Maximizing Energy-Efficiency through Joint Optimization of L1 Write Policy, SRAM Design, and Error Protection

Brian Zimmer
bmzimmer@eecs.berkeley.edu

Michael Zimmer
mzimmer@eecs.berkeley.edu

Abstract

L1 cache design contributes significantly to the performance and energy consumption of microprocessors due to L1's large proportion of die size and high activity factor. Voltage reduction reduces energy per operation, however, increased process variability in modern technology nodes causes an exponential growth in SRAM failure probability for linear voltage reduction, necessitating some form of error correction. We compare the implications on energy, area, performance, and error rate for two methods of error correction—a write-back cache with SEC-DED and a write-through cache with parity detection—and identify system factors (such as cache size and latency) that define the optimal solution. Additionally, we explore potential benefits of running a processor at finite error rates (aggressive voltage scaling) by analyzing recovery costs and system energy. Energy measurements come from full transistor-level simulation of functional 28nm SRAM designs over varying operating conditions, and hit/miss rate measurements come from a cache simulator embedded into the RISC-V ISA simulator.

1 Introduction

Dynamic Voltage and Frequency Scaling (DVFS) allows processors to dynamically trade-off energy for delay. In mobile applications, DVFS allows processors to minimize energy per operation (in order to extend battery life) by reducing the supply voltage, while maintaining the ability to perform latency critical tasks at a higher performance mode by increasing the supply voltage. DVFS becomes especially important in many-core systems, where each core has very different performance demands.

Both dynamic energy and leakage power decrease with decreasing supply voltage. At high supplies, dynamic energy dominates, while at low supplies, leakage energy dominates because cycle time increases exponentially and leakage power is integrated over a longer period of time. There is an optimal supply voltage at which energy per operation is minimized, and enabling operation at this optimal voltage ($V_{opt}$) is vital for DVFS systems.

However, SRAM arrays generally cannot operate at $V_{opt}$. At highly scaled process nodes, such as 28nm, a single weak cell (caused by increasing amounts of variability) in SRAM can cause a failure, so the probability of an error increases exponentially with decreases in the supply voltage. Logic has less of a problem operating at low supplies because the variability of weak cells in the critical path of the processor are averaged, and therefore are less likely to fail at low supplies. For example, Intel shows that in their recent 22nm transistors, SRAM arrays can only function at a VDD of 550mV, while logic can function at a VDD of 280mV [1]. If the L1 remains at the same supply as the processor core, which is very common, then the processor cannot operate at its minimal energy/op point because the SRAM will begin to fail. Industry designs use a large guardband to achieve $E_{min}$ while ensuring that the SRAMs still work, sometimes as much as 150mV [4].

Error correction or detection for the SRAM in the L1 cache allows for the removal of this guardband because a finite number of errors can be allowed to occur—providing substantial energy savings. Additionally, error correction is already required in many commercial designs to deal with potential soft errors due to particle strikes. However, it is unclear which particular error correction technique most effectively provides the required resiliency while minimizing overhead costs. Therefore, we perform an in-depth analysis of cache write policy and its effect on error tolerance, performance, area, and energy.

In general, there are two different ways to tolerate errors in the L1 cache—a write-back cache with single error correction, double error detection (SEC-DED) or a write-through cache with parity detection that corrects by retrieving data from L2. A write-back cache cannot use merely detection because there would be no way to correct dirty data, while a write-through cache could potentially use SEC-DED—where 1 bit errors are corrected locally and double bit errors are corrected from the L2.

This paper has two main goals. Assuming that error resiliency is needed on the L1 cache, our first goal is to iden-
identify the important factors needed to determine what write policy minimizes energy per operation on a DVFS core and memory system. Because energy is our primary metric, we use accurate energy measurements of real designs instead of relying on models, and because results are very sensitive on assumptions, we focus on intuition and sensitivities rather than absolute numbers. Our second goal is to understand the effect of supply scaling and finite error rates to determine whether aggressive voltage scaling (operating in a region with a high error rate) is worthwhile for energy reduction.

In order to accomplish these goals, we need accurate measurements of two pieces—hit/miss rates and energy costs—and to provide these, we have built a cache simulator for the RISC-V ISA simulator and made transistor-level energy measurements of all system components.

2 Related Work

Earlier work on enhancing cache reliability focuses on maintaining performance while minimizing area overhead. Codes can be considered too expensive due to the area overhead of parity or ECC bits on all data (e.g. 12.5%), and latency overhead of error detection and/or correction logic. One approach is to only add protection to just commonly used cache lines by storing redundant data or correction information [3]. Zhang et. al. [12] extended this work by proposing the replication of commonly used cache lines and placing at predicted dead areas of the cache. In both cases, not all the data is protected to reduce area overhead.

To better protect data for a write-back cache, Zhang also proposes [10] [11] using a fully-associative cache to store replicas of dirty data with parity to detect when a replica should be used. This has the advantage of only using parity for detection, but stores require duplication of data. A write-through protects stores by writing through to L2 (one approach used in this paper), but a write-back can avoid this communication by protecting dirty data in L1 with ECC (other approach used in this paper), storing ECC information in a separate cache [8] instead of data replication, or only writing through ECC information to L2 instead of all the data [9]. For example, Punctured ECC Recovery Cache (PERC) in [8] uses a punctured code to split error detection and correction, and stores the error correction bits in a separate cache. Compared to 32 bit granularity for ECC, 5 less bits need to be loaded because only the detection bits are needed, and 1 more needs to be stored because of the cost on a punctured code, resulting in less bits being loaded and stored for typical applications. All these approaches are a balance between write-through and write-back and would likely give slightly better performance and/or energy, but are left out of our analysis due to increased complexity.

More recent work, [5], [8], [6], addresses the impact of these approaches on energy efficiency, but uses very simple models (SimpleScalar and CACTI) that do not accurately represent real processors and caches, especially in highly scaled technologies. Also, they focus on protecting the L1 cache from soft errors, while we propose operating these SRAMs in regions with much higher error rates from issues beyond soft errors. Therefore, they do not include the energy or performance cost of error recovery in their estimates or study the pipeline implications. [5] performs an energy comparison of write-back with ECC, write-through with parity, and their proposed hybrid, but gives very little details on how the results were generated. [8] also performs a high-level comparison including their PERC scheme for performance, L2 cache bandwidth, power, and area. Their analysis concludes 3-5x more power required by a write-through, but does not mention if encoder/decoder or read-modify-write costs are included [6]. Last, [7] compares write policies, but does investigate how each scheme impacts cycle time due to the complexity of the encoder/decoders for error protection.

In comparison to the outstanding body of work, our paper provides the following novel contributions: 1) energy measurements are based on transistor level simulation of actual designs in 28nm instead of models 2) the contributions of various benchmark characteristics and system assumptions are identified rather than merely citing final results 3) energy analysis over a wide range of supply voltages using accurate models of how bit error probability changes with voltage 4) the effect of a finite error rate on energy and performance is investigated at low supply voltages.

3 System Assumptions

This analysis examines the energy consumption of an entire processor plus on-chip memory system shown in Figure 1. For our energy analysis, we assume that both the L1 and core share the same supply voltage—therefore, reducing the supply voltage of the SRAM is required for the core to save energy. There are a few reasons we believe that this assumption is valid. First, level shifting requires a delay of around 100ps in our 28nm process. For a target core speed of 25 FO4 (around 500ps period), this penalty represents a large addition to the critical path. Second, multiple power grids increases the complexity and decreases the effectiveness of the power grid. On-chip DC-DC converters have been proposed to decrease this cost, yet high efficiency converters produce an output supply voltage ripple that will not match between the L1 and core, which can cause dangerous situations where the L1 is much slower than the core and critical paths are violated.

We use the Rocket RISC-V core developed at UC Berkeley as platform because it embodies the properties of a modern mobile in-order processing core. Energy analysis from
a recent tape-out suggests that the instruction cache takes 12% of total core power while the data cache takes 15%, and most cache energy is dissipated in the SRAMs.

The L1 SRAM is optimized for speed so that it can achieve a single cycle latency. The data cache is built from 8kB sub-arrays of 8T cells, allowing for 1 read and 1 write operation of 64 bits per cycle. At 1V, a write operation takes 2.9pJ and a read operation takes 2.0pJ. Reads can only occur at a 64 bit granularity, and write masking allows writes to happen at 8 bit granularity. To avoid increasing the latency, all ways are read in parallel for load operations.

The L2 SRAM typically uses much denser cells that are even worse than cells in the L1 at operating at low supply voltages, so we assume that the L2 resides on its own supply voltage. The data array is built from 8kB sub-arrays that are interleaved 8:1 and access 64 bit words at a time. Read masking allows for only 32 bits to be read. Write masking is not implemented to save area as no sub-double word stores are needed. At 1V, a typical write operation takes 7pJ and a typical read operation takes 3.9pJ, and can maintain 1Ghz operation in ST28nm. The activity of the L2 is largely dependent on the voltage of the L1, so we assume the L2 voltage scales at the same rate as the L1, but stops at a different Vmin. A latency of 7 cycles for L2 accesses is assumed based on a consensus of various recent publications, where a data SRAM access will take 2 cycles. The L2 is protected by ECC because it is a write-back cache, and ECC granularity is 128 bits to save on checkbit overhead. Larger granularities of ECC save checkbit area as shown in Table 1, yet increase encoding and decoding delay slightly, increase energy linearly, and increases RMW operations for write-through caches with coalescing write buffers. Because the interface between the L1 and L2 is 128 bits wide, we have chosen a 128 bit granularity because it simplifies design and represents a fair compromise between area overhead and encoding/decoding complexity and energy. We assume a total L2 size of 256kB. We could easily extend our results to include the L2 to main memory interface.

Table 1: SEC-DED Granularity Comparison

<table>
<thead>
<tr>
<th># of data bits</th>
<th># of 1s in H matrix</th>
<th>Checkbit (overhead)</th>
<th>n-input XOR</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 (8B)</td>
<td>216</td>
<td>8 (12.5%)</td>
<td>27</td>
<td>240ps</td>
</tr>
<tr>
<td>128 (16B)</td>
<td>481</td>
<td>9 (7.03%)</td>
<td>53</td>
<td>280ps</td>
</tr>
<tr>
<td>256 (32B)</td>
<td>1050</td>
<td>10 (3.9%)</td>
<td>105</td>
<td>320ps</td>
</tr>
<tr>
<td>512 (64B)</td>
<td>2241</td>
<td>11 (2.1%)</td>
<td>204</td>
<td>320ps</td>
</tr>
</tbody>
</table>

Table 2: 8x(9,8) parity characterization in 28nm

| Encoding delay: | 117ps |
| Decoding detection delay: | 177ps |
| Decoding correction delay: | L2 latency |
| Encoding energy/op: | 0.07 pJ/op @ 1V |
| Decoding (no error) energy/op: | 0.186 pJ/op @ 1V |
| Decoding (error) energy/op: | allocation cost |
| Checkbit overhead: | (8/64) = 12.5% |
| Encoding Gates | 8x 9-input XOR gate |
| Decoding Gates | 8x 9-input XOR gate |

4 ECC Schemes

In this section, we explore the relationship between bit error probability and L1 operation error probability for different detection/correction schemes, and explore the implications that error identification circuitry has on the pipeline. In particular, the two schemes we investigate are (74,64) SEC-DED and (9,8) parity. We chose these two schemes because they are area neutral—they both require the same number of checkbits. Without this quality, the design space becomes much more complicated as differing cache sizes will affect area and energy.

4.1 Parity: (9,8) Parity Detection + Write-through Correction

We use an RTL implementation that produces one parity bit for every 8 bits. Synthesis in ST28nm LVT process produces the following result:

For parity, any sub-word writes will not require a read-modify-write (RMW) operation, because parity will be assigned as 1 bit for every 8 bits, so only the parity bits for the written word need to be changed. For ECC, sub-word writes will require a RMW operation because 8 bits of checkbits cover 64 bit granularity. The encoder and decoder are separate.

The probability that a bit needs correction and a bit error is uncorrectable is shown below for different bitcell error rates (ber).
Encoding delay: 189ps
Decoding detection delay: 257ps
Decoding correction delay: 387ps
Encoding energy/op: 0.2 pJ/op @ 1V
Decoding (no error) energy/op: 0.875 pJ/op @ 1V
Decoding (error) energy/op: 0.358 pJ/op @ 1V
Checkbit overhead: (8/64) = 12.5%
Encoding Gates: 8x 27-input XOR gate
Decoding Gates: 8x 27-input XOR gate
72x 8-input XOR gate
72x 2-input XOR gate

Table 3: (72,64) SEC-DED characterization in 28nm

\[
P(\text{exactly-zero-errors-in-eight-bits}) = (1 - \text{ber})^8 \\
P(\text{exactly-one-error-in-eight-bits}) = 8 \times \text{ber} \times (1 - \text{ber})^7 \\
P(\text{an-eight-bit-chunk-can-detect}) = \\
\quad P(\text{exactly-zero-errors-in-eight-bits}) \\
\quad + P(\text{exactly-one-error-in-eight-bits}) \\
P(\text{all-eight-chunks-can-detect}) = \\
\quad (P(\text{an-eight-bit-chunk-can-detect}))^8 \\
P(\text{all-eight-chunks-are-error-free}) = \\
\quad (P(\text{exactly-zero-errors-in-eight-bits}))^8 \\
P(\text{uncorrectable}) = 1 - P(\text{all-eight-chunks-can-detect}) \\
P(\text{a-bit-is-corrected}) \approx P(\text{exactly-one-error})
\]

4.2 ECC: (72,64) SEC-DED + Write-back

We use an RTL implementation of an Odd-weight-column SEC-DED code [2]. Synthesis in ST28nm LVT process produces the following result:

Note that the no-error case does not need to localize the error, so the tree of AND gates and the correction XOR gates to not change during normal operation. Any sub-64 bit writes will require a 2-cycle read-modify-write (RMW) operation to regenerate all checkbits. The encoder and decoder are separate.

The probability that a bit needs correction and a bit error is uncorrectable is shown below for different bitcell error rates (ber).

\[
P(\text{exactly-zero-errors}) = (1 - \text{ber})^{64} \\
P(\text{exactly-one-error}) = 64 \times \text{ber} \times (1 - \text{ber})^{63} \\
P(\text{uncorrectable}) = 1 - (P(\text{exactly-zero-errors}) \\
\quad + P(\text{exactly-one-error})) \\
P(\text{a-bit-is-corrected}) \approx P(\text{exactly-one-error})
\]

4.3 Scheme Effectiveness Comparison

The ability of each scheme to correct and detect errors is shown in Figure 2, with an additional inclusion of the SEC-DED+write-through scheme which could correct two errors. As both schemes cover 64 bits at a time, the probability that there is a single error in 64 bits is the same, and therefore the probability that a correction occurs will be the same. However, each scheme responds differently to more than one error. SECDED can detect only 1 error out of 64, while parity can detect up to 8 errors, but only with a probability that all 8 are located in different words and therefore covered by different ECC bits. When parity is combined with write-through, this means that parity can correct up to 8 errors as long as there are no more than one error in each 8 bit block. Note that this analysis assumes nothing about locality of errors. Error locality would diminish the gap between parity uncorrectable rate and SEC-DED uncorrectable rate.

4.4 Architectural Integration

Both schemes incur a delay to encode checkbits during stores and to decode checkbits and detect/correct errors on loads. A typical in-order low power core such as Rocket typically have a critical path of 25FO4 (or around 500ps) through the L1 cache for loads. Therefore waiting for detection to complete for either scheme would increase the operating frequency by an unacceptable amount, and we propose placing all decode logic in a subsequent pipeline stage for loads. Errors will only be detected during load operations, regardless of whether the actual failure happened
during writing, retention, or reading. For loads going to the pipeline, as shown in Figure 3, on errors, incorrect data will have already been written to the register file and forwarded into the execute stage, so the pipeline must be flushed, incurring a cycle penalty of 6 cycles. The incorrect data in the register file will be overwritten when the offending load instruction replays. For loads going to the L2 in our system, there is a queue between L1/L2 because both domains are asynchronous, so data can be corrected in the queue the following cycle.

5 Cache Simulator

A cache simulator was written in C++ to interface with the RISCV ISA simulator. The cache can be parameterized for write-through or write-back, along with size, associativity, and line size. Counters are maintained for relevant operations to allow energy calculation at the end of simulation.

5.1 Write-through

The write-through cache has a line-sized coalescing write buffer implemented as a first-in first-out (FIFO) queue. All stores result in the data being added to the write buffer. If the store is a hit, the L1 is also updated with the data and computed parity bit (associated counters $i_{L1:8}$, $i_{L1:S8}$, $i_{L1:S16}$, $i_{L1:S32}$, $i_{L1:S64}$ incremented). When a store occurs to a full write-buffer and the associated line is not in the write-buffer, the oldest line is sent to L2 ($i_{C\rightarrow L2:S512+64}$ incremented). A 64 bit dirty mask, one bit for every byte, is sent along with the line so the L2 can correctly store the data. A read-modify-write will occur at the L2 ($i_{L2:S128}$ incremented) if the data at ECC granularity is not completely dirty. Otherwise, the data can just be written to L2 ($i_{L2:512}$ incremented). On a load miss, the cache line is retrieved ($i_{L2\rightarrow L1:512}$ incremented) from L2 and the data and computed parity bits are stored in L1. A load hit returns the data and checks the parity bits (associated counters $i_{L1:8}$, $i_{L1:L16}$, $i_{L1:L32}$, $i_{L1:L64}$ incremented).

5.2 Write-back

Any load (associated counters $i_{L1:L8}$, $i_{L1:L16}$, $i_{L1:L32}$, $i_{L1:L64}$ incremented) will eventually read from L1 and check ECC, and any store (associated counters $i_{L1:S8}$, $i_{L1:S16}$, $i_{L1:S32}$, $i_{L1:S64}$ incremented) will eventually write to L1 and compute ECC. On a load or store miss, the write-back must first retrieve the line from L2 ($i_{L2\rightarrow L1:512}$ incremented) and compute the ECC bits before writing to L1. If dirty data is being evicted by a random eviction policy ($i_{L1\rightarrow L2:512}$ incremented), it must be read from L1, have ECC checked, and be sent to L2. Because ECC on L1 is at 64-bit granularity, any store less than double-word must be a read-modify-write operation where ECC is both checked and computed. For any size load, the double-word must be read to check ECC before returning the data.

5.3 Design Intuition

Figure 4 shows the results from our ISA simulator for a subset of the SPEC benchmark suite. Most programs exhibit a high portion of sub-double word writes, requiring read-modify-write operations for a write-back cache. The number of write-throughs past the end of the write buffer can be many times the number of evictions in a write-back cache. Most programs have about twice as many loads as stores.

6 Analytical Energy Analysis

Using frequencies derived from the ISA simulator, we now assign energy costs to each event. Energy costs for different supply voltages are scaled by $CV^2$. 

![Figure 4: Important characteristics of different benchmarks](image)
6.1 Write-through

A cost is assigned to each counter value to obtain total energy usage for the memory. Each size of load hit must read the respective number of bits and check parity bits, and each way must be loaded.

\[
\begin{align*}
\text{ways} & \times (i_{L1:L8} \times e_{L1:L8}+p + i_{L1:L16} \times e_{L1:L16}+p \\
& + i_{L1:L32} \times e_{L1:L32}+p + i_{L1:L64} \times e_{L1:L64}+p)
\end{align*}
\]

Each size of store hit must write the respective number of bits and compute parity bits.

\[
\begin{align*}
i_{L1:S8} & \times e_{L1:S8}+p + i_{L1:S16} \times e_{L1:S16}+p \\
& + i_{L1:S32} \times e_{L1:S32}+p + i_{L1:S64} \times e_{L1:S64}+p
\end{align*}
\]

The write-through of stores from the write buffer has communication costs and varying costs depending if L2 is a write or read-modify-write.

\[
\begin{align*}
i_{C\rightarrow L2:S512+64} & \times e_{C\rightarrow L2:S512+64} \\
& + i_{L2:LS128} \times e_{L2:LS128} + i_{L2:S128} \times e_{L2:S128}
\end{align*}
\]

Lastly, a load miss requires reading a line from L2, sending to L1, and writing data and parity bits to L1.

\[
i_{L2\rightarrow L1:S512} \times (e_{L2:L512} + e_{L2\rightarrow L1:S512} + e_{L1:S512}+p)
\]

Figure 5 shows how design decisions effect memory hierarchy performance for a write-through cache. Coalescing write buffer depth has little effect on energy (preliminary results showed that no buffer at all drastically harmed performance). Also, cache size has little effect beyond 8kB. More ways reduce conflict misses, but every load operation needs to read more ways so an optimum is found at 2 or 4 ways, depending on the benchmark.

6.2 Write-back

The same cost analysis is performed for the write-back cache. All loads require reading a double word to be able to check ECC for every way.

\[
\text{ways} \times ((i_{L1:L8} + i_{L1:L16} + i_{L1:L32} + i_{L1:L64}) \\
* e_{L1:L64+ecc})
\]

To correctly compute the ECC, all double word stores can just write, but sub-double word stores must first read the double word—a read-modify-write operation. However, RMW operations can wait for tag check and do not need to speculatively read all ways.

\[
\begin{align*}
(i_{L1:S8} & \times e_{L1:S8+ecc} + i_{L1:S16} \times e_{L1:S16+ecc} \\
& + i_{L1:S32} \times e_{L1:S32+ecc} + i_{L1:S64} \times e_{L1:S64+ecc}) \\
& + (i_{L1:S8} + i_{L1:S16} + i_{L1:S32}) \times (e_{L1:L64+ecc})
\end{align*}
\]

If an eviction occurs, the line must be read from L1 with ECC checked and sent and written to L2.

\[
i_{L1\rightarrow L2:S512} \times (e_{L1:L512+ecc} + e_{L1\rightarrow L2:S512} + e_{L2:S512})
\]

Lastly, a load miss requires reading a line from L2, sending to L1, and writing data and parity bits to L1.

\[
i_{L2\rightarrow L1:S512} \times (e_{L2:L512} + e_{L2\rightarrow L1:S512} + e_{L1:S512+ecc})
\]

Figure 6 shows how design decisions effect memory hierarchy performance for a write-back cache. Again, increasing size has little effect on energy, and 2 or 4 ways are optimal.

6.3 Sensitivity of model to assumptions

Figure 7 summarizes the difference in energy for write-back vs. write-through implementations. Our results are more dependent on benchmark characteristics than any other factor. Higher L2 energy makes write-back better than write-through in some cases.
**6.4 Effect of voltage scaling**

At high voltages, there are never errors, so the energy of different schemes depends only on a traditional write-back vs. write-through trade-off. However, as the supply voltage decreases, the SRAM error rate increases as shown in Figure 8. For our SRAM, readability is the limiting factor at low supply voltages.

If we assume there are a finite number of errors, then error recovery costs become a concern and a possible differentiator for both schemes. When we allow finite errors, we can lower the supply voltage and make the common case energy-efficient. The main concern with this strategy is that we must still avoid uncorrectable errors. Figure 9 can be used to understand this limit. The two lines shown correspond to the parity scheme. If the L1 cache is 32kB and entries are 64 bits, there are 4096 entries. For 90% yield, on average we can allow the probability that there are uncorrectable to be $1/4096 \times 1/10$ or approximately $2.44e^{-5}$.

Using Figure 9, we can relate this entry error rate to a bitcell error rate of $3.3e^{-4}$. At this bit error rate, the probability that we need to correct an error is then equal to $2.11e^{-2}$. Therefore operating in the most aggressive possible error mode without experiencing any undetectable errors results in about 2% of load operations requiring a correction operation.

By knowing these limits, we can determine the amount of energy savings enabled by operating in a finite error region. To operate in a traditional mode of operation, we need the probability that there are any errors at all to be $2.44e^{-5}$. This corresponds to a bitcell error rate of $3.8e^{-7}$. When operating in an aggressive mode of operation, limited by uncorrectable errors, we have already found that the bitcell error probability needs to be $3.3e^{-4}$. Figure 8 shows the probability of permanent errors versus supply voltage. As the supply voltage decreases, weak cells begin to start failing, and the probability of an error increases. The increases in acceptable BER for the aggressive mode of operation corresponds to a Vmin reduction of 75mV.

Because error rates are now finite, we need to include their effect into the energy per operation calculation of the processor. First, we calculate the correction energy. For the write-back case, the error is corrected locally and has virtually no effect on system energy, so we assign each error no extra correction cost. For the write-through case, each error requires an L2 line load and L2 communication to retrieve the data. This data could be optionally written to the L1 as well, however this decision relies on knowledge of fault longevity (if the cell is always not going to work, there is no reason to write back). One possible optimization is to send less than a single line from the L2. Second, each scheme increases CPI because of correction latency. For the write-back case, we assume a latency of 4 cycles to allow for a pipeline flush. For the write-through case, we assume a latency of 7 cycles (the L2 load latency). These idle cycles add energy of idle cycles to the total energy. These costs are incurred by calculating the probability of a correction for...
each supply voltage, based on the relationship determined by Figures 9 and 8.

Using this method of analysis, we can calculate system energy at different supply voltages, accounting for a finite error rate and energy penalty. Remember from Figure 2 that the probability of uncorrectable errors varies between each scheme, so therefore Vmin will be different for each scheme. However this is a minor effect—SEC-DED with write-back can achieve a 652mV Vmin while parity with write-through can achieve a 635mV Vmin. The third potential scheme, SEC-DED with write-through to correct two errors, can achieve a Vmin of 619mV, suggesting that the overhead required will not be worthwhile because it will increase overhead at all supply voltages to enable a slightly lower Vmin. In general, more complicated correction schemes yield diminishing results because potential decrease in allowable bitcell error rate diminishes for higher numbers of errors. In addition, these schemes require a much larger constant overhead due to more complicated correction schemes and extra checkbits. For this reason, we have limited our implementation to single bit error correction. The probability that a bit is corrected remains the same except for the SEC-DED + write-back case which needs to correct both single and double errors.

Figure 10 summarizes the difference between both schemes for all benchmarks. Note that write-through becomes very expensive at high error rates because correction requires reading from the L2 instead of repairing locally, but Vmin is only 650mV, so this effect is unimportant. By the time the error rate becomes significant enough that a large amount of energy is spent on correction, the probability that there are uncorrectable errors becomes too high and this operating region becomes non-operational. So in reality, correction cost does not differentiate between the schemes, and only comparison at error free operation is needed.

This result tells us that ECC will allow us to operate at a minimum energy point of 21pJ/op at 650mV (for this system). Without ECC and operating without any margin, the minimum energy point would be 26.1pJ/op at 725mV. Assuming 100mV of margin, the safe system would use 34pJ/op at 825mV. ECC allows for the removal of this margin because it can sense Vmin based on error rates. Together ECC saves around 20% of energy per operation.

Because ECC will incur a 12.5% overhead at all supply voltages, unless the core always runs at the minimum energy supply, this technique might not decrease overall energy per operation for the core. However, there are a couple reasons that ECC can be beneficial. First, because the core and L1 are assumed to be tied to the same supply, the 12.5% overhead applies only to memory energy, while the 17% savings applies to both the core and memory energy. Second, if correction is needed anyway to deal with transient and soft errors, and the rate of these errors is much smaller than transient errors, we can save 20% energy by operating at an aggressive supply.

An interesting issue beyond the scope of this work is how to determine the voltage required to achieve this optimal error rate that does not introduce undetectable errors. Figure 8 shows the sensitivity of different reasons of failure to voltage. ECC only finds errors on loads, and cannot determine if the error happened on a load or store. One possible scheme would be to bias the cell such that no stability errors occur, then on errors, perform a safe read operation that takes multiple cycles. If the error remains, then we can conclude that the error was caused on the write, but if the error disappears, we can conclude the error occurred on the store. This information can be used to adjust assist techniques on the array to trade-off between readabilty and writeability.

6.5 Effect of performance on energy

Figure 10 shows that finite error recovery cost is almost irrelevant for an aggressive DVFS system. Instead, we have determined that a more important factor is simply performance.

Figure 11 shows the effect of CPI increase on system energy at 0.8V. A simple analysis says that we increase energy proportionally to the increase in cycles. We assume that a stall cycle only requires half of the energy of a normal cycle, and a more accurate estimate would require processor simulation.

However, in a DVFS system, voltage is set such that a total number of operations can complete in a constant deadline in terms of seconds. Therefore, an increase in CPI also dictates an increase in supply voltage to ensure that the same deadline is met. By using the FO4 of the technology, we
calculate the voltage increase required to continue to meet the deadline, and add this to the calculation. Note that this effect diminishes considerably at lower voltages, because at low voltages, a very small change in voltage greatly changes the delay, so increase in CPI barely effects voltage.

Determining CPI increase for each scheme well enough to make informed design decisions requires cycle-accurate RTL simulation and energy measurement, and is beyond the scope of this paper.

7 Conclusion

Due to increasing variability in modern technology generations, SRAMs are becoming increasingly susceptible to errors. Error rates are exacerbated by DVFS systems that attempt to aggressively lower the supply voltage to operating at the system’s minimal energy per operation. Together with the threat of soft errors, these trends dictate an error correction requirement for the L1 cache in processors. We analyze two methods enable this error resiliency in the L1 cache: parity + write-through and SEC-DED + write-back. A cache simulator is used to determine the frequency of important parameters such as write-throughs vs. evictions and number of sub-word stores for a variety of SPEC benchmarks. These frequencies are multiplied by energy costs extracted from real processor and SRAM designs in 28nm to accurately determine the energy per operation difference between different schemes over a wide range of supply voltages. For our system, write-through vs write-back energy was dependent mostly on the benchmark under consideration. Energy cost of correction measurements was also measured, but found to be irrelevant as the threat of an uncorrectable error limits supply scaling well before repair costs become significant. Also, Vmin difference between each scheme was only 20mV, so Vmin cannot be used to differentiate between the schemes. Therefore we propose that decisions between the different schemes should be based primarily on CPI differences (if idle energy is significant), and L1 vs. L2 energy trade-offs.

References