Salvaging Chips with Caches beyond Repair[*]

Hsunwei Hsuing, Byeongju Cha, and Sandeep K. Gupta
Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089-2562
{hsunweih, byeongjc, sandeep}@usc.edu

Abstract— ***Defect density and variabilities in values of parameters continue to grow with each new generation of nano-scale fabrication technology. In SRAMs, variabilities reduce yield and necessitate extensive interventions, such as the use of increasing numbers of spares to achieve acceptable yield. For most microprocessor chips, the number of SRAM bits is expected to grow 2× for every generation. Consequently, microprocessor chip yields will be seriously undermined if no defect-tolerance approach is used. In this paper, we show the limits of the traditional spares-based defect-tolerance approaches for SRAMs. We then propose and implement a software-based approach for improving cache yield. We demonstrate that our approach can significantly increase microprocessor chip yields (normalized with respect to chip area) compared to the traditional approaches, for upcoming fabrication technologies. In particular, we demonstrate that our approach dramatically increases effective computing capacity, measured in MIPS-per-unit-chip-area. Our approach does not require any hardware design changes and hence can be applied to improve yield of any modern microprocessor chip, incurs low performance penalty only for the chips with unrepaired defects in SRAMs, and adapts without requiring any design changes as the yield improves for a particular design and fabrication technology.***

I. Introduction

As CMOS technology continues to move deeper into nano-scale, defect rates are increasing. Furthermore, in SRAMs many parametric changes caused by increasing variabilities also manifest as bit-cell failures. As a result, the SRAM bit-cell failure rate, defined as the probability that any single SRAM cell is defective, is predicted to reach 10^{-5} for 16nm technology [1]. In fact, for 12nm technologies SRAM cells are predicted as being less robust compared with logic gates under process variations.

Every state-of–the-art processor dedicates a large fraction of chip area to SRAM caches. For example in Intel's Itanium 2 Montecito processor, more than 80% of the die area is devoted to caches [2]. Furthermore, the number of SRAM cells on microprocessor chips is predicted to grow 2× every 3.5 years after 2013 [3]. As a consequence, on-chip SRAM yield will significantly influence the yield of future microprocessor chips. Hence, adoption of defect-tolerance schemes for on-chip SRAM caches is imperative to achieve acceptable yield for microprocessor chips fabricated in the future.

Conventionally, spare memory cells are added to memories and used to replace defective cells, rows, and columns [4] [5]. Adding more spare rows or columns can potentially help increase the yield of on-chip SRAM caches. However, the area overhead caused by spares needs to be investigated carefully under the high bit-cell failure rate of the future technology. In this paper, we will first reveal the fundamental limitation of spares-based schemes. We will show that in terms of the primary economic benefit, measured by yield-per-area (YPA), beyond some point we will obtain diminishing marginal returns as we add more spare rows and columns to SRAM to combat increasing bit-cell failure rates.

Although coding techniques such as ECC can be used as a defect-tolerance scheme [6] in SRAM caches, coding techniques are typically designed to tolerate faults or errors that occur during chip's operational life, such as single- or multi-bit upsets caused by alpha particles. Using ECC for repairing manufacturing defects reduces chips' ability to tolerate soft-errors, and potentially increases failure rate during system's operational life. Hence, we do not use ECC to combat manufacturing defects.

In this paper, we propose a software-based defect-tolerance approach for targeting defect-tolerance in the unified *last-level cache (LLC);* we do not target lower level caches as these occupy much smaller chip area compared to LLC in general-purpose microprocessors. The proposed approach aims to salvage microprocessor chips with LLC defects that are beyond the repair capacity of hardware-based defect-tolerance scheme incorporated in the LLC. The proposed approach is successfully implemented by modifying operating system (OS) kernel and the booting procedure. Our preliminary implementation proves that our approach can be applied in a straightforward manner to any modern OS. Our implementation demonstrates the considerable benefits, especially for higher SRAM failure rates expected in the near future. One main qualitative advantage of our approach is that it can be applied to combat fabrication defects in most existing microprocessor designs, since it does not require any modifications to the hardware design. Another important advantage of our approach is that it causes overhead only for those chips that are fabricated with defective cache bit-cells that cannot be completely repaired using the available spares. Our evaluations will quantitatively demonstrate that the proposed approach significantly improves the usable cache size as well as computation capability per silicon area.

II. Limitations of spares-based schemes

In this paper SRAM design decisions are made to maximize YPA. SRAM designs optimized for power and performance will have slightly different limitations on the number of spares that can be used without adversely affecting YPA (more ahead). Hence, the specific numbers derived from the case study in this section should not be arbitrarily compared with other SRAM designs. However, the limitations of different designs do not differ by large amounts since the limitations are fundamental in nature. In this paper, we focus on defects in (1) the SRAM bit-cells, and (2) the interconnect wires. The interconnect yield is derived via critical area analysis [7]. Area is estimated using a version of CACTI [8], a widely used model for characterizing most metrics of a cache design, enhanced to consider spares required for defect-tolerance.

*This research was supported by National Science Foundation

A. *Adverse effect of spares*

The spares-based scheme causes the following area overheads: the area of the spares and reconfiguration circuitry, and the area needed to fit the wire track of reconfiguration circuitry into the pitch of each SRAM row or column. To be able to replace any row containing one or more defective bit-cells with N_{sr} spare rows, a wordline driver fan-outs to the row it drives as well as neighboring N_{sr} rows through a de-multiplexor. The height of a row will be the maximum of the height of a de-multiplexor's output wire track or the height of a SRAM bit-cell. It is imperative that the re-configuration circuitry for each row to fit within the height of the row, since otherwise the SRAM density will decrease dramatically. Similar situation occurs for spare columns. Hence as the number of spare rows and columns increases, the area of the SRAM increases non-linearly. In our analysis and experiments using CACTI [8] assuming the tightest wire pitch allowed, only one spare row and one spare column can be added to a sub-array without dramatically increasing the overall area.

Case study:

A 3MB 12-way associative cache with 64-byte blocks is used as a case study throughout the paper. The cache is divided into 48 equal-size sub-arrays. Each sub-array is equipped with N_{sr} spare rows and N_{sc} spare columns. The YPA of the cache is plotted in Fig. 1 for various numbers of spares rows and columns. The yield calculation will be presented in Section IV. CACTI is enhanced to include the area overheads of spares. The area of the cache is then obtained for a 32nm technology. The SRAM bit-cells are assumed to have a failure rate of 10^{-6} [1]. It can be seen that the YPA value reaches its maximum value at $N_{sc} = 2$ and $N_{sr} = 1$ and decreases if higher numbers of spare rows and columns are used.

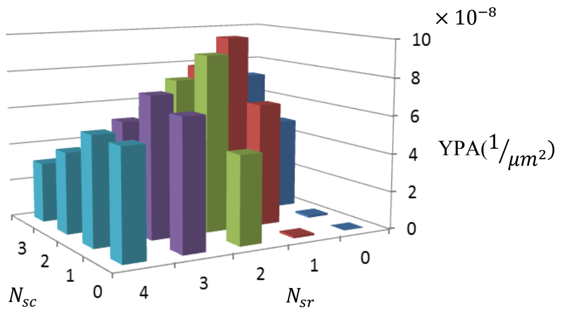


Figure 1. YPA of the 3MB cache with various number of spares

B. *Dividing into smaller sub-arrays*

By dividing the SRAM into smaller sub-arrays, we can increase the total number of defective bit-cells that can be repaired without using too many spares in each sub-array. However, increasing the number of sub-arrays <u>also</u> increases the interconnect wires in the array, and the yield of such design is often limited by the interconnect wires. By increasing the number of spares (N_{sr}, N_{sc}) for different number of sub-arrays (N_{SA}), we calculate the YPA of each configuration until the YPA trend starts falling. Fig. 2 shows the numbers of spare rows and columns that maximize YPA for different numbers of sub-arrays. Note that the highest achievable YPA drops as the number of sub-arrays increases beyond a certain level. Because interconnect wires grow with the number of sub-arrays, the overall yield is limited by the interconnect yield rather than sub-array yield for designs with large numbers of sub-arrays. Besides, the overall area also grows thus YPA decreases further.

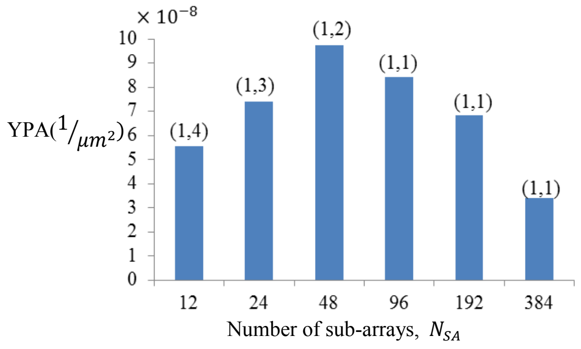


Figure 2. Spares allocation ($\boldsymbol{N_{sr}, N_{sc}}$) that maximize YPA for ent $\boldsymbol{N_{SA}}$, for failure rate of $\mathbf{10^{-6}}$

TABLE I shows the optimal YPA can be achieved by varying (N_{SA}, N_{sr}, N_{sc}) of the SRAM cache under failure rates predicted for future technologies [1]. The optimal YPA drops 42% and the yield is only 57% for this YPA-optimized design under the higher failure rate. The analysis clearly shows that by using only spares, low YPA and low yield will be expected as the failure rate increases. Clearly, more advanced defect-tolerance schemes must be adopted to achieve high YPA and yield in the future.

TABLE I. HIGHEST YPA ACHIEVED BY OPTIMAL SRAM DESIGN

Failure rate	(N_{SA}, N_{sr}, N_{sc})	Yield	Optimal YPA
10^{-6}	(48, 1, 2)	71.2%	9.74×10^{-8}
2.5×10^{-6}	(96, 1 ,3)	57.1%	5.69×10^{-8}

III. PROPOSED APPROACH AND RELATED SCHEMES

A. *Proposed Approach: OS-Based Defect Tolerance*

The basic idea behind our approach is to actively avoid using the unrepaired bit-cells. Available spare rows and columns are used to repair a maximum number of defective bit-cells. For some chips this repairs all defective bit-cells, while in other chips some defective cells cannot be repaired. Our approach salvages the defective chips beyond the hardware repair capacity by not accessing memory locations containing unrepaired bit-cells. Our goal is to develop an approach that causes least hardware overhead and depends as little as possible on the circuit implementation, so our approach can be applied more universally.

Fig. 3 illustrates the main idea behind our approach. SRAM cache and DRAM main memory are shown in logical view and each block of the DRAM has the same size as a cache block. Each set of N_S contiguous DRAM blocks constitutes a *cache region*. Since the number of blocks in each cache region is N_S, there is a one-to-one mapping relation between blocks in one cache region and all sets in the cache. "×" denotes a single defective block in the cache and the stars mark the DRAM blocks which may be corrupted when cached into the defective cache location marked by ×. If we can avoid using every one of the starred blocks in the DRAM, the defective block in the cache will not be accessed. Since LLC and DRAM locations are accessed using physical addresses, we can only avoid *page frames* in the part of the system that works with physical addresses. In the modern OSs with virtual memory, a physical *page frame* is the granularity at which the memory management unit of an OS can access DRAM.

DRAM is divided into a numbers of page frames. The shaded sections in the DRAM represent *page frames,* and each consists of a number of blocks. The shaded section in the cache represents a *page cover* which is a continuous region of the cache to which a *page frame* maps. A *page cover* is defective if it contains one or more defective blocks. The OS allocates the frames to software processes on demand. We can avoid using the *page frames* containing the starred DRAM blocks by modifying the physical memory management. The cache will become useable with reduced capacity as we avoid the use of defective *page cover*. The percentage of the disabled memory size is the same in both SRAM and DRAM.

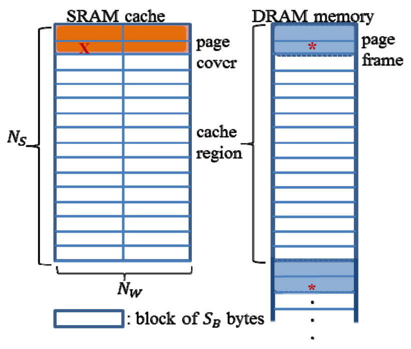


Figure 3. Logical view of SRAM cache and DRAM

The idea of avoiding the use of defective page cover can also be implemented in TLB by changing the address translation to avoid using the defective *page covers.* However, the OS level approach clearly has lower hardware overhead. Hence we choose to implement the approach at the OS level. Under the predicted failure rates, the organization of the lower level caches in today's microprocessor chips makes the application of the approach to the lower level cache impractical due to the high reduction in memory size.

Similar to other related schemes to be addressed later, our approach assumes that testing provides the information about defective cache locations. When OS boots, we instruct the OS to reserve every *page frame* that is mapped onto defective cache locations. Subsequently, the OS meets all requests for memory from processes by allocating only the unreserved pages. Since *page frames* that are mapped to defective locations are never assigned to any process or bound with any virtual page, the defective cache locations will never be accessed. A *page cover*-disabling scheme is hence achieved. The proposed approach has been implemented by modifying Linux 2.6.32 kernel. The modified OS boots-up successfully and can operate without any exceptions or crashes. We also check the output values for benchmark applications to assure correctness.

B. *Related Hardware Schemes*

Hardware-based reconfiguration approaches can be categorized into two: *remapping* and *disabling* schemes. Remapping refers to schemes which modify the decoders to achieve dynamic configurability. In [9], a programmable decoder is proposed to remap any access to defective blocks in the cache to healthy ones. The decoder is programmed to implement different mapping functions based on the information in a defective-bit-cell map. Disabling refers to schemes that bypass the defective cache locations during accesses. A defective-bit-cell map can be integrated into the cache tag array in the form of additional valid bits, which are called *availability bits* [10] or *fault bits (FTB)* [11]. A cache hit occurs only if the tag matches, the valid bit is set, **and** the FTB for the target location indicates healthy block. Data written to the cache cannot be placed in blocks with a FTB indicating defective block; instead, the data is placed into other healthy blocks in the same set or is written directly to a higher level cache or DRAM. Disabling can be applied at different granularities, namely blocks, sets, and ways. Way-disabling is currently implemented in a mass market microprocessor chip for power management [12]. A similar disabling mechanism to combat high bit-cell failure rate in future technologies can be designed in a straightforward manner. Hence we will compare our approach to way-disabling in later sections.

Compared to these hardware-based schemes, our approach does not need hardware reconfiguration circuit and does not require any modification to existing cache designs. These facts set our approach apart from other schemes, since (1) the reconfiguration circuit adds area and latency overhead to defective chips as well as healthy chips, and (2) the reconfiguration circuit is also susceptible to defects.

C. *Proposed Metrics*

Way-disabling scheme and our OS-based approach both prevent the system from using the defective parts of the cache. We propose a new metric, *expected capacity per area (EC-per-area)*, to measure the effectiveness of these schemes. *EC* is the measure of the percentage of all cache bits that can be used. Chips with the same number of usable bits are assigned to a *grade*, denoted by g_x , where x indicates that there are x defective *page covers* or ways to be avoided in the chips of this grade.

$$EC = \sum_{x=0}^{N_G-1}(1-\frac{X}{N_G}) \times P(g_x) \qquad (1)$$

where, $P(g_x)$ denotes the percentage of grade g_x chips among all the chips manufactured, N_G is the total number of the disabling granules, due to the proposed *page cover*-disabling or the way-disabling (used as a baseline for comparison) in the cache. *EC-per-area* of a configuration (N_{SA}, N_{sr}, N_{sc}) would be the *EC* divided by the area of the configuration. In order to measure the benefits of our approach in term of computing capability (performance), *expected million-instruction-per-second per area (EMIPS-per-area)* is derived from experiments and area data.

$$EMIPS = \sum_{x=0}^{N_G-1}(MIPS_x) \times P(g_x) \qquad (2)$$

where, $MIPS_x$ is the million-instruction-per-second for grade g_x measured by simulating benchmark applications.

D. *Operational and Product Delivery Model*

Our approach personalizes each fabricated microprocessor chip by binning chips into finer grades, since we can mask the defects at a finer granularity. The approach assumes that LLC testing is capable of identifying and recording the locations of unrepaired bit-cells in each fabricated copy of the microprocessor chip. When booting up, the bootstrapping software further translates the locations of unrepaired bit-cells in its LLC into locations of defective *page covers*. The bootstrapping software then relays the locations of defective *page covers* to the OS. In the above mentioned process, the location information of unrepaired bit-cells must be delivered to microprocessor users/vendors. Such information can be delivered in many possible ways. For example, the location information can be encoded into processor identification registers. Bootstrapping software can then read the registers to identify defective *page covers'* locations. In our implementation for experiments ahead, the defect locations are coded directly as OS kernel parameters and sent to the bootstrapping software.

IV. Analysis of Proposed Approach

A. Sub-array organization

Fig. 4 illustrates the relation between logical granules (*page covers* and ways) and the underlying sub-array arrangement of a 3MB cache realized using 6 sub-arrays and is used to explain our terminology. The distribution of unrepaired bit-cells in a physical sub-array is derived first and used to derive the distribution of defective logical granules for the proposed approach.

In the 3MB cache of the case study, each 4KB *page frame* is direct-mapped to 64 continuous sets. The continuous 64 sets mapped by a *page frame* constitute a non-overlapping *page cover* and there are total 64 *page covers* in the cache. Each *page cover* spans 3 sub-arrays, and a group of 3 sub-arrays enveloped by a same set of *page covers* is called a *sub-array group (SAG)*. The number of sub-arrays in an SAG (N_{gSA}) is 3. The number of *page covers* in an $SAG (N_{SAGt})$ is 32. The number of SAG (N_{SAG}) is 2. Similarly, when adopting the way-disabling scheme, the sub-arrays in a non-overlapping disabling granule form an SAG. In this specific case, $N_{gSA} = 2$, $N_{SAG} = 3$, and $N_{SAGt} = 4$. *SAG* and N_{SAGt} are defined by the defect-tolerance scheme used as well as sub-array configurations of the SRAM array.

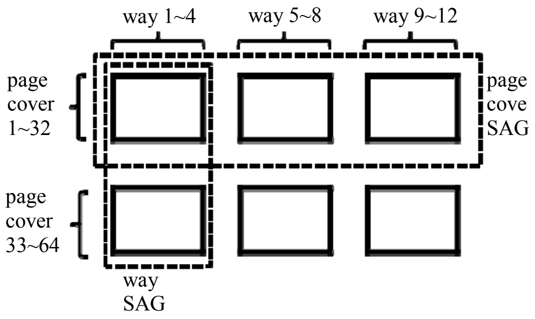


Figure 4. Mapping of *page covers* and ways onto 6 sub-arrays

B. Formulations

Sub-array level:

The probability that a sub-array has total x defective bit-cells, $P_A(x)$, is derived assuming uniform distribution of the defective bit-cells.

$$P_A(x) = {(p \times N_{bit})^x e^{-(p \times N_{bit})}}/{x!} \tag{3}$$

where, p is the failure rate of a bit-cell, N_{bit} is the total number of SRAM bit-cells including spares in a sub-array. The probability that a sub-array has n unrepaired bit-cells, $P_{SAu}(n)$, is calculated as below,

$$P_{SAu}(n) = P_A(n + N_{sr} + N_{sc}) \times \prod_{i=1}^{n+N_{sr}+N_{sc}} \frac{N_{tc} \times N_{tr} - (i-1)(N_{tc}+N_{tr}-1) + (\sum_{j=i-2,\forall i \in N: i>2}^{i} 2j)}{N_{tc} \times N_{tr} - (i-1)} \tag{4}$$

where, N_{tc} and N_{tr} are the total number of columns and rows respectively, including spares. The second term represents the probability that there are $n + N_{sr} + N_{sc}$ un-aligned defective bit-cells in any row/column. It is possible that the available spares may be able to repair all–but n defective bit-cells even if more than $n + N_{sr} + N_{sc}$ cells are defective thus $P_{SAu}(n)$ value we calculate is a lower bound.

Yield of spare rows and columns scheme:

The yield of an SRAM array with N_{SA} sub-arrays is estimated as follows.

$$Y^{N_{SA}} = \{\sum_{i=0}^{N_{sr}+N_{sc}} P_A(i) + \sum_{n=1}^{\infty} P_A(n + N_{sr} + N_{sc}) \times (1 - \prod_{i=1}^{n+N_{sr}+N_{sc}} \frac{N_{tc} \times N_{tr} - (i-1)(N_{tc}+N_{tr}-1) + (\sum_{j=i-2,\forall i \in N: i>2}^{i} 2j)}{N_{tc} \times N_{tr} - (i-1)})\}^{N_{SA}} \tag{5}$$

where, Y is the yield of a sub-array. Equation (4) neglects the contribution of the possibilities that more than $n + N_{sr} + N_{sc}$ defective bit-cells can result in n unrepaired bit-cells. However, in (5) these are counted towards the yield of the baseline spares-based scheme. Thus in all our calculations ahead, we overestimate the yield of the baseline spares-based scheme and hence underestimate the percentage of additional chips saved by our approach.

OS-based approach:

The probability that an *SAG* has n unrepaired defective SRAM cells is represented as below

$$P_{SAGu}(n) = \sum_{i=1}^{N_{PW}(n)} \left(\frac{\prod_{k=0}^{N_{SAG}-1} f(k, Mn\_N_{gSA}\_i(x))}{\prod_{j=0}^{n} Mn\_N_{gSA}\_i(j)!}\right),$$

$$\text{where,} \quad f(k,x) = \begin{cases} P_{SAu}(x) \times (N_{gSA} - k) & if\ x \neq 0 \\ P_{SAu}(0) & if\ x = 0 \end{cases} \tag{6}$$

$N_{PW}(n)$ is the number of all possible ways of partitioning n into N_{gSA} sub-arrays, $Mn\_N_{gSA}\_i(j)$ is the number of sub-arrays with exactly j bit-cells in the i^{th} case of n distributed into N_{gSA} sub-arrays, and $Mn\_N_{gSA}\_i(0) = 0$. Given that an *SAG* has a total n unrepaired bit-cells, there are several ways in which these n unrepaired bit-cells are distributed among N_{SAGt} *page covers*. $P_{SAGf}(n_{gf})$, the probability that there are n_{gf} defective *page covers* in an *SAG* can be computed as follows.

$$\sum_{i=n_{gf}}^{S} P_{SAGu}(i) \times \frac{\binom{i-1}{i-n_{gf}}\binom{N_{SAGt}}{n_{gf}}}{\sum_{j=1}^{\min(N_{SAGt},\ i)} \binom{i-1}{i-j}\binom{N_{SAGt}}{j}} \tag{7}$$

where, S is the range of number of unrepaired bit-cells to be considered. The probability that there are total n_{Af} defective *page covers* is calculated in a manner similar to equation (6) as:

$$P(n_{Af}) = \sum_{i=1}^{N_{SAG}(n)} \left(\frac{\prod_{k=0}^{N_{SAG}-1} f(k, Mn\_N_{SAG}\_i(x))}{\prod_{j=0}^{n} Mn\_N_{SAG}\_i(j)!}\right),$$

$$\text{where,} \quad f(k,x) = \begin{cases} P_{SAGf}(x) \times (N_{SAG} - k) & if\ x \neq 0 \\ P_{SAGf}(0) & if\ x = 0 \end{cases} \tag{8}$$

$N_{SAG}(n)$ is the number of all possible ways of partitioning n into N_{SAG} *SAG*, $Mn\_N_{SAG}\_i(j)$ is the number of *SAG* with exactly j defective *page covers* in the i^{th} case of n distributed into N_{SAG} *SAG*, and $Mn\_N_{SAG}\_i(0) = 0$.

Way-disabling scheme:

The yield of an SAG is represented as below.

$$Y_{SAG} = Y^{N_{gSA}} \tag{9}$$

The probability that there are total n_{AW} groups of ways are defective is calculated as below.

$$P(n_{AW}) = \binom{N_{SAG}}{n_{AW}} (1 - Y_{SAG})^{(n_{AW})} \times Y_{SAG}^{(N_{SAG}-n_{AW})} \tag{10}$$

The four ways are disabled as a group at a time [12]. Using equations (1) and (2) and area derived using enhanced CACTI, we calculate *EC* and *EC-per-area* and compare the benefits of our approach with way-disabling and spares-based baseline schemes. The results are presented in the next section.

V. Performance Evaluation using Implementation

A. Implementation of our OS-based approach

We implement our approach by modifying Linux 2.6.32 kernel. By sending customized early kernel parameter commands via the GRand Unified Bootloader (GRUB) during system boot-up, we reserve physical *page frame*s mapped to locations in LLC that are identified as being defective. This implementation is used for performance evaluation by running SPEC CPU2000 benchmarks [13] on a system with a dual core processor and 1GB of DRAM. Each core of the processor has one 8-way 64KB L1 data cache and one 8-way 64KB L1 instruction cache. Two cores share a 12-way 3MB unified L2 cache (LLC). Disabling different numbers of *page covers* in a real machine emulates microprocessor chips of different LLC grades. Since OS does not allow us to control cache ways, we do not compare our approach to way-disabling in terms of *EMIPS-per-area*.

B. Evaluation Results

TABLE II shows the configurations (N_{SA}, N_{sr}, N_{sc}) of the 3MB cache with three different scheme combinations optimized for *EC-per-area* under different failure rates expected in upcoming technologies [1]. *EC-per-area of* spares-based scheme is equal to its YPA. TABLE II also shows *EC* and *EMIPS* data as original values and in normalized form against the values for spares-based scheme.

TABLE II. Comparison of three combinations of the schemes under different failure rates (FR)

		Spares-based	Way-disabling & spares		*Page cover*-disabling & spares	
			Value	Norm.	Value	Norm.
FR= 10^{-6}	(N_{SA}, N_{sr}, N_{sc})	(48, 1, 2)	(48, 1, 1)		(12 , 1, 1)	
	EC	71.2%	61.5%	0.86	78.9%	1.11
	EC-per-area $(\times 10^{-8})$	9.7	10	1.12	14	1.51
	EMIPS	3288.1	N/A	N/A	3749.5	1.14
	EMIPS-per-area $(\times 10^{-4})$	4.5	N/A	N/A	6.9	1.55
FR= 2.5 $\times 10^{-6}$	(N_{SA}, N_{sr}, N_{sc})	(96, 1 ,3)	(96, 1, 2)		(48, 1, 1)	
	EC	57.1%	53.1%	0.93	69.6%	1.22
	EC-per-area $(\times 10^{-8})$	5.6	7.0	1.23	12	2.18
	EMIPS	2639.2	N/A	N/A	3619.5	1.37
	EMIPS-per-area $(\times 10^{-4})$	2.6	N/A	N/A	6.7	2.56

We obtain optimal designs by enumerating various values of (N_{sr}, N_{sc}) for different numbers of sub-arrays (N_{SA}), for each of the three schemes. Way-disabling and our approach both provide higher *EC-per-area* compared to when only spares scheme is used. Our approach achieves the highest *EC-per-area* by using configuration which requires significantly less area. Compared to way-disabling, our approach achieves 34% and 77% higher *EC-per-area* for the two failure rates. Although we cannot obtain *EMIPS* for the way-disabling scheme, it is clear that our approach will provide higher *EMIPS-per-area* since our approach provides much higher *EC-per-area*.

C. Benchmark Results

Fig. 5 shows the increase in LLC miss rates for microprocessors in various grades measured by Oprofile [14]. It is observed that LLC size reduction has different impacts on different benchmarks. We estimate LLC utilization for each benchmark using *sim-cache* [15] to confirm these differences. Fig. 6(b) shows the simulation results for benchmarks that have significant increase in LLC miss rate during the execution on the actual machine.

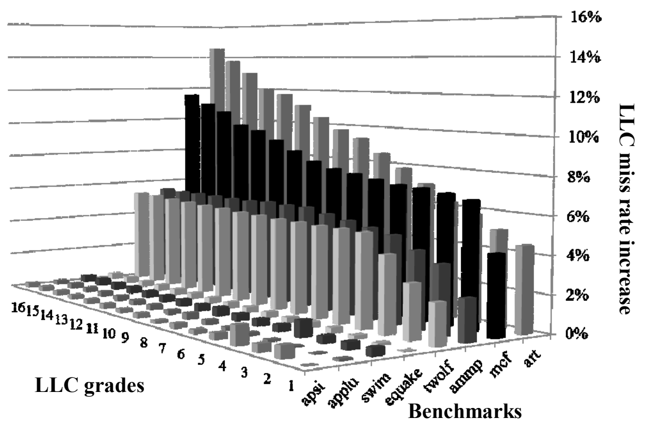
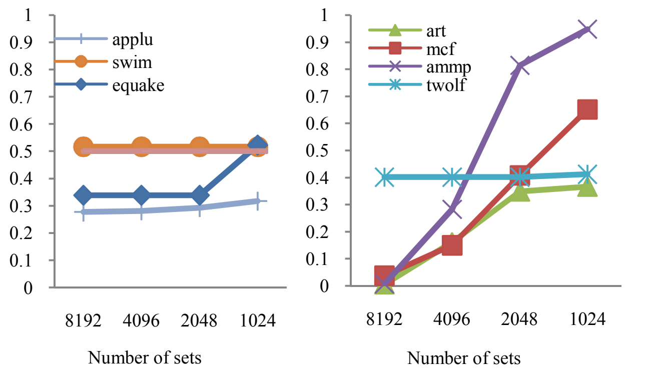


Figure 5. LLC miss rate increase for systems with different grades of LLC

The trends in Fig. 5 and Fig. 6 agree except for *twolf*. The machine on which we perform our experiments has equivalent of 4096 sets in Fig. 6. As the number of sets decreases, *art*, *mcf,* and *ammp* show significant increases in miss rates both in simulations and in real machine benchmarking. For other benchmarks the LLC utilization is already low, so the reduction in the LLC size during execution shows no significant change in miss rate. Besides the verification from the system boot-up logs and comparison of benchmark outputs, the similarity of LLC miss behavior also indirectly verify that our implementation indeed reduces the usable LLC size.



(a) Applications with small increase in LLC miss rate

(b) Applications with large increase in LLC miss rate

Figure 6. LLC miss rate change over different number of sets

VI. Disscussion

A. Advantages

The fact that our approach adapts to the outcome of fabrication and yet does not require any changes to the hardware design constitutes a significant benefit. One consequence is that as yield learning improves bit-cell failure rate for a given design, our approach does not require any changes to the cache design to achieve optimality for the new failure rate, thus imposes zero delay and area penalties in microprocessor chips manufactured in mature technology. In contrast, microprocessors with only hardware-based schemes either need to be re-designed to achieve optimal YPA or volume-manufactured with a non-optimal YPA once the technology matures.

We have shown that adopting our approach in conjunction with spare rows and columns scheme can further increase the amount of usable cache. The fact that our approach is able to salvage defective chips beyond hardware repair capability also makes it an excellent addition to any hardware defect-tolerance scheme. In addition, this flexible nature can also benefit the usage of microprocessor chips under various operating conditions, since the numbers and locations of defects might change under different voltage, temperature, age, and so on. Availability of built-in self-test (BIST) [16] can enables this approach to also combat failures that occur throughout a microprocessor's operational life as well as under harsh conditions.

B. Cost of the approach

DRAM capacity loss:

Operating with chips with defective *page covers* comes with the price of sacrificing some amount of DRAM. In our experiments with the 3MB cache, under the higher failure rate the expected percentage of DRAM that can be used is 68%. However, we must emphasize that our approach targets to salvage chips beyond hardware repair. The systems that suffer from less effective DRAM are actually using microprocessor chips that would have been discarded. Comparing the prices of microprocessor chips and DRAM chips, this is indeed an attractive tradeoff.

DRAM fragmentation:

The presence of unrepaired bit-cell fragments the physical memory space. Under the bit-cell failure rate and LLC organizations expected in the near future, the expected value of the number of contiguous available locations in the physical memory space is much larger than the normal *page* size, namely 4KB in many operating systems, but smaller than the larger page size, e.g., *hugepage* in Linux*,* which is typically 2MB or more. The default option in most operating systems is to use normal *pages* and allocate physical space to programs - even to data structure such as extremely large arrays – one *page* at a time. Hence, in most cases the fragmentation in the physical address space caused under our approach by unrepaired bit-cells does not change how the data is stored and accessed. Consequently, by itself, fragmentation of physical address space typically does not cause any performance penalty. In contrast, a few specialized systems use *hugepage* to improve program performance [17]. We evaluate the performance loss due to the incapability of using *hugepage* by supporting the benchmarks using *libhugetlbfs* library. Out of eight benchmarks, *hugepage* improves the performance of five benchmarks. On average, hugepage improves the performance of these five benchmarks by 9.2%. However, the applicability of the proposed approach is not limited, since this feature is not commonly used in most general purpose systems. Also, such fragmentation penalty will only occur on copies of a specialized system that use salvaged chips; i.e., penalty will remain zero for copies using non-salvaged chips.

VII. CONCLUSION

We have proposed a software-based LLC defect-tolerance approach to overcome the high failure rates expected for SRAMs manufactured using future fabrication technologies. Our approach can salvage a dramatically high proportion of microprocessor chips fabricated with unrepaired bit-cells in their LLC. This is especially beneficial when a fabrication process is still immature. Our approach requires no circuit modification thus incurs zero circuit-level delay overhead and zero area overhead. Consequently, our approach can provide higher performance-per-$ than existing hardware based schemes. The flexibility of our approach can makes it useful in conjunction with any hardware defect-tolerance scheme, as well as enables field programmability of general purpose processors under challenging operating conditions.

REFERENCES

[1] "International Technology Roadmap for Semiconductors," http://public.itrs.net/.

[2] B. Amelifard, "Power efficient design of SRAM arrays and optimal design of signal and power distribution networks in VLSI circuits," *Ph.D. Dissertation, University of Southern California*, 2007.

[3] K. Jeong and A. B. Kahng, "A power-constrained MPU roadmap for the International Technology Roadmap for Semiconductors (ITRS)," in *SoC Design Conference (ISOCC)*, November 2009, pp. 49–52.

[4] M. Horiguchi, "Redundancy techniques for high-density DRAMs," in *2nd Annual IEEE International Conference Innovative Systems Silicon*, 1997, pp. 22-29.

[5] S. E. Schuster, "Multiple word/bit line redundancy for semiconductor memories," *IEEE J. Solid-state Circuits*, vol. 13, no. 5, pp. 698-703, October 1978.

[6] H. Sun, N. Zheng, and T. Zheng, "Realization of L2 Cache Defect Tolerance Using Multi-bit ECC," in *IEEE Intl' Symposium on Defect and Fault Tolerance of VLSI Systems*, 2008.

[7] J. C. Cha and S. K. Gupta, "Characterization of granularity and redundancy for SRAMs for optimal yield-per-area," in *IEEE International Conference on Computer Design*, 2008, pp. 219-226.

[8] Thoziyoor et al, "CACTI 5.1," HP Lab. , Tech. Rep. HPL-2008-20, April 2008.

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

[10] G. S. Sohi, "Cache memory organization to enhance the yield of high performance VLSI processors," *IEEE Trans. Computers*, vol. 38, pp. 484-492, April 1989.

[11] D. A. Patterson, P. Garrison, M. Hill, D. Lioupis, C. Nyberg, T. Sippel, and K. Van Dyke, "Architecture of a VLSI instruction cache for a RISC," *the 10th annual international symposium on Computer Architectur*, vol. 11, pp. 108-116, June 1983.

[12] V. George, et al, "PENRYN: 45-nm Next Generation Intel® Core™ 2 Processor," in *ASSCC Dig. Tech. Papers*, 2007.

[13] J. L. Henning, "SPEC CPU2000: measuring cpu performance in the new millennium," *IEEE Computer*, vol. 33, no. 7, pp. 28-35, July 2000.

[14] J. Levon, "Oprofile - a system profiler for linux," Technical report http://oprofile.sourceforge.net/.

[15] D. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0.," Comput. Sci. Dept., Univ. Wisconsin-Madison, Tech. Rep. 1342, Jun.1997.

[16] M. H. Tehranipour, Z. Navabi, and S. M. Fakhraie, "An efficient BIST method for testing of embedded SRAMs," *IEEE Int. Symp. Circuits and Systems*, vol. 5, pp. 73-76, Map 2001.

[17] E. Shmueli, G. Almasi, J. Brunheroto, J. Castanos, G. Dozsa, S. Kumar, and D. Lieber, "Evaluating the effect of replacing CNK with Linux on the compute-nodes of Blue Gene/L," in *ACM International Conference on Supercomputing*, 2008.