TRADING OFF CACHE CAPACITY FOR LOW-VOLTAGE OPERATION

TWO PROPOSED TECHNIQUES LET MICROPROCESSORS OPERATE AT LOW VOLTAGES DESPITE HIGH MEMORY-CELL FAILURE RATES. THEY IDENTIFY AND DISABLE DEFECTIVE PORTIONS OF THE CACHE AT TWO GRANULARITIES: INDIVIDUAL WORDS OR PAIRS OF BITS. BOTH TECHNIQUES USE THE ENTIRE CACHE DURING HIGH-VOLTAGE OPERATION WHILE SACRIFICING CACHE CAPACITY DURING LOW-VOLTAGE OPERATION TO REDUCE THE MINIMUM VOLTAGE BELOW 500 MV.

Technology improvements and feature size reduction have coincided with an increase in manufacturing-induced parameter variations. These variations affect various memory-cell circuits, making them unreliable at low voltages. Voltage scaling is therefore limited by a minimum voltage, \( V_{\text{CCmin}} \), beyond which memory circuits fail to operate reliably.

Modern microprocessors contain many large memory structures. For each of these structures, the bit with the highest \( V_{\text{CCmin}} \) determines the whole structure’s. Because defective cells are distributed randomly throughout the microprocessor, the memory structures with the highest capacity (the L1 data and instruction caches and the L2 cache) typically determine the whole microprocessor’s.

Voltage scaling is one of the most effective ways to reduce the power a microprocessor consumes since dynamic and static power have quadratic and exponential dependencies on \( V_{\text{CC}} \), respectively.\(^1\) Therefore, \( V_{\text{CCmin}} \) is a critical parameter that prevents reducing a particular design’s power consumption. Overcoming \( V_{\text{CCmin}} \) limits lets designs operate at lower voltages, improving energy consumption and battery life for handheld and laptop products.

In this article, we show how reducing \( V_{\text{CC}} \) below \( V_{\text{CCmin}} \) affects a microprocessor’s reliability. Based on published bit failure data, we estimate that a 2-Mbyte cache implemented in 65 nm technology (similar to the Intel Core 2 Duo processor’s L2 cache size) has a 825-mV \( V_{\text{CCmin}} \). We examine two architectural mechanisms that will let us redesign some of the largest memory structures (the L1 data and instruction caches and the L2 cache) to decrease \( V_{\text{CCmin}} \). (Prior solutions to the \( V_{\text{CCmin}} \) problem mostly involved increasing cache cell size to achieve lower voltages, therefore sacrificing performance for the same cache area at high voltages; see the “Related Work in Voltage Scaling” sidebar for more details.) With minimal overhead, these schemes significantly reduce \( V_{\text{CCmin}} \) of a cache in low-voltage mode while marginally reducing performance in high-voltage mode. Both schemes achieve a significantly lower overhead compared to error-correcting-code- (ECC-) based defect-tolerance schemes at low voltages.
Related Work in Voltage Scaling

Prior work proposed mechanisms to improve the reliability of memory cells at low voltages. One solution relies on up-sizing devices in the cell, which reduces the impact of variations on device performance and improves cell stability. Alternatives to the traditional six-transistor (6T) static RAM (SRAM) design can also be adopted, including 8T and 10T cells. Kulkarni et al. proposed the Schmidt Trigger (ST) 10T SRAM cell, which has better low-voltage reliability than 6T, 8T, or 10T cell designs for a given area, at the expense of a 100 percent area increase and higher latency.\(^1\) Error-correcting codes (ECC) require a code with the ability to correct 10 bits per cache line to operate at the 500-mV level. Kim et al. demonstrate the high overhead required for multibit ECC schemes.\(^2\) Row and column redundancy would require thousands of redundant rows and columns for a 2-Mbyte cache to operate at 500 mV because nearly one out of 2,000 bits will fail at that voltage.\(^3\)

Although using multiple power supplies with caches running at a higher voltage would minimize area overhead,\(^4\) this approach also has several drawbacks. It requires voltage-level shifters that might increase latency, and an additional power supply will increase the overall platform’s cost. This approach fails to reduce dynamic and static power in the caches. During low-voltage operation, the components operating at high voltage will dominate power consumption.

Figure A shows the \(V_{\text{CCmin}}\) reduction achievable for some of the prior solutions including single error correcting double error detecting (SECDED) ECC, the ST cell, and a hypothetical 10-bit ECC scheme.

![Figure A. The probability of failure (pfail) for various failure mitigation schemes.](image)

References

Impact of \( V_{CC_{\text{min}}} \) on cache reliability

An SRAM cell can fail in four different ways. A read failure occurs when the stored value is flipped during a read operation. This happens when the noise on a node during a read exceeds the trip point of one of the cell’s cross-coupled inverters. A cell with weak pass devices relative to the strength of the cell’s pull-up devices can result in a write failure, in which the cell contents can’t be toggled during a write operation. An access failure occurs when the differential voltage developed across the bitlines during a read operation isn’t sufficient for the sense amplifier to identify the correct value. A retention (hold) failure occurs when the cell’s stored value is lost during standby; these are typically caused by dynamic events unrelated to reading and writing the cell, such as voltage droops. Write and read failures dominate the other two types of cell failures.

A prior work measured the six-transistor (6T) static RAM (SRAM) cell’s failure probabilities as voltage decreases. We used that bit-failure data to derive the probability that memory structures consisting of many bits (SRAM cells) would contain at least one failing bit. Figure 1 shows the probability of failure (pfail) of an SRAM cell (pfail bit) and various multibit structures as a function of voltage. Using the data in Figure 1, we find that a processor similar to the Intel Core 2 Duo implemented in 65-nm technology has a \( V_{CC_{\text{min}}} \) of 825 mV (using a pfail of \( 10^{-3} \)).

Defect-tolerance mechanisms

Ideally, we could turn off a mechanism that handles defective bits when the processor runs in a high-voltage, high-performance operating mode that precludes defects and turn it on when running in a low-voltage operating mode where bit failures are pervasive. We propose two architectural mechanisms that achieve that goal:

- **Word disable** identifies and disables defective words in the cache.
- **Bit fix** identifies broken bit pairs and maintains patches to repair the cache line.

Both mechanisms leverage memory tests to identify portions of the cache that are defective at low voltages. Such defective portions can be used during normal operation at high voltages and disabled to ensure correct operation at low voltages. Both mechanisms trade off cache capacity at low voltages, where performance (and cache capacity) might be less important, to gain the improved reliability required for low-voltage operation. At high voltages where performance is critical, both mechanisms have minimal overhead to maximize the availability of cache resources.

Both mechanisms address bit failures in the cache data array but not the tag array. As a result, we implement our caches’ tag arrays using the Schmidt Trigger (ST) cell. This lets the 32-Kbyte and 2-Mbyte cache tag arrays operate at 460 and 500 mV, respectively. The 2-Mbyte cache’s tag array can also be implemented with single error correcting double error detecting (SECDED) ECC, which will allow correct operation at voltages as low as 400 mV.

Cache word disable

The word-disable mechanism isolates defects on a word-level granularity and then disables words containing defective bits. Each cache line’s tag keeps a defect map with one bit per word that represents whether the word is defective (1) or valid (0). For each
cache set, two physical lines in the same set (line 0 and line 1, for example) combine to form one logical line. After disabling defective words, the two physical lines together store the contents of one logical line. This cuts both the cache size and associativity in half.

Example. Let’s assume we have a 32-Kbyte eight-way L1 cache containing 64-byte lines. For each cache set, eight physical lines correspond to four logical lines. We use a fixed mapping from physical lines to logical lines: Lines in physical ways 0 and 1 combine to form logical line 0, lines in physical ways 2 and 3 combine to form logical line 1, and so on. The first physical line in a pair stores the first eight valid words, and the second stores the logical line’s next eight valid words for a total of 16 32-bit words. As a result, in low-voltage mode, the cache effectively becomes a 16-Kbyte four-way set associative cache. In the tag array, each tag entry must store a 16-bit defect map that consists of one defect bit for each of the 16 32-bit words. These bits are initialized at system boot time and remain unchanged in both high- and low-voltage modes.

Operating in low-voltage mode. During the tag match, the incoming address’s tag bits are matched with the address tag for even ways 0, 2, and so on. We don’t need to match tags in odd ways because they contain the same information as their pair partners. If one of the tags matches, we select the even way (for example, way 0) or the odd way based on the address tag’s sixth-least significant bit because half the words are stored in the even/odd way. We treat requests that require data from two separate 32-byte lines as accesses that cross a cache-line boundary.

To obtain the 32-byte data in aligned form, we use two four-stage shifters to remove the defective words and aggregate working words, as Figure 2 shows. We divide the cache line into two halves, each with a maximum of four defective words and each storing four of the eight required words. A line with more than four defective words in either half renders the whole cache defective. Each of the shifting stages we depict in Figure 2 “shifts out” a single defective word. To control each shifter, we decompose the defect map into four separate bit vectors, each with a single bit set to 1 to indicate the location of a defective word—that is, a 1-hot representation.

Figure 3 shows a simplified version of the logic we use to disable and remove a single defective word from a group. Starting with the defect map, we extract a 1-hot repair vector that identifies a single defective word’s position. In Figure 3, the vector “00100” identifies the third word from the left as defective. The decoder converts the 1-hot vector into a multiplexer control vector containing a string of 0s up to (but not including) the defective position followed by a string of 1s. This has no effect on the words to the left of the defect, but each word to the right of the defective word shifts left, thereby shifting the defective word out. Each level of 32-bit 2:1 muxes eliminates a single defective word. Our proposed implementation must eliminate four defective words, requiring a cache line to pass through four levels of muxes. The additional multiplexing, and associated logic for mux control,
will add latency to the cache access. Assuming 20 fan out of four (FO4) delays per pipeline stage, we estimate that the additional logic increases the cache latency by one cycle.

Cache bit fix

The bit-fix mechanism differs from the word-disable mechanism in three respects. First, instead of disabling at word-level granularity, the bit-fix mechanism lets us disable groups of 2 bits. These groups are defective pairs in which at least one bit is defective. Second, for each defective pair, the bit-fix mechanism maintains a 2-bit patch that can be used to correct the defective pair. Third, the bit-fix mechanism requires no additional storage for repair patterns, instead storing repair patterns in selected cache lines in the data array. This eliminates the need for the additional tag bits required to store the defect map in the word-disable mechanism, but it introduces the need to save repair pointers elsewhere in the system (main memory, for example) while they are not in use (in high-voltage mode).

During low-voltage operation, the repair patterns (repair pointers and patches) are stored in the cache. As a result, any read or write operation on a cache line must first fetch the repair patterns for the cache line. When reading, repair pointers allow reads to avoid reading data from broken bits. Using patches from the repair patterns, the cache line is reconstructed before being forwarded to the core or another cache or being written back to memory. When writing, repair pointers allow writes to avoid writing to broken bits. New patches must be written to the repair patterns to reflect new data written to the cache. Although we focus on reads in the following discussion, reads and writes are symmetric.

Example. Let’s assume we have a 32-Kbyte eight-way L1 cache containing 64-byte lines. Each access to data stored in the cache requires an additional access to retrieve the appropriate repair patterns. To access the repair patterns without increasing the number of ports, the bit-fix scheme organizes the cache into two banks. For our eight-way cache, we maintain two fix lines, one in each bank, with each responsible for repairing cache lines in the opposite bank. The repair patterns for three cache lines fit in a single cache line. Therefore, we maintain a single fix line (a cache line storing repair patterns) for every three cache lines. A fix line is assigned to the bank opposite the three cache lines that use its repair patterns. This strategy allows a cache line to be fetched in parallel with its repair patterns without increasing the number of cache ports.

Operation in low-voltage mode. Figure 4 shows the bit-fix mechanism’s operation at a high level. On a cache hit, both the data line and a fix line are read. In this figure, we fetch the data line from bank A and the fix line from bank B. The data line passes through \( n \) bit shift stages, where \( n \) is the number of defective bit pairs. Each stage removes a defective pair, replacing it with the fixed pair. Because the fix line might also contain broken bits, we apply SECDED ECC to correct the repair patterns in the fix line before they are used.

After the repair patterns have been fixed, they are used to correct the data line. Repairing a single defective pair consists of three parts:

1. SECDED ECC repairs any defective bits in the repair pattern.
2. A defect pointer identifies the defective pair.
3. After the defective pair has been removed, a patch reintroduces the missing correct bits into the cache line.

The bit-fix implementation adds three cycles to the cache access latency, as explained in more detail elsewhere.²

Fix-line structure. To repair a single defective pair in a 512-bit cache line, each defect pointer requires 8 bits (\(\log_{2} 256 = 8\)), and each patch requires 2 bits, in addition to four bits of SECDED ECC to compensate for broken bits in our repair pattern. In total, we need 14 bits for a single defect. Our pfail model estimates that, with a bit-failure probability of 0.001, only three out of every billion cache lines contain more than 10 defective bit pairs. A 140-bit repair pattern can effectively repair a single data cache line with 10 or fewer defects. Each 512-bit fix line will store three 140-bit repair patterns for a total of 420 bits.

Results

Figure 5 shows the improvement in \(V_{CC_{\text{min}}}\) achievable using both the word-disable and bit-fix techniques for a 2-Mbyte cache. For a pfail of 0.001, a 32-Kbyte cache can reliably operate at 490 mV, and a 2-Mbyte cache can reliably operate at 510 mV using the word-disable scheme. For bit fix, a 32-Kbyte cache can reliably operate at 455 mV, and a 2-Mbyte cache at 475 mV. Compared to a modern high-performance microprocessor, where low-power voltages are in the 800 to 850 mV range, this represents a more than 40 percent drop in \(V_{CC_{\text{min}}}\).

Overhead

Both word disable and bit fix require that we perform memory tests when the processor is powered on. When switching between low- and high-voltage modes, the cache contents must be flushed to memory. In addition, both schemes require improving the tags’ reliability through the use of ST cells and potentially ECC. The tag array is roughly 5 percent of the data array’s size. Using ST cells and ECC will increase the tag array’s size by roughly 2.5 times for the bit-fix mechanism, and four times for the word-disable mechanism, which also requires a more reliable defect map. For a 2-Mbyte cache, the bit-fix mechanism increases the cache size by approximately 7 percent, and word disable increases the cache size by approximately 15 percent.

Compared to word disable, bit fix has a lower area overhead, operates at a lower \(V_{CC_{\text{min}}}\), and preserves three-fourths of the cache for use.
at low voltages versus one-half for the word-disable approach. However, word disable requires less complex muxing and, as a result, has a smaller cache latency. Word disable’s tag overhead includes storage for the defect map, which eliminates the bit-fix complexity of using the cache to store repair patterns.

Performance and power

We compared a processor that uses word disable in the L1 cache (where latency is critical) and bit fix in the L2 cache (where cache size is more important) to a baseline design using a 6T SRAM cell that must operate at higher voltages and an alternative low-voltage design using the ST cell. (Execution times for different mechanisms appear elsewhere.2)

Table 1 summarizes the achievable \( V_{CC_{min}} \), cache area, frequency, power consumption, instructions per cycle (IPC), and energy per instruction (EPI) of each scheme in the low-voltage mode. We normalized the area, power, and EPI results for each scheme to the corresponding results for the conventional 6T-cell-based cache, while showing absolute results for \( V_{CC_{min}} \), frequency, and IPC. To estimate power consumption, we assumed that dynamic power scales quadratically with supply voltage and linearly with frequency. We also assumed that static power scales with the cube of supply voltage. (This is an approximation of the exponential relationship described in an earlier work.3) We ignored an ST cell’s additional latency and leakage due to its larger area.

Table 1. Comparison between the ST cell-based cache and proposed schemes.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>( V_{CC_{min}} ) (mV)</th>
<th>Normalized cache area</th>
<th>Clock speed (MHz)</th>
<th>Normalized power</th>
<th>IPC</th>
<th>Normalized EPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>6T cell</td>
<td>825</td>
<td>1</td>
<td>1900</td>
<td>1</td>
<td>1.02</td>
<td>1</td>
</tr>
<tr>
<td>ST cell</td>
<td>530</td>
<td>2</td>
<td>700</td>
<td>0.19</td>
<td>1.17</td>
<td>0.45</td>
</tr>
<tr>
<td>L1WDis_L2BFix</td>
<td>500</td>
<td>1.08</td>
<td>600</td>
<td>0.15</td>
<td>1.07</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Reducing the area overhead of the ST-cell-based cache by reducing the cache capacity isn’t an option because it sacrifices performance in the high-voltage mode. Although our scheme’s latency overhead and capacity loss in the low-voltage mode result in a lower IPC compared to the ST-cell-based cache (where we ignore overheads), they achieve a similar energy efficiency to the ST-cell-based cache, which is a more than 50 percent improvement over the 6T-cell-based cache.

As customer demand continues to drive the development of energy-efficient microprocessors, architectural solutions for low power are becoming increasingly important. \( V_{CC_{min}} \) is a critical parameter that affects microprocessor power and reliability, and the improvements in energy efficiency we’ve described here can significantly increase battery life for mobile and laptop products as well as reduce the energy cost for desktop and server products.

Acknowledgments

We are grateful to Kaushik Roy and Jaydeep Kulkarni for their feedback and help with the bit-failure data. We thank Tingting Sha, Eric Ernst, and the article’s reviewers for their constructive feedback.

References

Chris Wilkerson is a research scientist in the Oregon Microarchitecture Lab at Intel. His research interests include low-power design including microarchitectural mechanisms to enable low-voltage operation for microprocessors. Wilkerson has a combined BS and MS in electrical and computer engineering from Carnegie Mellon University.

Hongliang Gao is a computer architect in the Digital Enterprise Group at Intel in Oregon. His research interests include control-flow speculation, fault tolerance, and power efficiency in processor designs. Gao has a PhD in computer science from the University of Central Florida.

Alaa R. Alameldeen is a research scientist in the Oregon Microarchitecture Lab at Intel. His research interests include memory and cache design for multiprocessors and power-efficient processor design. Alameldeen has a PhD in computer science from the University of Wisconsin-Madison.

Zeshan Chishti is a research scientist in the Oregon Microarchitecture Lab at Intel. His research interests include energy-efficient microprocessor and memory design and cache hierarchies for chip multiprocessors. Chishti has a PhD in computer engineering from Purdue University.

Muhammad Khellah is a research scientist in the Circuits Research Lab at Intel in Oregon. He works on embedded memories and interconnect circuits. Khellah has a PhD in electrical and computer engineering from the University of Waterloo.

Shih-Lien Lu is a principal engineer and currently leads the Oregon Microarchitecture Lab at Intel. His research interests include resilient design and low power architecture. Lu has a PhD in computer science and engineering from the University of California, Los Angeles.

Direct questions and comments about this article to Alaa R. Alameldeen, 2111 NE 25th Ave., Mailstop JF2-04, Hillsboro, OR 97124; alaa.r.alameldeen@intel.com.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/csdl.