On-Chip Cache Device Scaling Limits and Effective Fault Repair Techniques in Future Nanoscale Technology

David Roberts, Nam Sung Kim*, Trevor Mudge
University of Michigan, *Intel Corporation
daverobe@umich.edu, nam.sung.kim@intel.com, tnm@eecs.umich.edu

Abstract

In this study, we investigate different cache fault tolerance techniques to determine which will be most effective when on-chip memory cell defect probabilities exceed those of current technologies, which is highly anticipated in processor on-chip caches manufactured with future nanometer scale technologies. Our most significant finding from this study is that the devices in on-chip memory cells cannot be scaled at the same rate as devices in logic circuits due to the increasing number of erroneous memory cells with voltage scaling, requiring strong fault-tolerance techniques. Second, we propose a technique to minimize performance impacts under aggressive technology and voltage scaling. It works by merging pairs of faulty cache lines to make good lines and performs better than TMR at high error rates and at lower cost. We also estimate up to 28% energy savings at low voltage, relative to a recent fault-tolerance scheme [1].

Keywords: cache, device scaling, fault-tolerance, DVS

1. Introduction

As microarchitects demand larger on-chip caches for higher performance, continuous device scaling has provided improved memory density for multi-megabyte upper-level on-chip caches at a reasonable die cost. However, the device scaling comes at a price. The reduced device feature size causes exponentially increasing subthreshold and gate-leakage power problems in on-chip caches fabricated with sub-90nm process technology resulting in more static power consumption [2]. Furthermore, process parameter variations have worsened yield problems in on-chip caches manufactured with sub-90nm technology [3]. These include random dopant fluctuations and oxide defects.

To overcome low yield problems caused by scaling device sizes and integrating more on-chip memory cells, there have been several proposed techniques. One is to implement redundant memory columns; there are one or two redundant columns per memory sub-bank or sub-array. If a defective cell is found during the manufacturing test, the entire column containing the defective cell is replaced with a redundant column. This wastes many memory cells to fix one defective cell and requires fuses to replace the column containing the defective cell with the redundant column. The second technique is to use error correction codes (ECC). Currently, a single error correction (SEC) and double-error detection (DED) technique is used. Even though this can fix one defective cell per sub-array row, the memory array is made more vulnerable to soft errors since the correction capability of the code has been used up by fixing defective memory cells. The third technique is to disable a part of the on-chip cache memory array resulting in a smaller capacity. An example is the Intel Celeron processor. It is very similar to the Pentium processor, but has all or half of the L2 cache disabled as a result of memory sub-arrays containing defective cells that could not be fixed using the redundant columns in the disabled part of the on-chip cache memory block. All of these techniques are only effective when there are a small number of defective cells in the on-chip cache. However, the number of defective cells in large on-chip caches will rise if we want to continue scaling memory cell size along with technology scaling.

Hard-wired redundancy is becoming a less attractive option due to limited area available for spare memory cells. In addition, it will no longer be possible to find a single set of cache blocks which consistently fail at each operating point [3].

Under aggressive voltage scaling and on-chip memory cell sizing, we show that higher defect rates with existing fault tolerance schemes result in significant processor performance degradation. A dynamic voltage scaling (DVS) environment adds to the complexity of working with on-chip caches containing unpredictable defective memory cells; as the operating voltage changes, so does the number of defective cells.

In this paper we begin with an analysis of L2 cache activity in a modern processor architecture based on the Intel Pentium 4. Emphasis is placed on L2 caches because of their widespread use and relatively large area compared to L1 (L1 caches are also relevant, and the error analysis within this paper can also be applied to other levels besides L2). We show the impacts of defective cache blocks on performance and compare ways of addressing this problem. The major contributions of this paper are;

- Trade-off analysis between performance and area for different cell sizes and fault-tolerance techniques.
- A novel cache block grouping scheme for good performance at higher fault probabilities.

10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)
0-7695-2978-X/07 $25.00 © 2007 IEEE

The rest of the paper is organized as follows. Section 2 presents related work and explains in detail the basic fault-tolerance scheme upon which this work is based. Section 3 explains the problems encountered with on-chip cache memory reliability in new processes and its impact on performance of set-associative caches when defects are present. Based upon this analysis, we show existing and proposed techniques of reducing performance impacts in the presence of defects in on-chip caches in Section 4. The techniques are compared in Section 5, and concluding remarks are presented in Section 6.

2. Related Work

Pour et al. [4] derive an analytical model of the performance loss of a set-associative cache given a set of defective blocks. They employ an extra "valid" bit per cache block to identify whether or not it is defective. Their key findings for caches of size up to 32 KB are that miss ratio increase is negligible unless a set is completely disabled by faults.

In [5] they present a model to estimate memory-failure probability using combined row and column redundancy.

The Power4 architecture [6] employs parity on L1 caches and Hamming codes on L2. In addition, L1 and L2 have spare bits, while L3 has redundant cache lines. If correctable error thresholds are exceeded, a cache line delete function allows up to 2 deletions per L3 cache. For defects detected at power-on BIST that cannot be handled, the L3 cache is disabled.

The Nanobox [7] applies redundancy and other ECC codes to logic functions built using lookup tables.

A technique for memory self-repair at high defect densities is presented in [8]. It relies upon prior knowledge of the polarity of the error (i.e. faults are always stuck at 0 or 1). In our cache application, the scheme will not work because the value read from faulty bits is unpredictable and can change with operating point (e.g. voltage or temperature).

Agarwal et al. [1] noted that the number of defective cells and their location changes depending on operating voltage. In addition, they proposed a cache block re-mapping technique for direct-mapped caches. The technique relies on a defective block mapping table determined prior to execution using BIST. They consider the use of block re-mapping in conjunction with ECC and row redundancy. Because we often refer to this scheme in the paper, it is explained in more detail as follows. Figure 1 is the same figure as Figure 7 in [1] and illustrates the fault tolerance scheme presented in that work. It is based on a direct-mapped cache consisting of lines organized in rows and columns. Rows are addressed as usual using part of the incoming address. However, the column address may be re-mapped to avoid a known faulty block. This is achieved

Figure 1. The one bit implementation (OBI)

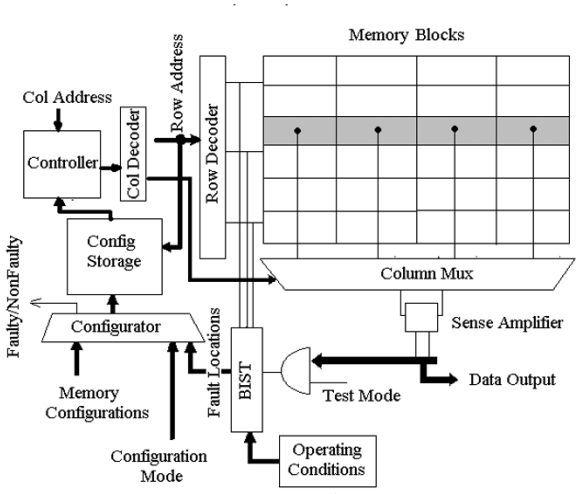
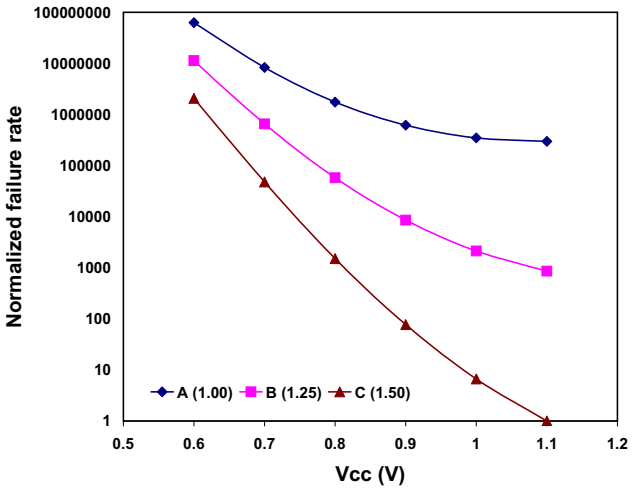


Figure 2. Normalized cell failure rates.



by performing a look-up in the "config storage" which contains a map of defective block locations. In this instance, there is one bit per block (hence one bit implementation, or OBI) which is set to 1 if the corresponding block is defective. When the cache is accessed, the controller uses the OBI to select a non-defective column to store data to, using a fixed mapping. Additional bits are required in the tag to indicate the column in which data is stored. This prevents faulty blocks from being read.

3. Impact of On-Chip Cache Failure Rate on Processor Performance

3.1 On-Chip Cache Device Scaling and Failure Rate

Figure 2 shows normalized on-chip memory cell failure rates for 3 different memory cell sizes (A, B and C with relative areas of 1, 1.25, and 1.5, respectively) as a

Table 1: M5 CPU configuration (2 GHz clock).

Parameters	Value
pipeline width	4
branch prediction / BTB	hybrid, 4-way, 2K entries
ROB / LSQ size	196 / 32 entries
INT ALUs/multi-divs/ mem ports	6 / 2 / 4
FP ALUs / multi-divs	4 / 2
functional unit latencies	INT: mul 3, div 20, all others 1 FP: adder 2, mul 4, div 12, sqrt 24
IL1 cache	16KB, 2-way, 64B blocks, 1-cycle lat.
DL1 cache	16KB, 2-way, 64B blocks, 3-cycle lat.
L2 cache	1MB, 8-way,64B blocks, 19-cycle lat.
memory bus / latency	16 bytes with 6-cycle lat. / 100 cycles

function of memory cell supply voltage. Note that depending on how we tune the sizes of the 6 transistors in the memory cell, the result varies significantly. They are obtained using Monte-Carlo simulations on memory cells designed with a 45nm technology and process parameter variations corresponding to the technology. The failure rate we assume is significantly higher than the data presented in other work [1], however the increased failure rates can be expected in future smaller semiconductor process technology (e.g., 32nm technology). As shown in Figure 2, as either voltage or cell size decreases, the failure rate starts to increase exponentially. In other words, a larger cell can achieve a much lower voltage at the same failure rate. Finally, defect rate is proportional to die size. Hence, when we integrate more on-chip memory cells on a die along with device scaling, there will be a much greater chance that some memory cells contain defects and fail during post-manufacturing tests resulting in poor yield. The next section examines the relationship between on-chip cache memory cell failure and performance impact to determine the number of tolerable, non-corrected faults.

3.2 Performance Impact of Set-Associative Cache Defects

To assess the impact of faulty cache blocks, we examine the performance of standard benchmarks on a realistic processor. All data was obtained using the M5 simulator [10]. The simulator was configured to represent a modern out-of-order pipeline with similar specifications to a Pentium 4 (Table 1). The memory latency is relatively low, although this will not significantly affect L2 miss rates.

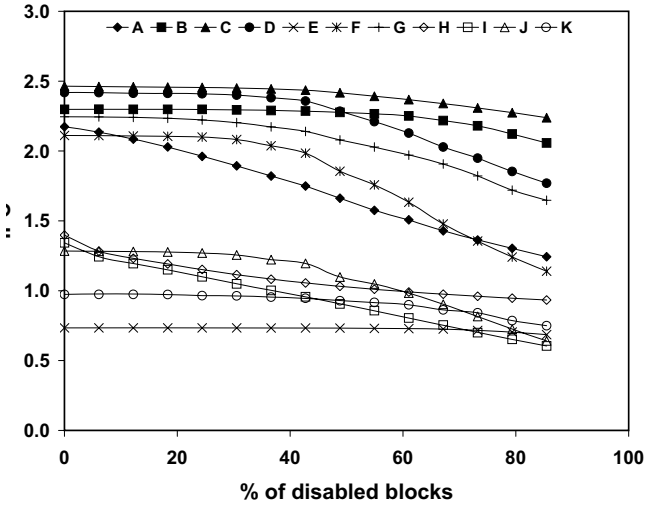
Figure 3 presents the impact of defective blocks on IPC in L2 (each containing 1 or more defective cells). We use the SPLASH-2 benchmark suite [11] as a workload representative of both memory and compute-intensive applications. Defective block locations are allocated randomly, but consistently between benchmarks. The LRU replacement algorithm was modified so that defective (non-correctable) blocks are disabled, identical to the scheme in [12]. The data in Figure 3 confirms the previous study [4] in that high block failure rates are required before there is any significant performance penalty. With this in mind, the next section compares existing and our fault-tolerance techniques for their performance and area at significant failure rates.

4. Comparison of Fault-Tolerance Techniques

The one-bit-implementation (OBI) mapping table model [1] is effective for low failure rates, but for higher rates we show that it rapidly becomes ineffective. We aim to allow more faults with stronger error correction, and observe the trade-off with area cost. In this section we derive an analytical model representing the fraction of good blocks remaining in the cache at different cell failure probabilities (and sizes). In the error models we consider tag bits as additional bits contained in each block.

We compared several cache fault-tolerance schemes in order to determine their area efficiency at different error rates (voltages and cell sizes). The model used represents the fraction of fault-free blocks available in the cache, denoted by F_{avail}. As a minimum, we decided to first apply the OBI scheme, followed by other error correction. From a storage standpoint, OBI provides the minimum data needed to identify where faulty blocks are, for avoid-

Figure 3. IPC as a function of #disabled blocks



A-Cholesky, B-FFT, C-LUContig, D-LUNoncontig, E-Radix, F-Barnes, G-FMM, H-OceanContig, I-OceanNoncontig, J-Raytrace, and K-WaterNSquared, respectively.

0-7695-2978-X/07 $25.00 © 2007 IEEE

Table 2: Candidate fault-tolerance schemes.

Schemes	Storage (MB)	Description
OBI	1.002	One-Bit Implementation from [1]
Hamming SEC	1.020	Single Error Correcting Hamming code.
BCH DEC	1.037	Double Error Correcting BCH code.
log(B)	1.022	Bad block table contains index of one bad bit and a spare cell to store the value of that faulty bit. If there is more than one defective bit, the block is disabled.
Triple-modular redundancy (TMR)	1.002	Faulty blocks are combined in groups of 3 inside the 1MB cache with a majority vote on each bit.
Block grouping (GRP2)	up to 1.528	Pairs of faulty blocks are combined to form single good blocks. A paired block is "good" if there is only one faulty bit for each corresponding pair of 2 bits.

ance. Since it has already been proven superior to redundant rows and SECDED ECC, all of our models build on the OBI baseline. OBI does not affect the cache access time and has minimum effect on processor performance [1]. Throughout the paper we refer to p_{fault} as the probability of failure of a single on-chip memory cell. Cell failures are assumed to be independent. As a first approximation this assumption is valid and has been widely used in other published cache-error related work ([1][5][8]).

To improve the effectiveness of schemes requiring an additional storage table (e.g. OBI) which must contain correct bits, we introduce a factor OBI_{ff}. This factor reduces the bit failure probability, representing larger size or higher voltage on-chip cache memory cells (see Figure 2) used specifically for that table. We refer to this as "guaranteed correct" storage because one fault in this table could lead to bad cells being accessed. Using large cells is viable as long as the table does not contain too much data. In addition, delay does not vary significantly with cell size. "Basic storage" refers to the cells used in the cache itself.

The fault-tolerance schemes (Table 2) were chosen from a range of candidates, most of which are widely used today. We only model storage-related reliability while logic reliability is beyond the scope of this paper. The following sections explain the fault model and storage overhead of each scheme.

Each cache consists of M sets and N ways where each block contains B bits (including tag bits).

We also derive an area efficiency E_{area} which takes into account the probability of failure of the "guaranteed correct" storage which includes the OBI table and any additional bits added by a scheme which must be correct for the cache to operate reliably. The fraction of available blocks is divided by die area consumed by all SRAM cells, then scaled by the probability of the guaranteed correct storage containing no faults (EQ 1).

$$E_{area} = \frac{F_{avail}}{area} \times p_{non\_faulty\_GC} \quad \text{(EQ 1)}$$

The $p_{non\_faulty\_GC}$ value is the probability that the guarranteed correct cells are fault-free, as a function of the probability of the large-size cell failure p_{fault_GC} and the number of guaranteed correct bits (*GC_bits*).

$$p_{non\_faulty\_GC} = (1 - p_{fault\_GC})^{GC\_bits} \quad \text{(EQ 2)}$$

In all of these schemes,

$p_{fault\_GC} = \text{OBI}_{ff} \times p_{fault} = 10^{-5} \times p_{fault}$.

4.1 Existing fault-tolerance schemes

4.1.1 OBI

The "one bit implementation" consists of a table of bits, one per block, indicating whether or not each block contains 1 or more faulty bits. All of our schemes incur this storage overhead, because we use an OBI to indicate whether a block can be corrected or is unusable and cannot be accessed.

Storage overhead

GC_bits = M × N bits (guaranteed correct storage).

Fault model

The probability of a faulty bit is p_{fault}. The probability of a non-faulty block is the likelihood of every bit being fault-free in that block. For our typical-case analysis, we assume that this probability represents the fraction of non-faulty cache blocks, as follows;

$$F_{avail} = (1 - p_{fault})^{B} \quad \text{(EQ 3)}$$

4.1.2 SEC

Single error correcting (SEC) codes were included due to their widespread use in existing devices.

Storage overhead

$$b = \lceil log_2(B) \rceil \quad \text{(EQ 4)}$$

where *b* is the number of added ECC bits per cache block (basic storage).

Fault model

The model is modified to account for the increased block size (for check bit storage) and the ability to correct 0 or 1 bits.

$$F_{avail} = (1 - p_{fault})^{B+b} + \binom{B+b}{1} \times p_{fault} \times (1 - p_{fault})^{B+b-1} \quad \text{(EQ 5)}$$

4.1.3 BCH (DEC)

The BCH error correcting code was selected as a candidate DEC scheme. Alternatives such as Reed-Solomon and Golay codes are mentioned in [13]. BCH was chosen because of its low storage overhead. However, in practise a less compute-intensive code can be used depending on the sensitivity of performance on L2 latency. We modelled a DEC BCH code storage overhead based upon the equations in [14].

Storage overhead

$$b = 2 \times \lceil log_2(B) \rceil \quad \text{(EQ 6)}$$

where *b* is the number of added ECC bits per cache block (basic storage).

Fault model

The probability of a faulty block is modified to account for the extra check bit storage and the ability to correct 2 bits.

$$F_{avail} = (1-p_{fault})^{B+b} + \binom{B+b}{1} \times p_{fault} \times (1-p_{fault})^{B+b-1} + \binom{B+b}{2} \times p_{fault}^2 \times (1-p_{fault})^{B+b-2} \quad \text{(EQ 7)}$$

4.1.4 Log(B)

The *log(B)* scheme is an alternative single-error-correcting SEC scheme. A table stores the index of one faulty cell location per block, along with an additional bit to hold the correct state of that cell. This is equivalent to the distant repair scheme of [9] using one spare unit. The model equations are similar to SEC.

4.1.5 Triple Modular Redundancy (TMR)

Our triple modular redundancy implementation assigns faulty blocks to groups of 3 blocks with a majority vote on every bit (0 or 1 errors can be corrected per bit position). At most, 1/3 of logical bits can be recovered from the physical bits which are combined for a majority vote. In a hardware implementation, the bit comparison for the majority vote is performed at the final cache output stage. Therefore logic overhead will be small.

Storage overhead

No additional storage is allocated to identify which blocks are combined for TMR. For this 'typical case' analysis, we assume that faulty blocks are combined with other arbitrarily located faulty blocks.

Fault model

We first consider each bit index as 3 bits which must have 0 or 1 faults to be corrected. This applies for all B bit indexes. However, because we only combine known *faulty* blocks after determining fault-free blocks, none of the three blocks are ever error-free and this probability (p_{good}) is subtracted from the main expression. The probability of a non-faulty blocks is $p_{nfb} = (1-p_{fault})^B$.

$$p_{good} = \binom{3}{1} p_{nfb} \left[(1-p_{fault})^2 + \binom{2}{1} (1-p_{fault}) \times p_{fault} \right]^B - 3(1-p_{nfb})(p_{nfb})^2 - 2(p_{nfb})^3 \quad \text{(EQ 8)}$$

$$fraction\_repaired_{TMR} = \frac{1}{3} \times \left(\left(\binom{3}{1} \times p_{fault} \times (1-p_{fault})^2 + (1-p_{fault})^3 \right)^B - p_{good} \right) \quad \text{(EQ 9)}$$

$$F_{avail} = p_{nfb} + (1-p_{nfb}) \times fraction\_repaired_{TMR} \quad \text{(EQ 10)}$$

4.2 Proposed Fault-Tolerance Scheme

4.2.1 Block Grouping

For high error rates, we propose a new scheme (Figure 4). Faulty physical blocks are grouped together (in groups of size G) to form a new, fully working logical block. In the rest of our analysis, we assume pairs (G=2). Using larger groups is beneficial at extremely high error rates, but the analysis is beyond the scope of this paper. The concept is a similar to [8] except that knowledge of failure polarities is not required.

Compatible blocks have up to one faulty bit between them, at every corresponding bit index. This means that an additional "selector bit," which is known to be correct, can specify which bit contains a good value when reading data.

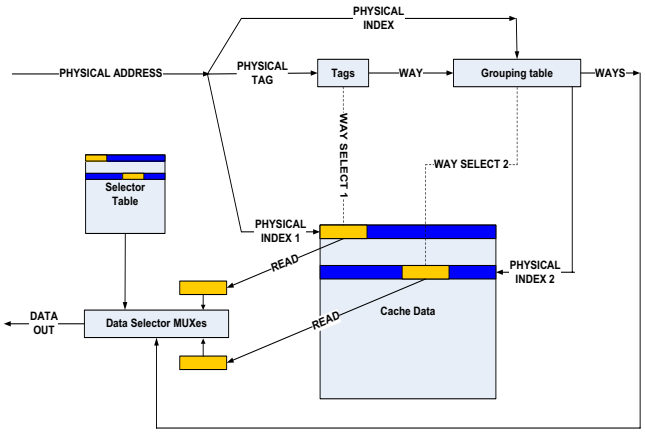
A "grouping table" is accessed as an additional step before a cache access, to identify the paired block.

- To read a grouped block, all blocks in the group are read. The selector bits then indicate which block in the pair contains good data, at each bit index. A single logical block is then returned to the processor.
- To write a grouped block, the same value is written to every component block.

Grouping table

This table is used to look up the location of the other block in a group. If there is more freedom to combine faulty blocks that are compatible, more blocks can be recovered. They can be physically adjacent, in the same set or in any location inside the cache (depending on the desired complexity of block selection hardware). Each alternative has performance tradeoffs, discussed later.

Figure 4. The proposed block grouping scheme.



Selector bit table

Selection of the block which the data bit is read from is performed using a table of selector bits. These are stored in guaranteed correct cells, and can either cover every bit index in the block (50% storage overhead for groups of size 2) or a small number of adjacent bits. Using fewer selector bits reduces the number of defects which can be tolerated but reduces storage overhead. For example, a single selector bit covering two adjacent data bits cannot handle the case where there is a fault in both blocks at that position. Later, we discuss off-chip caching of selector bits to reduce the die area of on-chip SRAM.

Table configuration

The tables are programmed during system start-up. Self-test routines determine whether cache cells are operating reliably at each voltage and frequency point, and the map is stored in main memory or on disk. When performance settings change, the cache is flushed and a new selector table loaded. These tables could also be hard-wired at manufacturing test.

Storage overhead

We call the first block to be accessed the "primary" block, and its paired compatible block the "secondary" block. For the grouping table, we first consider the most storage-intensive scenario where blocks are paired anywhere in the cache. The grouping table has a number of entries equal to the number of blocks in the cache. Each entry stores the set and way index of a compatible block, to be looked up on a read access. The equation below assumes that there is an entry pointing to another block for every block position in the cache.

$$group\_table\_size = M \times N \times log_2(M \times N) \qquad \text{(EQ 11)}$$

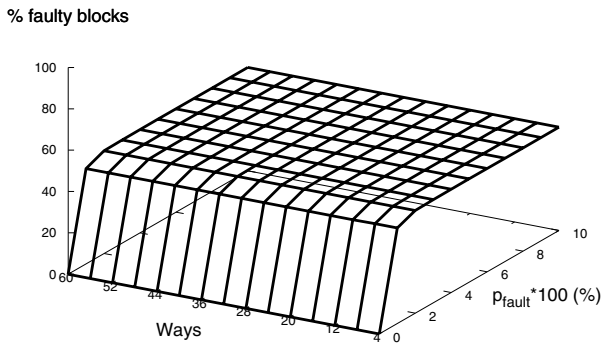
As a lower-cost alternative, the storage requirement for pairs limited to the same set is given below. When implemented as an associative look-up, half of the blocks in a set have a pointer to another block in the same set.

$$group\_table\_size = \frac{N}{2} \times log_2(N) \times M \qquad \text{(EQ 12)}$$

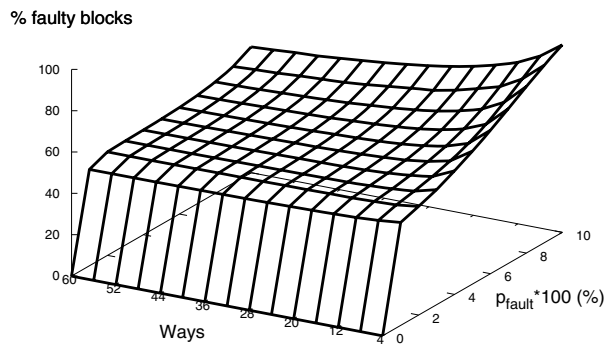
Instead of using a grouping table, one could use the existing tag matching mechanism to simultaneously hit multiple blocks of the same group since their address tags are identical. It requires that the group resides in a single set so that address indexes are identical for each block. A banked cache design where ways are in different banks would allow fast parallel access to a group of blocks. A sequential access model is still feasible however, because an extra cycle to look up a secondary block does not significantly impact performance for low-level caches (e.g. L2). Another, less effective zero-overhead alternative is to use a fixed grouping, for example, pairing together adjacent blocks.

The error-correcting ability of two variants are shown in Figure 5. The results were derived from simulation, and pairs were formed using a greedy algorithm that allocates each faulty block with the next free compatible faulty

Figure 5. Percentage of faulty blocks using block pairing. Block size = 32 bytes.



(a) Paired blocks located anywhere in the cache



(b) Pairs must be in same set

block in sequential order. An optimal grouping will be even more effective.

It is clear that arbitrary pairing (Figure 5(a)) is most effective at high error rates, although the per-set limitation (Figure 5(b)) can be almost as effective. Increased associativity helps in this case by providing more pairing alternatives and can be seen in processors such as Niagara, with a 12-way L2 cache [15].

The number of selector bits (used to choose one good bit from a group of G bits at each bit index) is the logarithm of the number of bits in the group. There are B selectors per cache block, and $(M \times N)/G$ logical blocks after grouping.

$$selector\_bits = \frac{M \times N}{G} \times (log_2(G) \times B) \quad \text{(EQ 13)}$$

where G is the number of blocks in each group.

$GC\_bits = M \times N + group\_table\_size + selector\_bits$ (guaranteed correct storage).

Fault model

The parameter G can be varied, but all analysis in this paper uses block pairs (G=2).

$$p_{good} = p_{nfb}^2 + \binom{2}{1} p_{nfb} \times (1 - p_{nfb}) \quad \text{(EQ 14)}$$

$$fraction\_repaired_{GRP2} = \frac{1}{2} \times ((1 - p_{fault}^2)^B - p_{good}) \quad \text{(EQ 15)}$$

$$F_{avail} = p_{nfb} + (1 - p_{nfb}) \times fraction\_repaired_{GRP2} \quad \text{(EQ 16)}$$

Each logical bit is formed from two bits. A set of two faulty physical blocks are compatible and can be recovered into a single logical block when;

- At most one of the grouped bits at each bit index are faulty, and
- This is true at every bit index in the block of B bits

Note that we make an adjustment p_{good} to remove the impossible cases where any block contains no faults, as per TMR.

4.2.2 Selector Bit Caching for Block Grouping

Compared with the other schemes, block grouping has strong fault tolerance characteristics but a potentially large storage overhead. By caching the working set of selector bits in on-chip SRAM and keeping less frequently used bits off-chip, we show that area overheads can be reduced without significant performance impact.

The design in Figure 4 was extended to have the working set of selector bit pages (stored in a parallel structure to the TLB) on-chip. On a page fault, we assume that the page table is accessed from main memory, so access latency to a small off-chip DRAM holding pages of selector bits is already accounted for.

Table 3: Selector bit caching results.

Parameters	Value
Cache size	1024KB
Linux page size	8KB
Logical bits per selector bit	1
Selector page size	4KB
Full grouping table size / OBI table size	4KB / 2KB
Set-restricted grouping table size	3KB
On-chip SRAM selector bit storage	96KB
Off-chip DRAM selector bit storage	512KB
ITLB / DTLB entries	8 / 16
Selector bit table DRAM throughput	32 bytes / cycle
Total storage overhead of full group table	12.3%
Total storage overhead of set-limited group table	9.9%
Total storage overhead of tag-based grouping table	9.6%

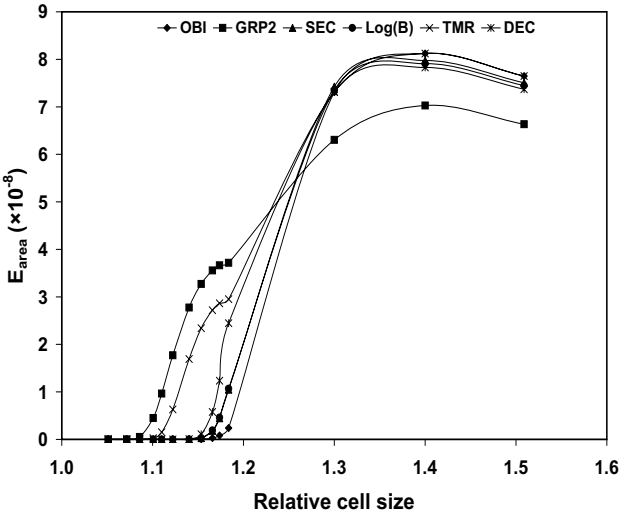
Using the same M5 configuration as before, we recorded TLB miss rates for varying numbers of TLB entries, then derived the performance hit for off-chip selector bit loading. For more realism and to support virtual memory, the simulator was run in full-system mode and benchmarks were run to completion under Linux. The results are given in Table 3. By keeping just the working set of cache pages in on-chip SRAM we have reduced the on-die storage overhead from 50% to less than 10%. We opted to use 24 TLB entries because the performance data indicated much smaller slowdowns of 2% and 8% respectively for the Cholesky and OceanNonContig benchmarks. Due to the larger working set of OceanNonContig, increasing the number of data TLB entries does not significantly reduce miss rate.

5. Results

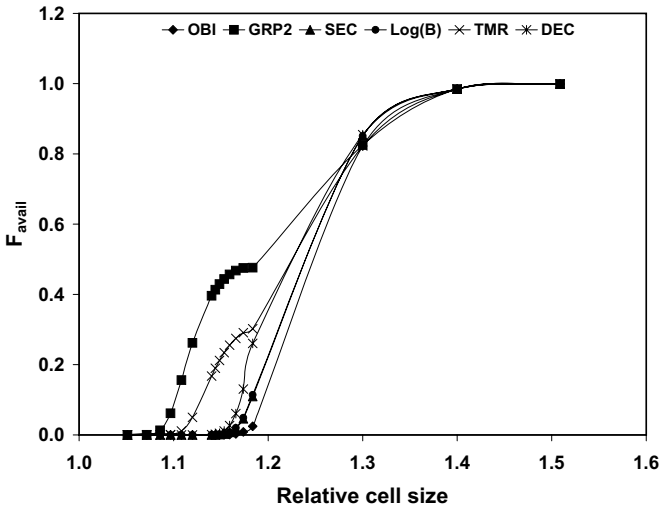
The E_{area} metric is shown in Figure 6-(a). The most area efficient scheme is to use an OBI with TMR, as long as a cell is not scaled below size 1.4. This can be seen in the figure as the point of greatest E_{area} value. In fact, there is only a marginal improvement over using an OBI alone. Therefore, the area overhead of stronger error correction offsets the benefit of cell shrinking. The off-chip caching and arbitrary pairing variant was used for the grouping (GRP2) scheme. Therefore it initially has the worst E_{area} value due to the on-chip selector and grouping tables, but outperforms the others at smaller cell sizes due to superior fault-tolerance.

In Figure 7 we examine E_{area} and F_{avail} as voltage is varied. This shows the same trends as Figure 6. This means that cache performance will drop significantly depending on the fault-tolerance scheme and how far volt-

Figure 6. ***E_{area}*** **in (a) and** ***F_{avail}*** **in (b) with device scaling at 1.1V (64-byte block size). Cell size is relative to the smallest considered size from Figure 2.**



(a) E_{area}



(b) F_{avail}

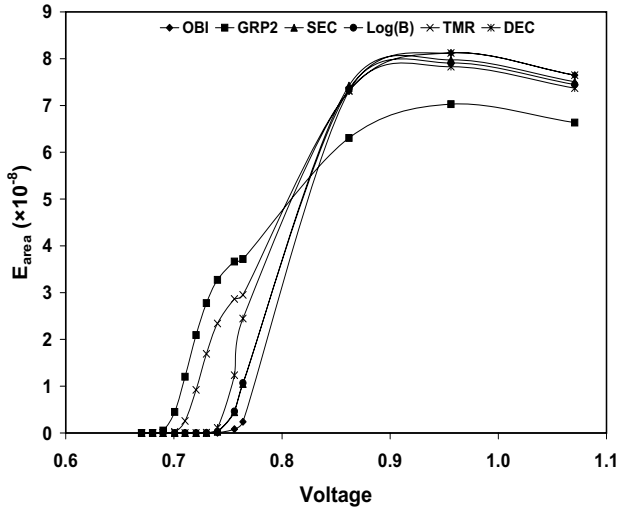
age is scaled in low-power (or low activity) modes. CPUs using DVS should dynamically select a fault tolerance scheme with the highest F_{avail} at the operating voltage. For example, Figure 7(b) indicates that DEC should be used down to 0.86V and GRP2 below that.

5.1 Energy Saving using Block Grouping at low V

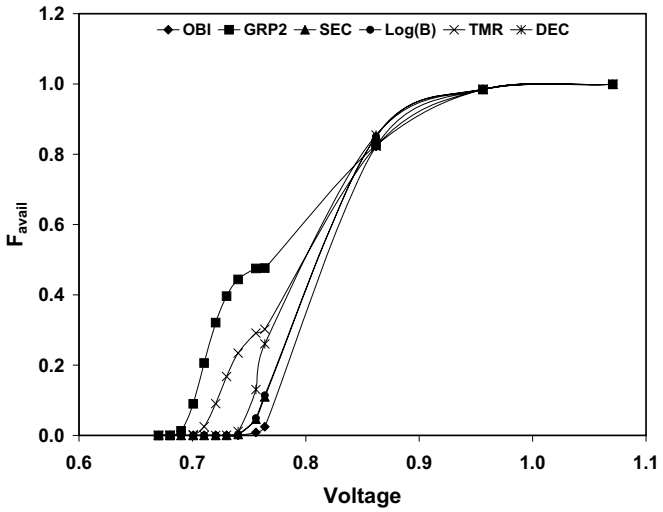
An example of the energy benefits of block grouping is as follows. In an ultra-low voltage mode of 0.76V (Figure 7(b)) conventional SEC code has an F_{avail} of around 0.02 while GRP2 is approximately 0.45. This means that GRP2 provides much more L2 cache at that voltage, reducing miss rate and improving IPC. Considering the Barnes benchmark in Figure 3, IPC for 86% disabled blocks is at most 1.14 while for grouping the IPC is 1.76 (55% disabled blocks). This means that excution using grouping completes at least (1-1.14/1.76) = 35% sooner.

Energy savings are offset by the overhead of the selector bits and grouping table. If these are on-chip, they will consume approximately 512KB (or half the cache size). Despite this overhead, there will be a net energy

Figure 7. ***E_{area}*** **in (a) and** ***F_{avail}*** **in (b) with voltage scaling (64-byte block size, Cell size C).**



(a) E_{area}



(b) F_{avail}

0-7695-2978-X/07 $25.00 © 2007 IEEE

Table 4: Grouping energy saving example.

Parameters	Value
Pentium M thermal design power	24.5 W
Selector table overhead power	2.6 W
Speedup of grouping relative to SEC	35%
Power with SEC	24.5 W
Power with grouping table	24.5 + 2.6 = 27.1 W
Average power with grouping	= 27.1 * (1 - 0.35) = 17.6 W
Energy savings with grouping	28%

saving. Using the power consumption of a 512KB, 130nm cache to represent the overhead of grouping [16], and that of a Pentium M running at around the same frequency [17] we estimate power consumption for the SEC and grouping schemes (Table 4). We use the "thermal design power" of the processor which implies the CPU is 100% utilised by the workload. In addition, we assume that the processor uses SEC fault tolerance. Even though we theoretically scale voltage down to 0.76V, both the processor and selector table have their voltages scaled by the same factor, so the power consumption ratio between the two is approximately the same.

6. Conclusions

We determined that basic OBI-based fault tolerance is the most area-efficient scheme for fault tolerance at a single voltage. However, as voltage is scaled down, maximum performance and energy savings are obtained by switching from DEC to our GRP2 scheme.

Instead of using process scale advances to reduce cell size, scaling should not go beyond a certain point. This is because the area overhead of trying to protect the smaller, but much less reliable cells is greater than that of not scaling the cells at all. However, error correction still has a place in cache design for low-voltage performance and soft error tolerance. The strength of this error correction will be a function of expected soft error rates and how aggressively DVS is applied.

References

[1] A. Agarwal, et al. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Trans. VLSI Systems*, Vol. 13(1), pp. 27~38, Jan. 2005.

[2] N. Kim, et al. Leakage current — Moore's law meets static power. IEEE Computer, Vol. 36(12), pp. 68~75, Dec. 2003.

[3] M. Agostinelli, et al. Erratic fluctuations of SRAM cache Vmin at the 90nm process technology node. *IEEE Int'l. Electron Devices Meeting (IEDM)*, pp. 655~658, Dec. 2005.

[4] A. Pour and M. Hill. Performance implications of tolerating cache faults. *IEEE Trans. on Computers*, Vol. 42(3), pp. 257~267, Mar. 1993.

[5] S. Mukhopadhyay, et al. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *IEEE Trans. on CA*D, Vol. 24(12), pp. 1859~1880, Dec 2005.

[6] D. Bossen, J. Tendler, and K. Reick. Power4 system design for high reliability. *IEEE Micro*, Vol. 22(2), pp. 16~24, Mar./Apr., 2002.

[7] A. KleinOsowski and D. Lilja. The NanoBox project: exploring fabrics of self-correcting logic blocks for high defect rate molecular device technologies. *IEEE Computer Society Annual Symp. on VLSI*, pp. 19~24, Feb. 2004.

[8] M. Nicolaidis, et al. A memory built-in self-repair for high defect densities based on error polarities. *IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI Systems*, pp. 459~466, Nov. 2003.

[9] M. Nicolaidis, N. Achouri, and S. Boutobza. Dynamic data-bit memory built-in self-repair. *Int'l Conf. on Computer Aided Design (ICCAD)*, pp. 588~594, Nov. 2003.

[10] .N. Binkert, E. Hallnor, and S. Reinhardt. Network-oriented full-system simulation using M5. *Workshop on Computer Architecture Evaluation using Commercial Workloads*, pp. 36~43, 2003.

[11] S. Woo et al. The SPLASH-2 programs: characterization and methodological considerations. *Int'l Symp. on Computer Architecture (ISCA)*, pp. 24~36. Jun. 1995.

[12] D. Lamet and J. Frenzel. Defect-tolerant cache memory design. IEEE VLSI Test Symp., pp 159~163, Apr. 1993.

[13] P. Mazumder. Design of a fault-tolerant three-dimensional dynamic random-access memory with on-chip error-correcting circuit. *IEEE Trans. on Computers*, Vol. 42(12), pp. 1453~1468, Dec. 1993.

[14] L. Joiner and J. Komo. Decoding binary BCH codes. *IEEE SoutheastCon*, pp. 67~73, Mar. 1995.

[15] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: a 32-way multithreaded Sparc processor. *IEEE Micro*, Vol. 25(2), pp. 21~29 Mar./Apr. 2005.

[16] J. Shin et al. Design and implementation of an embedded 512-KB level-2 cache subsystem. *IEEE Journal of Solid-State Circuits (JSSC)*, pp. 1815~1820, Sept. 2005

[17] Intel Pentium M power data, 1.6 GHz, *0.13*µ technology, 1 MB L2 cache *http://www.intel.com/design/intarch/pentiumm/pentiumm.htm*

0-7695-2978-X/07 $25.00 © 2007 IEEE