.37 ns		
	Nom Vdd	0.68 ns			
L1 Singlebit Correction Latency	Hi-ECC	—	2 cycles		
	VS-ECC	—	2 cycles		
	BitFix	—	3 cycles		
L1 Multibit Correction Latency	Hi-ECC	—	13 cycles		
	VS-ECC	—	13 cycles		
	Bit-Fix	—	3 cycles		
Pipeline Stages	Hi-ECC	7	11	9	9
	VS-ECC	7	11	9	9
	Bit-Fix	7	13	10	10
L2 Latency		10 ns			

TABLE III: **Operating Point Parameters.**

design points and a slightly lower frequency than Deep Pipe and SEA at 650mV to account for the addition of two MUXes to the critical path of a cache access (see Figure 8).

C. Experimental Setup

Number Cores		1 in-order
Register File		32 Int, 32 FP
Fetch/Decode/Issue Width		2/2/2
BTB Size		4096 entries
RAS Size		16 entries
Branch Predictor		Tournament
ALUs/FPUs/MDUs		2/1/1
Cache Line Size		64B
L1 I$	Nominal (1.2V, all designs)	32KB, 4-way
	Hi-ECC (650-840mV)	16KB, 2-way
	VS-ECC (650-840mV)	32KB, 4-way
	Bit-Fix (650-840mV)	24KB, 3-way
L1 D$	Nominal (1.2V, all designs)	32KB, 4-way
	Hi-ECC (650-840mV)	16KB, 2-way
	VS-ECC (650-840mV)	32KB, 4-way
	Bit-Fix (650-840mV)	24KB, 3-way
L2 Unified $		1MB, 8-way
Memory Configuration		2 GB of 1066 MHz DDR3

TABLE IV: **Basic core characteristics.**

We evaluate CP over benchmarks from the Spec2000 [25] and Spec2006 [26] benchmark suites executing on a 2-issue in-order core. Core microarchitectural parameters were chosen to be similar to the ARM Cortex-A7 [4] and are enumerated in Table IV. An aggressive branch predictor was chosen to not unduly penalize *Deep Pipe*. Performance simulations were run using Gem5 [5], fast forwarding for 1 billion instructions and executing for 1 billion instructions. Frequency for a given voltage was determined by the linear scaling in [2]. Nominal dynamic and static power overheads were determined using the simulation results and McPAT [19]. Low supply voltage dynamic power was scaled quadratically with respect to voltage [32]. Low supply voltage static power was scaled cubically with respect to voltage [32]. The L2 and the main memory are not scaled (*i.e.,* we model them to have an absolute latency in terms of ns).

VII. Experimental Results

In this section, we evaluate the impact of the reduced error correction latency enabled by CP on processor-level performance, power, and energy.

A. Low Voltage (650mV) Operation

Figure 10 shows the performance of the different schemes at 650mV compared to the nominal baseline operating at 1.2V. For simplicity, we discuss Hi-ECC results here, although the trends for VS-ECC and Bit-Fix are similar. We see a 22% performance benefit over the deepened pipeline design in spite of slightly lower frequency (945MHz vs 968MHz). This is due to a reduced penalty for branch and data hazards since the number of pipeline stages is smaller (9 vs 11). Furthermore, the additional pipeline stages in CP continue to forward to dependent instructions, while the additional pipeline stages in the deepened pipeline design lead to stalls for the load-dependent instructions. CP increases performance by 42% over the SECDED with word disable scheme, which performs the worst of all correction schemes on average. When compared to Speculate on Every Access (SEA), CP increases performance by 18%. There are two reasons for this performance improvement. First, CP reduces the mis-prediction rate and thus the penalty from squashing speculative instructions. Second, CP reduces the number of long-latency stalls for strong error correction because the fast, weak correction in CP can correct errors within the data word, which may reduce multi-bit errors to single-bit errors. Finally, CP achieves average performance within 13% of the ideal, zero-latency, zero-capacity correction. The slight difference in performance is due to the slightly lower frequency of operation at 650mV, stalls due to non-predictions and mis-predictions (see Table I). When compared against the SECDED+Disable design point, CP has a performance benefit of 42% because SECDED requires additional pipeline stages for both L1 caches and also incurs long-latency misses to the next levels of the cache hierarchy.

Figure 11 shows the energy benefits that the error correction prediction scheme can provide over the reduced frequency and pipeline deepening schemes. The observed benefit of CP compared to the deepened pipeline design is 19% and it occurs due to the 22% higher performance. CP has 16% lower energy than SEA because of the increased performance from correct prediction and decreased power consumption. All error correction techniques show a reduction of average energy by 58% or more relative to the nominal baseline. We report energy benefits for a core where logic and SRAMs are on a single power rail. If logic and SRAMs are on different rails, energy benefits will still be significant because SRAM energy would dominate at low supply voltages.

B. Nominal Operation

We recognize that for many applications low power/energy operating points are desirable, but they must not come at the price of hurting performance at nominal operating points. We evaluated the performance of each design point at the nominal voltage (1.2V). Figure 12 shows that CP has only a slight performance degradation of 1.2% when operating at the nominal voltage due to the addition of a MUX on the critical path of L1 cache accesses. This shows that CP is not only an attractive design point for fixed voltage systems running at low voltages (*e.g.,* 650mV), but also for systems that support DVFS to provide high performance at nominal operating points. In contrast, pipeline deepening has an average

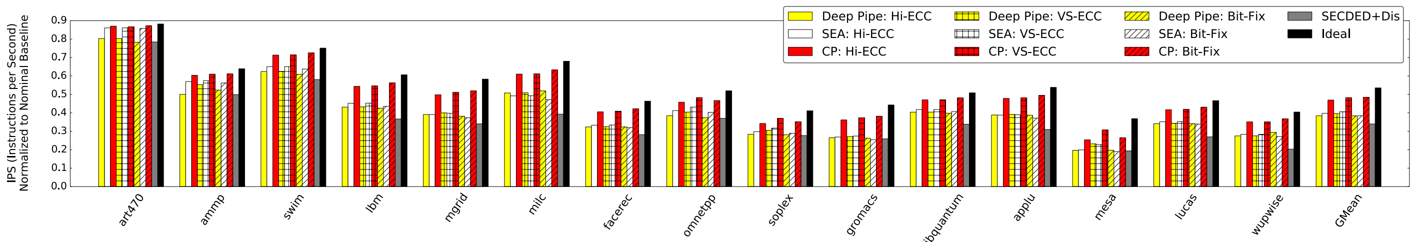


Fig. 10: **Performance for 650mV Operation.**

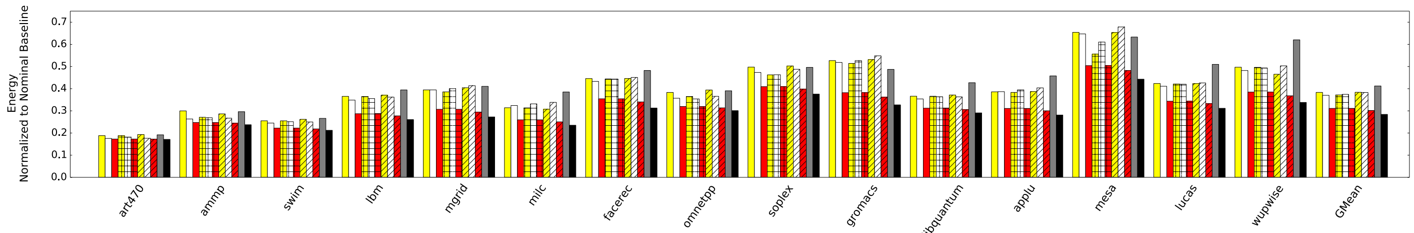


Fig. 11: **Energy for 650mV Operation.**

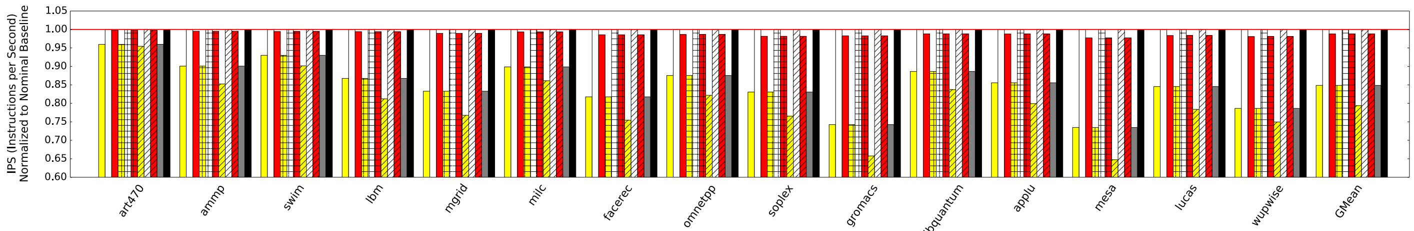


Fig. 12: **Performance for Nominal Voltage Operation.**

performance degradation of over 15%. The high performance degradation is due to two factors (1) data and control hazards are more expensive because of increased number of pipeline stages, and (2) since the additional pipeline stages corresponding to the data cache accesses cannot forward values until the access is complete, dependent instructions have to stall. Finally, the reduced frequency design and SEA have high performance at nominal operation. However, these designs may have significant performance and energy overhead at low voltage operation (Figure 10, Figure 11).

C. *Sensitivity*

The benefits provided by CP are dependent on several parameters, including the voltage of operation, the latency impact of CP, and the fault rates of a given voltage. In this section we perform a sensitivity analysis of these parameters.

Our results in Section VII-A were presented for 650mV. It is the lowest voltage at which all the ECC bits for a 32-bit data word can fit within another 32-bit word. In this section, we compare the performance of correction prediction against other design points over higher supply voltages and corresponding error rates. For Hi-ECC and VS-ECC, we select the weakest BCH code which guarantees reliability at that voltage equivalent to the reliability at nominal voltage. This means that the given supply voltage is the lowest supply voltage at which the design can be run (*e.g.,* the BCH decoder that is required for 840mV operation would not be sufficient to operate at 700mV). The CP error correction hardware is constant across all supply voltages.

Figure 13 shows the results. The results show that CP allows operation over a large voltage range at a performance comparable to the ideal scheme. Performance degradation of CP applied on top of Hi-ECC compared to the ideal is only 13%, 7%, and 7% at 650mV, 700mV, and 840mV respectively. The primary reason why CP performs comparably to the ideal is that a large fraction of errors over a wide voltage range are predictable by the fast, weak correction. We also observe that while CP provides better performance than other design points over the entire range that was evaluated, the performance advantage depends on the operating voltage. For example, while CP has nearly identical performance to SEA at 840mV, the performance advantage increases to 18% at 650mV. This is because the number of errors is considerably higher at 650mV than 840mV. For SEA, this leads to significantly higher number of squashes and long-latency stalls. For CP, most of the cache accesses continue to be predicted by the fast, weak correction scheme at 650mV. Finally, CP continues to perform much better than reduced frequency and pipeline deepened designs even at higher supply voltages (*e.g.,* 840mV). This is because the reduced frequency design has significantly lower frequency even at 840mV (1.111GHz vs 1.524GHz). Similarly, the deepened pipeline not only suffers from the standard disadvantages of having a large number of pipeline stages (increased cost of branch and data hazards), but also has additional stalls due to the load-dependent instructions waiting for the error correction for the data cache access to complete.

The results in Section VII-A correspond to a 2.5% latency overhead for CP as modeled by the methodology in Section V-B. Figure 14 shows the performance of CP applied

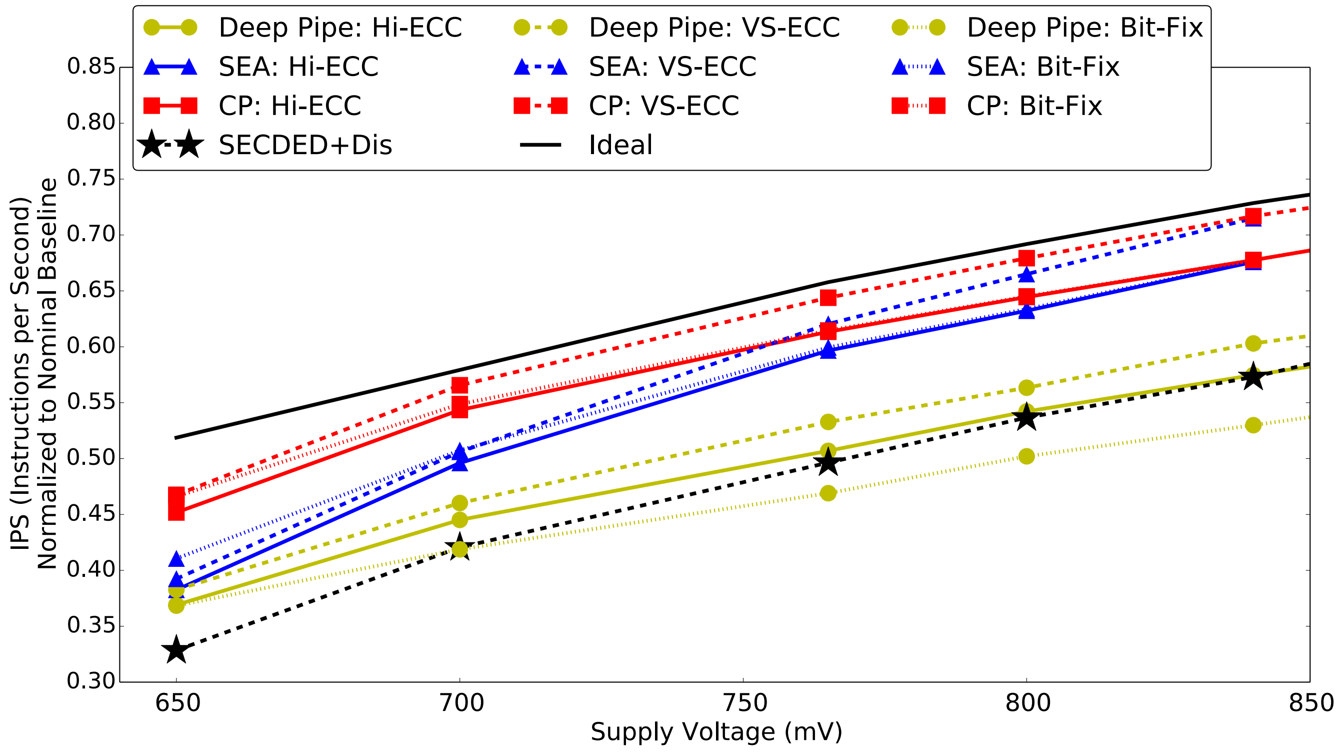


Fig. 13: **Voltage Scaling Sensitivity Study.**

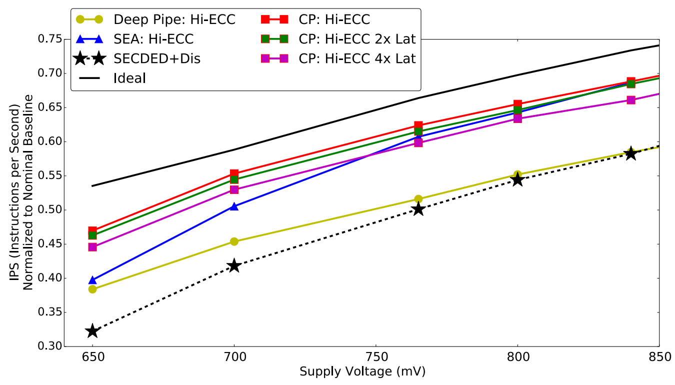


Fig. 14: **Fast Correction Table Latency Sensitivity Study.**

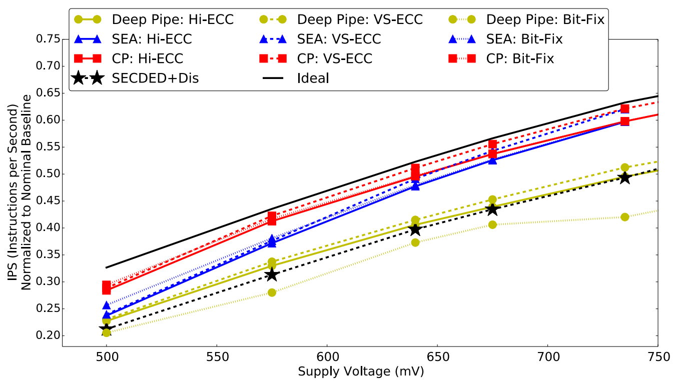


Fig. 15: **Fault Rate Sensitivity Study.**

to Hi-ECC (the correction scheme CP performs worst with) assuming its latency overhead is double and quadruple our calculated value. Despite the increased overhead (5% and 10%), CP still increases performance by 16% and 12%, respectively.

Our results in Section VII-A are based on the 65nm fault rate data from [12] (note that prior work [2], [32] used 130nm fault rate data). We explored the sensitivity of benefits to fault rates nearly 5x lower than [12]. Figure 15 shows that CP can still provide performance improvements of 14-20% for a lower fault rate vs voltage curve. These results (including results in Figures 13, 14, and 15) attest to the generality of our approach.

The results reported above are for min-sized 6T SRAMs. Prior work (*e.g.,* [17]) has demonstrated that circuit-level techniques such as doubling the size of transistors in 6T cells or using 8T cells may allow reduced voltage operation. However, CP can still yield performance benefits when used with such circuit-level techniques. To quantify the benefits we rely on the fault rate vs voltage dataset from [17] and voltage versus frequency scaling dataset from [15].[2] 6Tx2 SRAM cells allow operation at 454mV with the evaluated strong error correction techniques. When implemented on top of 6Tx2 SRAM cells, CP can achieve a performance improvement of up to 13-20% as shown in Figure 16. Such upsizing results

[2]We use fault rate vs voltage data from [17] since the data set used for our main set of results in Section VII-A did not have fault rate vs voltage data for 6T upsized or 8T SRAM cells. We use the non-linear voltage vs frequency scaling data from [15] since the linear voltage vs frequency scaling assumption no longer applies for the low voltages allowed by 6Tx2 and 8T cells.

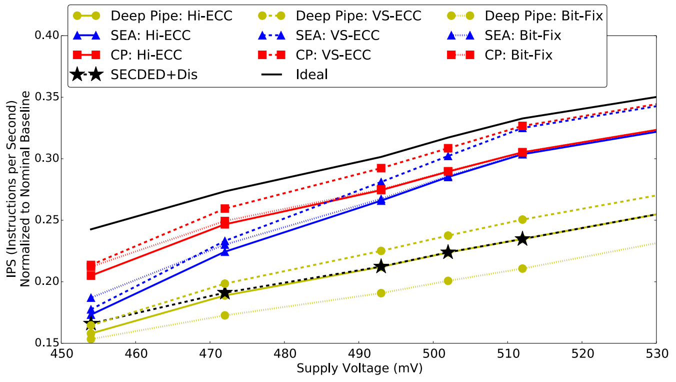


Fig. 16: **6T 2x Upsized Sensitivity Study using [17], [15].**

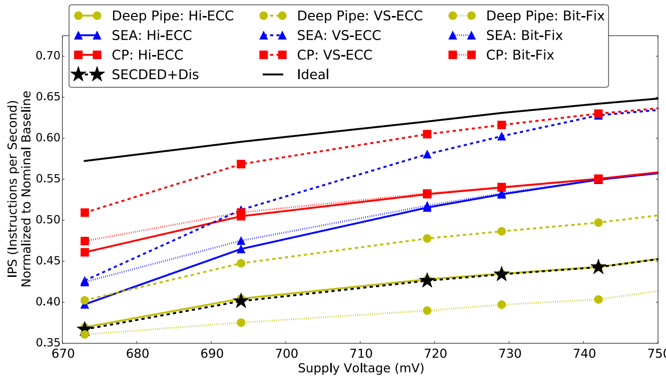


Fig. 18: **6T Sensitivity Study using [17], [15].**

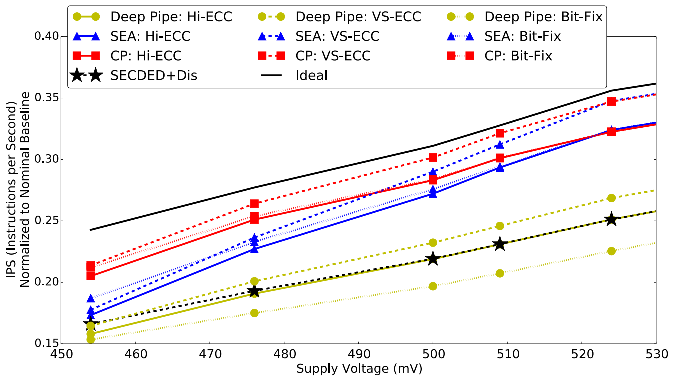


Fig. 17: **8T Sensitivity Study using [17], [15].**

in a 33% area increase and a doubling of static power. By using 8T SRAM cells, the additional static power can be reduced, while still allowing a minimum voltage of 454mV. When implemented on top of 8T SRAM cells, CP provides up to a 13-20% performance improvement as shown in Figure 17. For completeness, Figure 18 shows CP results when applied to 6T cells using these datasets. At the lowest voltage where the reliability target can be met, CP shows a performance benefit of 12-19% for 6T transistors, similar to the primary results presented in Section VII-A.

VIII. Future Work and Discussions

In this paper, we explore one implementation of correction prediction; many other implementations with different tradeoffs exist. For example, instead of using a BIST routine to populate the CPT as described in the paper, one can use a learning mechanism to populate the CPT by dynamically identifying the weak memory cells at runtime. Also as an example, instead of allocating a static CPT entry to every group of four words as described in the paper, one can also use a smaller CPT table with fewer *dynamic* entries that correspond to the most frequently accessed words. Finally, other implementations of correction predictors are possible, such as weak error-correcting codes or history prediction.

While our evaluation is performed in the context of an L1 cache, the concept of correction prediction is applicable to other on-chip memories. Examples include L2 and L3 caches, as well as non-cache on-chip memories such as GPU register files and the memory systems of embedded processors which often reside on-chip. Correction prediction is also applicable to emerging technologies such as STT-RAMs, where fault rates are high [22]. In addition to in-order cores, correction prediction can be applied to other processor micro-architectures that support speculative execution, such as out-of-order processors and processors with runahead threads. The details needed to apply correction prediction to these different contexts are outside the scope of this paper.

Finally, correction prediction bears some resemblance to value prediction, which seeks to predict the value of a load before the load instruction completes. They differ in three main ways. First, value prediction is beneficial only for accesses to words with value locality, while correction prediction is not limited by this requirement. Second, value prediction benefits only load instructions, while correction prediction benefits all instructions by predicting the correct values of instruction words accessed from the I-cache. Third, by predicting the values of weak cells in a word, instead of predicting the complete value of the whole word, correction prediction can require significantly lower overheads than value correction while providing the same prediction accuracy.

IX. Conclusion

On-chip memories consume an increasingly large fraction of chip power. The reliability of on-chip memories determines their voltage and, therefore, the power these memories consume. Voltage scaling can be used to significantly reduce the power consumed by on-chip memories and chips as a whole. However, aggressive voltage scaling leads to high error rates in on-chip memories (*e.g.,* caches). Strong error correction can be used to tolerate high error rates in on-chip memories. However, such strong error correction may require significant latency relative to the memory access itself. We propose correction prediction, a scheme that reduces the latency of strong error correction by using a fast mechanism to predict the result of strong error correction. We present CP, a fast, weak correction mechanism that predicts the result of strong error correction with a mis-prediction rate of <0.1%. This reduces the effective access latency of a 32KB, 4-way SRAM L1 cache by 38%-52%. For a 2-issue in-order core, CP provides 16%-21% energy reduction compared with using a strong error correction scheme alone, while incurring less than a 1.2% performance degradation at nominal voltage.

Acknowledgment

This work was supported in part by Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA. The authors would like to thank Janak Patel and the anonymous reviewers for their helpful feedback.

References

[1] "International technology roadmap for semiconductors 2001 edition process integration, devices, and structures and emerging research devices." [Online]. Available: http://www.itrs.net/Links/2001ITRS/PIDS.pdf

[2] 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 *Proceedings of the 38th annual international symposium on Computer architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 461–472.

[3] A. Ansari, S. Feng, S. Gupta, and S. A. Mahlke, "Archipelago: A polymorphic cache design for enabling robust near-threshold operation." in *HPCA*. IEEE Computer Society, 2011, pp. 539–550. Available: http://dblp.uni-trier.de/db/conf/hpca/hpca2011.html#AnsariFGM11

[4] ARM, "Cortex-a7 technical reference manual, rev r0p5." Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0464f/index.html

[5] 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*, vol. 39, no. 2, pp. 1–7, Aug. 2011. Available: http://doi.acm.org/10.1145/2024716.2024718

[6] T. Bonnoit, M. Nicolaidis, and N.-E. Zergainoh, "Using error correcting codes without speed penalty in embedded memories: Algorithm, implementation and case study," *Journal of Electronic Testing*, vol. 29, no. 3, pp. 383–400, 2013. Available: http://dx.doi.org/10.1007/s10836-013-5386-8

[7] B. Calhoun and A. Chandrakasan, "A 256kb sub-threshold sram in 65nm cmos," in *Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, Feb 2006, pp. 2592–2601.

[8] V. Chandra and R. Aitken, "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos," in *Proceedings of the 2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, ser. DFT '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 114–122. Available: http://dx.doi.org/10.1109/DFT.2008.50

[9] L. Chang, D. Fried, J. Hergenrother, J. Sleight, R. Dennard, R. Montoye, L. Sekaric, S. McNab, A. Topol, C. Adams, K. Guarini, and W. Haensch, "Stable sram cell design for the 32 nm node and beyond," in *VLSI Technology, 2005. Digest of Technical Papers. 2005 Symposium on*, June 2005, pp. 128–129.

[10] Z. Chishti, A. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, 2009, pp. 89–99.

[11] *Alpha 21264 Microprocessor Hardware Reference Manual*, Compaq Computer Corporation. Available: http://h18000.www1.hp.com/cpq-alphaserver/technology/literature/21264hrm.pdf

[12] 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, 2010.

[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, 2010.

[14] K. Khubaib, M. Suleman, M. Hashemi, C. Wilkerson, and Y. Patt, "Morphcore: An energy-efficient microarchitecture for high performance ilp and high throughput tlp," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, 2012, pp. 305–316.

[15] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, Feb 2008, pp. 123–134.

[16] J. Kulkarni, K. Kim, and K. Roy, "A 160 mv robust schmitt trigger based subthreshold sram," *Solid-State Circuits, IEEE Journal of*, vol. 42, no. 10, pp. 2303–2313, Oct 2007.

[17] J. Kulkarni and K. Roy, "Ultralow-voltage process-variation-tolerant schmitt-trigger-based sram design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 2, pp. 319–332, Feb 2012.

[18] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, ser. HotPower'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8. Available: http://dl.acm.org/citation.cfm?id=1924920.1924921

[19] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 469–480. Available: http://doi.acm.org/10.1145/1669112.1669172

[20] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable power control for many-core architectures running multi-threaded applications," in *Proceedings of the 38th annual international symposium on Computer architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 449–460. Available: http://doi.acm.org/10.1145/2000064.2000117

[21] K. Meng, R. Joseph, R. P. Dick, and L. Shang, "Multi-optimization power management for chip multiprocessors," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 177–186. Available: http://doi.acm.org/10.1145/1454115.1454141

[22] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, *Intel Technology Journal*, vol. 17, no. 1, 2013. Available: http://www.csit-sun.pub.ro/~cpop/Documentatie_SM/Intel_Microprocessor_Systems/Intel%20TechnologyNew/intelr_technology_journal__volume_17_issue_1_2013.pdf#page=54

[23] M. K. Qureshi and Z. Chishti, "Operating secded-based caches at ultra-low voltage with flair," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, 2013, pp. 1–11.

[24] P. P. Shirvani and E. J. McCluskey, "Padded cache: a new fault-tolerance technique for cache memories," in *VLSI Test Symposium, 1999. Proceedings. 17th IEEE*. IEEE, 1999, pp. 440–445.

[25] Standard Performance Evaluation Corporation, "Spec cpu2000." Available: www.spec.org/cpu2000

[26] Standard Performance Evaluation Corporation, "Spec cpu2006." Available: www.spec.org/cpu2006

[27] D. Strukov, "The area and latency tradeoffs of binary bit-parallel bch decoders for prospective nanoelectronic memories," in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, 2006, pp. 1183–1187.

[28] M. B. Taylor, "Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 1131–1136. Available: http://doi.acm.org/10.1145/2228360.2228567

[29] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," *HP Laboratories, Palo Alto, Tech. Rep*, vol. 20, 2008.

[30] N. H. West and D. M. Harris, "Cmos vlsi design: A circuits and systems perspective," 2010, p. 312.

[31] 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," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 83–93, Jun. 2010.

[32] 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 *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 203–214.