ABSTRACT

Process variability in future technology nodes is expected to severely limit the benefits from dynamic voltage scaling. To keep power at bay, low voltage operation has been proposed. Because of the cubic relation between voltage,frequency and power, when operating at low voltage, significant power and energy savings can be achieved. However, at this mode of operation some devices fail. SRAM cells, used to built caches, are the most sensitive to low voltage operation since they are built with minimal geometry in order to save area. As a result, when operating at low voltage large numbers of faults occur in the caches. Traditional reliability techniques, such as sparing and ECC, are unable to deal with large numbers of faults. Because of this, novel reliability mechanisms have been proposed that are able to protect caches in high fault rate scenarios. However, most of these techniques are costly in terms of area or very complex to implement.

In this work we provide a new approach for dealing with high fault rates in caches. We propose to use a simple, well known reliability technique - block disabling - and combine it with performance enhancing mechanisms, such as prefetching and victim caching, and with careful selection of cache parameters such as block size and associativity. This approach is easy to implement since it uses technology that already exists in modern processors and requires little area overhead.

To select the optimal cache configuration for block disabling we use a combination of probability analysis and accurate performance simulation. Using probability analysis we show that a smaller block size is preferable since more cache capacity is available(72% for a 32B cache over 54% for a 64B cache). Also, we show that by using a smaller block size and higher associativity, the probability of having clustered faults in the same set is reduced. Using simulations we show that the capacity benefit from the smaller block size is beneficial to performance. Furthermore, we show that prefetching and victim caching can be useful to reduce performance losses caused by

faults. The victim cache is especially useful for reducing performance non-determinism caused by the random placement of faults in the cache.

Our best performing block disabling configuration is shown to outperform word disabling/bit-fix, a recently proposed mechanism for low voltage operation, by 7%. Furthermore, our low-cost block disabling configuration performs similarly to word disabling/bit-fix, requires less area overhead and is simpler to implement.

CACHE RELIABILITY FOR LARGE NUMBERS OF PERMANENT FAULTS

Nikolas Ladas

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

December, 2010

APPROVAL PAGE

Master of Science Thesis

CACHE RELIABILITY FOR LARGE NUMBERS OF PERMANENT FAULTS

Presented by

Nikolas Ladas

Research Supervisor ______________________
Research Supervisor's Name

Committee Member ______________________
Committee Member's Name

Committee Member ______________________
Committee Member's Name

University of Cyprus

December, 2010

ACKNOWLEDGEMENTS

CREDITS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 High Fault Rates in Caches

The goal of this work is to provide reliable cache operation when operating in the presence of many permanent-like faults. These conditions can occur due to increased process variation in future technology nodes(e.g. 12nm process) or when operating below the minimum voltage(Vccmin) where correct operation is ensured. We describe these two scenarios in more detail in the following subsections.

1.1.1 Static Process Variation in Future Technology Nodes

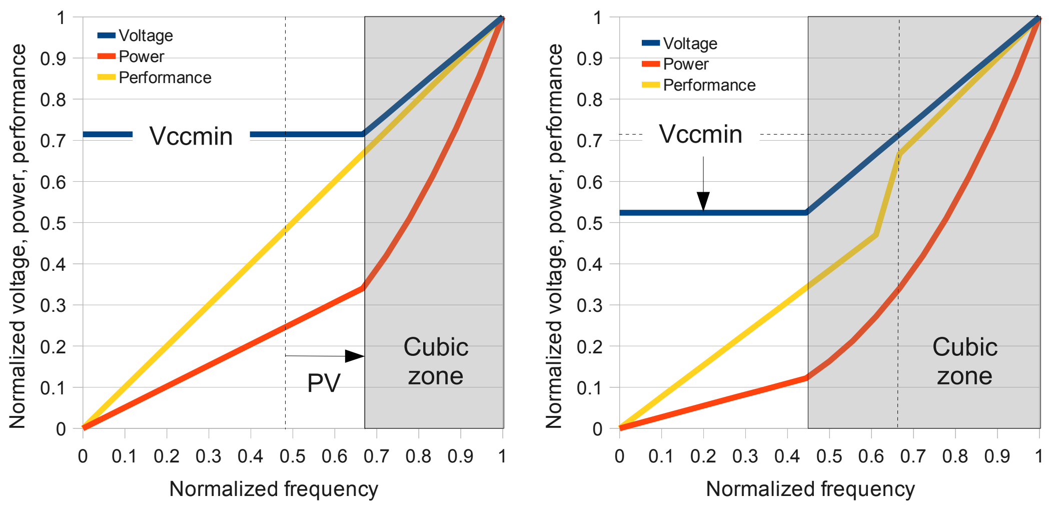
Due to imperfections in the manufacturing process, on chip devices(wires, transistors etc) may have different physical and operational properties than their design specications [4, 7, 5, 3]. For example, the length, width, oxide thickness and doping of a transistor may vary. These variations can occur at different granularity [6] wafer-to-wafer, die-to-die and within-die. Large scale variations, across wafers and between dies, are more predictable and can be addressed with manufacturing tuning or post-manufacturing techniques like body-biasing [7, 5]. The most challenging variations are within-die variations that occur either as systematic or random [4, 7, 3, 6]. Systematic within-die variations can be addressed using solutions capable of tuning differently parts of the same chip [18, 31, 26]. Random process variations can occur at very fine granularities. For example, neighboring transistors may have completely different supply voltage requirements.

These effects are especially pronounced for caches [19] since they are built with minimal geometry in order to maximize area efficiency. Random process variations can be handled by setting the supply voltage to the highest value required by the slowest device on the chip. Additionally, voltage guard bands must also be applied in order to address voltage, frequency and temperature fluctuations that may appear during operation.

As devices continue to scale, this approach becomes infeasible. In future technology nodes process variation phenomena are expected to become more pronounced [30, 19]. Increasing the supply voltage to compensate for process variations will cause unrestrained increase in power. Because power must be kept in check, supply voltage has to be constrained to acceptable values. As a result, some devices will fail due to process variation. To sustain technology scaling, future processors will need to be able to handle large numbers of faults caused by process variation.

1.1.2 Below Vccmin Operation

Below Vccmin operation [32] is a technique that can be used to reduce power and save energy. This technique allows the supply voltage to drop below the value where correct operation is guaranteed(Vccmin) so that more power savings can be achieved. However, because of process variation, some devices will fail at this voltage. As a result, reliability methods must be applied in order to ensure reliable operation. The technique is applied to caches since they occupy a large portion of the chip area and are a major contributor to static and dynamic power consumption.



(a) Voltage scaling

(b) Voltage scaling below Vcc min

Figure 1: Voltage Scaling vs Power and Performance

Figure 1(a) illustrates the relation between voltage, power, frequency and performance. As voltage decreases, cubic reductions in power occur. This reduction, however, stops when Vccmin is reached. Because of process variation, the zone of cubic power reductions(marked gray in the figure) is reduced as Vccmin is forced to remain at a value where correct operation is ensured. Figure 1(b) shows the effects of scaling voltage below Vccmin. Again the figure illustrates the relation between voltage, power, frequency and performance. By scaling voltage below Vccmin, we achieve further reductions in power(the cubic zone is extended). However, to ensure reliable operation when operating below Vccmin, reliability techniques must be applied. Reliability techniques have overheads which can translate to degraded performance. This is illustrated in the figure by the sharp drop in performance when the low voltage region is entered. For voltage scaling to be effective, the performance cost of operating below Vccmin must be kept as low as possible.

In the following section we describe our approach for ensuring reliable operation under the presence of faults in caches.

1.2 Our Approach

Traditional reliability techniques such as spare parts or error correcting codes are not suited for scenarios where faults are numerous. When operating below Vccmin, for example, hundreds of faults could be present in the cache. As a result, spares cannot be used since the area cost will be prohibitive. At the same time, when many faults are present, the probability of multiple errors in the same region increases. Multi-bit errors cannot be handled by error correcting codes unless the number of code bits is increased which results in large area overhead. Furthermore, by using error correcting codes to correct permanent faults, soft error detection and correction is compromised.

To address the reliability problems that occur due to process variation or from below Vccmin operation, novel reliability techniques have been proposed that are able to handle large numbers of faults. Like our method, most of these techniques focus on cache reliability since caches are the components most likely to be affected by process variation. However, as we will see in the following chapter, currently proposed techniques have significant shortcomings:

- Require significant area/power overhead
- Require extensive modifications to the baseline cache operation
- Too complex to implement on real hardware
- Do not take important cache hierarchy parameters into account(block size, associativity, prefetching etc)

In contrast, our approach is to use a very simple, low cost, reliability technique: block disabling. Block disabling [20] is a well known technique that disables faulty parts of the cache at the granularity of the cache block. It known to be implementable [8] and requires very little area overhead (1 bit per cache block). We describe word disabling in detail in Chapter 3.

The main contribution our work is not the notion of block disabling but the analysis that explains why it can be an attractive option to consider for high fault rate scenarios. This analysis explains how to tune cache parameters such as block size and associativity in order to increase resilience to faults. By using a smaller block size(32 bytes instead of 64), more capacity can be achieved. Furthermore, the smaller block size helps reduce clustering of faults into heavily accessed sets. We show, using simulations, that these two advantages translate into increased performance. In addition to the analysis, we show that by leveraging performance mechanisms, such as victim caching and prefetching, we can remedy the performance losses caused by faults. Specifically, we show that a victim cache is beneficial as it relieves heavily accessed sets that have many faults. This is particularly useful for reducing performance variation -caused by the random placement of faults- among different processors. Prefetching was also found to be useful for increasing performance in the presence of faults.

Our best performing block disabling configuration is shown to perform significantly better than word disabling and bit-fix [32], two recently proposed, reliability techniques targeting high failure rate caches. Also, our minimum-cost block disabling configuration performs similarly to word disabling/bit-fix while being simpler to implement and requiring less area overhead.

1.3 Prior Work

In [16] we evaluated block disabling as a way to protect L1 caches when operating below Vccmin. Using probability analysis we showed that for a wide range of probability of failure, block disabling has more capacity than word disabling, a recently proposed technique for low voltage operation. Using experiments we showed that block disabling performs better than word disabling. Furthermore we showed that using a victim cache, performance variability caused by the random placement of faults was greatly reduced. This thesis builds upon the work in

[16] and extends the analytical and experimental analysis of block disabling for the whole cache hierarchy(L1 and L2 caches). Additionally, we evaluate caches equipped with prefetching and show that it is beneficial in reducing the performance overhead of faults.

During the course of this thesis we performed other work that is not directly related to this thesis. In [15] we explored the effects of faults in non-architectural arrays such as the branch predictor the return address stack the line predictor and the LRU bits of the caches. This work showed that performance can drop significantly(up to 53%) because of faults in the prediction arrays. Also, the return address stack was shown to be the most sensitive to faults while the LRU bits of the caches did not significantly affect performance when faulty. In [28],we proposed a reliability scheme based on address remapping that is able to recover the performance loss caused by faults in the line predictor. When up to 5% entries in the line predictor are faulty our mechanism recovers most performance loss and when no faults exist it does not degrade performance.

Chapter 2

Related Work

In this chapter we review various cache reliability techniques that are suitable for high fault rate scenarios. Most of these techniques treat low voltage failures and process variation induced faults as permanent faults. The common assumption, which we also share for this work, is that faults can be identified a priori using low voltage tests after manufacturing or during system boot. We focus our review on *word-disabling* and *bit-fix* as these are the techniques that we will compare our method against.

2.1 Word Disabling

This section reviews the word-disable scheme proposed in [32] to enable correct cache operation below Vcc-min.

The word-disable scheme tracks low-voltage faults at word granularity. It maintains a fault mask per block in the tag array. The fault mask contains as many bits as words in a block and each bit indicates whether its corresponding word contains a fault. The fault mask is initialized during the boot sequence of a processor using low voltage memory tests.

During high voltage operation the fault-mask is ignored and cache operates normally. When operating at low-voltage a pair of physical blocks in a set is merged into one logical block. This divides by two the cache capacity and associativity[1] .

The first physical block is responsible to provide only the first half of the logical block while the other half is provided by the second block of the pair. This means that up to n/2 faulty words can be tolerated for a subblock with n words. If a subblock has more than n/2 faulty words it turns the whole cache defective and not suitable for low-voltage operation. Chapter 4 analyzes how fault distribution affects the likelihood of a word-disable cache to be classified as faulty.

To read out in aligned form the valid half block contained in each physical block, the data in each block need to pass through a shift-multiplexer network controlled by each block's fault-mask. This alignment network increases the access latency of the cache in low-voltage mode and may even increase the cache latency during high-voltage operation.

For this work when using word-disabling the subblock size is 8 words and, therefore, no more than 4 faulty words can be tolerated in each subblock. The paper by [32] shows that for an 8 word subblock size the alignment network increases cache latency by 1 cycle.

Word-disabling is only applied to the data array of a cache. The tag array where the fault-mask is stored uses 10-transistor Schmitt trigger cells (10T) which are known [13] to be robust even at low-voltage. These transistors have roughly twice the area overhead of a regular 6-transistor (6T) cell.

2.2 Bit-Fix

This section reviews the bit-fix scheme proposed together with word disabling in [32].

[1]This scheme is only applicable to associative caches.

This scheme repairs faults at the granularity of bit-pairs. In low-voltage mode a subset of the blocks in each set are devoted to maintain a list of repair entries. The entries in the repair blocks are used to correct the faults in the remaining blocks of a set. Therefore, bit-fix, unlike word-disable, does not require additional space in the tag array to store the repair entries.

Each repair entry contains a pointer field that specifies a block position with a fault and a value field that contains the correct 2-bit value that replaces the faulty value. Additionally, each repair entry includes error correction codes to protect it from faults. When the number of faulty bit-pairs of any block is greater than the maximum allowed number of repair entries per block the whole cache becomes unsuitable for low-voltage operation. This can be made very rare, based on probability analysis in [32], by using one repair block for every three blocks. Therefore, the finer repair granularity of bit-fix decreases cache associativity and size by only 1/4 as compared to 1/2 of word-disable.

Bit-fix added complexity is that during low voltage operation any access that hits in the cache needs to access both the matching block as well as its repair block and merge them to produce the correct data. This block merging requires many shift-multiplexer stages that can increase the cache access latency by several cycles. The operation during high voltage remains unaffected except possibly longer latency due to the shift-multiplexer logic.

For both word-disable and bit-fix schemes a cache flush is needed when switching to low voltage mode to initialize the cache for low voltage operation.

2.3 Word-Disable vs Bit-Fix

The performance analysis in [32], for a specific processor configuration, revealed that word-disabling is more suitable, as compared to bit-fix, for a first-level cache. The fastest access latency of word-disable makes up for its lower capacity as compared to the slower but with larger capacity bit-fix scheme. For the second level cache the two schemes provide the same performance but the bit-fix consumes less of-chip bandwidth and thus may be preferable. Overall, the best configuration degrades average performance for high-voltage operation by 4% and low-voltage operation by 10%.

2.4 Other Techniques For High Fault Rates

Roberts et al. proposed a technique that can deal with cache memories with high cell defect probabilities[25]. Their technique is similar to word-disabling[32] in that it merges pairs of blocks together to produce fault-free blocks. Their mechanism however, only merges blocks if there is fault(while word-disabling always merges adjacent blocks in low voltage mode). Additionally, a selector table is used that allows the mechanism to pair together any combination of blocks(where word-disabling paired together adjacent blocks). These two improvements allow for significantly higher capacity but increase the complexity of the cache access mechanism.

Koh et al. proposed the Buddy cache mechanism[12] that also aims to protect caches at high fault rates. In the Buddy cache mechanism, cache blocks are divided into smaller segments(bytes, half-words, words etc.).The mechanism keeps a fault map and a Buddy map for each block in the cache. The fault map tracks the segments in the block that are faulty while the buddy map associates each block with another block in the cache. When reading a faulty block, its associated buddy block is read as well. The two blocks are merged together using information from their fault maps to give a working, fault-free block. Blocks that do not contain faults are associated with themselves. In order to keep the area overhead of the fault map and the Buddy map reasonable, block segmentation needs to be coarse and the number of blocks that any block can be associated with must be limited.

Another work that also deals with high defect rate caches was by Ansari et al[2]. They assumed the presence of spares for the protection of faulty cache lines. Their mechanism, called ZerehCache(ZC), allows to use spares optimally in order to increase yield. Spare lines are statically assigned to a group of logically adjacent cache blocks. The ZC mechanism uses a Benes network to rearrange the physical-logical mapping of caches lines so that spares are better utilized. Additionally spares are segmented by the mechanism into smaller sections(bytes, words etc.). This allows to use segments of the spare to fix different cache lines that can be faulty in the same cache line grouping.

Abella et al. [1] showed how to use subblock disabling without sacrificing performance predictability. They keep one bit per subblock(disable bit) that signals whether the subblock contains fault or not. When a subblock is accessed, if its corresponding disable bit is set the subblock is discarded and the access results in a miss. Subblock disabling can provide high capacity but suffers from the problem of unpredictable performance. Depending on the locations of the faults in the cache, performance may vary significantly. To remedy this, the authors proposed to use an XOR-based address remapping scheme. By using address remapping, heavily accessed cache sets can avoid being mapped in locations with many faults. The authors also show how remapping can be implemented without increasing the cache access time.

Sasan et al. [27] used Monte Carlo simulation to estimate the cache failure rate for different supply voltage values. They also introduced the RDC-cache, a reliability mechanism for enabling low voltage operation. The RDC-cache keeps a defect map at the cache word granularity. In contrast to word-disabling however, this defect map is stored inside the data array of the cache so as to not require additional storage. The mechanism allows for non-adjacent blocks to be merged together to give a fault free block. Furthermore, fault free blocks do not have to be merged. This approach achieves higher capacity than word disabling at the expense of increased complexity.

An alternative to using cache deconfiguration mechanisms is to design memories using transistors that are built specifically for operation below Vccmin. An example of such a transistor is described in the work by Kulkarni et al. [14]. This approach, however, is only applicable to small SRAM memories since the area overhead for using this type of transistors is very high(as much as 100% area overhead).

Many of the above techniques have been shown to outperform word disabling and bit-fix. However, the additional performance comes at the expense of higher area overhead and increased complexity. We chose to compare our technique against word disabling and bit-fix as we believe they are more suited for implementation on an actual processor.

2.5 Other Relevant Work

In [29], Sohi et al. explored how tolerating cache faults(using block disabling) affects performance. In the presence of few defects they concluded that performance is not significantly affected and so block disabling can be used to enhance processor yield. Pour et al. [21] performed analytical evaluation of cache block disabling and concluded that few disabled blocks do not affect miss ratio considerably unless a whole set happens to be disabled. Lee et al. [17] also concluded that amongst cache lines, sets and ways, deleting cache sets has the most severe performance impact as a whole portion of the address space cannot be mapped to the cache.

Chapter 3

Proposed Technique

In this chapter we describe block disabling, which is the basis of our proposed configuration for protecting caches against faults. We provide a high level comparison between block disabling and the two mechanisms, word disabling and bit-fix, that we will use to compare our approach with. Also, we discuss possible limitations of the block disabling mechanism and our approach for dealing with these limitations.

3.1 Overview of Block Disabling

Block disabling is a well-known technique that was proposed in [20] for increasing yield. One extra bit per cache block is required by the mechanism. The faulty blocks in a cache are identified using standard memory tests(e.g. march tests) or low voltage tests if operating below Vccmin. This can be done during manufacturing or at-field using built in self test(BIST) hardware. When a block is identified as faulty, its matching disable bit is enabled and that block never gets allocated. This allows correct operation in the presence of permanent faults at the expense of reduced cache capacity.

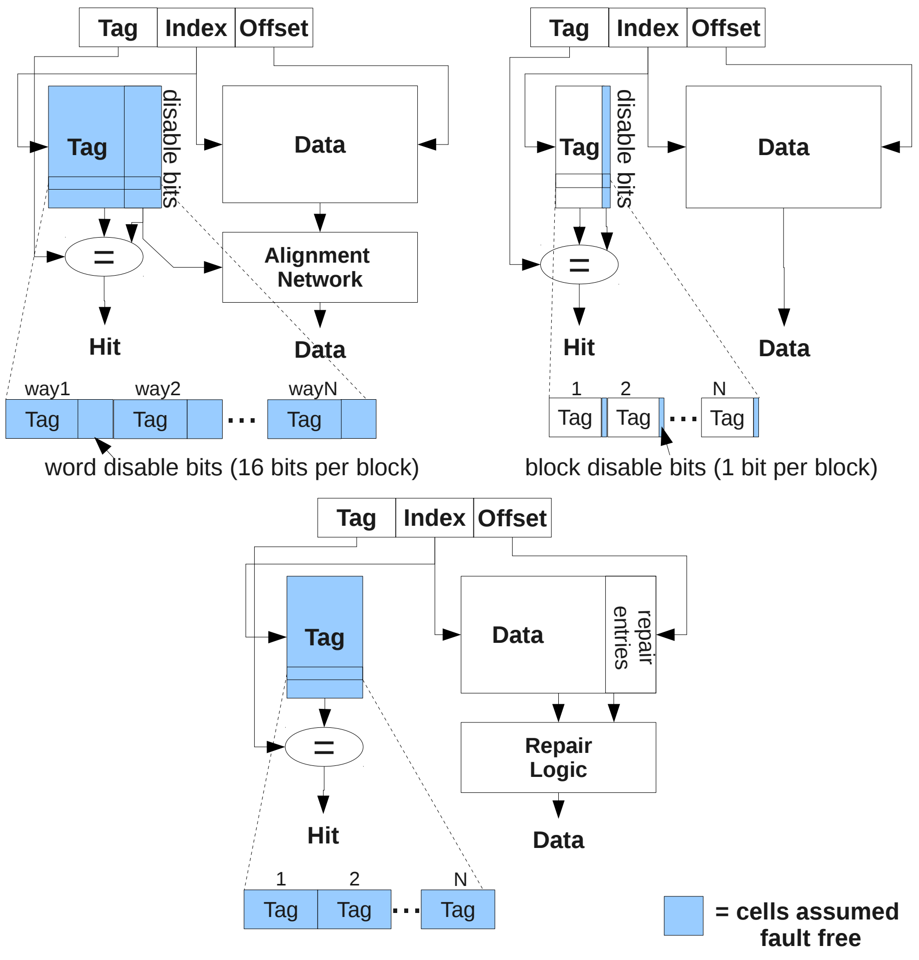


Figure 2: *Left*: Word disabling mechanism. *Right*: Block disabling mechanism. *Bottom*: Bit-fix mechanism

3.2 Block Disabling, Word Disabling and Bit-Fix

Figure 2 shows a high-level comparison of word disabling, block disabling and bit-fix. The figure indicates that block disabling is a simpler mechanism. It requires one extra bit per block whereas word disabling requires 1 extra bit per word. Additionally, word disabling requires an alignment network to shift out faulty words. This further increases area cost and implementation complexity while adding extra latency to the cache access time. The parts shaded in the figure(tag bits and fault bits for word-disable, disable bits for block disabling and tag bits for bit-fix) are assumed to be fault free by the mechanisms. To ensure that these bits to fault free, special transistors

	Block Disable	Word Disable	Bit-fix
Advantages	Simple to implement Small area overhead	Deterministic capacity(50%) Deterministic associativity(50%)	Deterministic capacity(75%) Deterministic associativity(75%)
Limitations	Possibly low capacity Non deterministic associativity	High area cost Complex Small increase in cache latency	Complex Big increase in cache latency

Table 1: Comparison of block disabling, word disabling and bit-fix

must be used that are resilient to process variation [14]. This further increases the area cost for word disabling while it is less of a problem for block disabling since it requires fewer protected bits. Bit-fix requires the cache data to pass through multiple stages of repair logic in order to repair faulty bit pairs. Also, as in word disabling it requires the tag bits to be fault-free. The advantage of bit-fix is that it does not require extra storage as it stores its repair entries inside the cache(taking up 25% of cache capacity). The above complexities of word disabling and bit-fix make block disabling the simpler and less expensive mechanism to use.

However, block disabling has some disadvantages. It disables the cache at a coarse granularity(cache block). This may have adverse effects on cache capacity if the number of faults in the cache is high. Furthermore, block disabling behaviour is non deterministic. Depending on the location of faults in the cache, different cache blocks may be disabled. As a results some sets may have significantly less available blocks than other sets. In worst case scenarios a whole set may be completely disabled requiring the complete bypass of the cache when that block is accessed. This can lead to significant performance variations from processor to processor. In contrast, this is not a problem for word disabling and bit-fix since capacity and associativity is guaranteed by the mechanisms to be constant(50% and 75% respectively).

Table 1 shows an overview of the benefits and limitations of each mechanism. From this qualitative comparison it is not clear which mechanism is better. In the following section we describe how we propose to compensate for the limitations of the block disabling mechanism making it a more attractive solution than the other methods.

3.3 Improving Block Disabling

Block disabling suffers from the following limitations: capacity is dependent on the number of faults and associativity(and as a result performance) is non deterministic as it depends on the location of faults. In Chapter 4 we will show, using probability analysis, that by selecting the correct cache block size, block disabling capacity can be similar or higher to word disabling and bit-fix. Also, we will show that when very large numbers of faults are present, word disabling cannot operate whereas block disabling can(albeit with reduced capacity). Furthermore, we show that by increasing the associativity of the L2 cache we can greatly reduce the chance of having a set that is completely faulty.

For the L1 caches we cannot increase the associativity since it can affect the cache access time. To remedy possible performance non determinism, for 64B caches, we propose to use a victim cache [11]. As we will see in Chapter 5, using a victim cache relieves accessed sets that happen to have many disabled sets and makes performance more deterministic. For 32B caches we will show both analytically and experimentally that performance is more deterministic since faults are less likely to cluster on few sets. Furthermore, we will show that by using prefetching much of the performance loss occurred due to faults can be recovered.

Chapter 4

Analytical Evaluation and Comparison

In many cases, reliability problems for regular structures, such as caches, can be abstracted and studied analytically. This approach allows to quickly explore parameters without the need for numerous time consuming simulations. In this chapter, we use probability analysis to evaluate and compare the capacity and yield of block disabling, word disabling and bit-fix for varying numbers of cache faults.

For the analysis in the next sections we assume that faults occur with uniform random distribution at the granularity of a cell. Random process variation faults vary at such fine granularity [6]. The work that we compare against [32] also makes the same assumptions. In our analysis we use the term p_{fail} which is the probability of cell failure. For example, if p_{fail} is 0.01, a cell has 10% chance to be faulty or, put otherwise, 1 out of 10 cells will be faulty on average.

4.1 Capacity Analysis

The block disabling mechanism guarantees that cache capacity will be 50% in the presence of faults. Similarly, bit-fix enables 75% of the capacity when operating under faults. For block disabling however, capacity is dependent on the number of faults in the cache. The number of

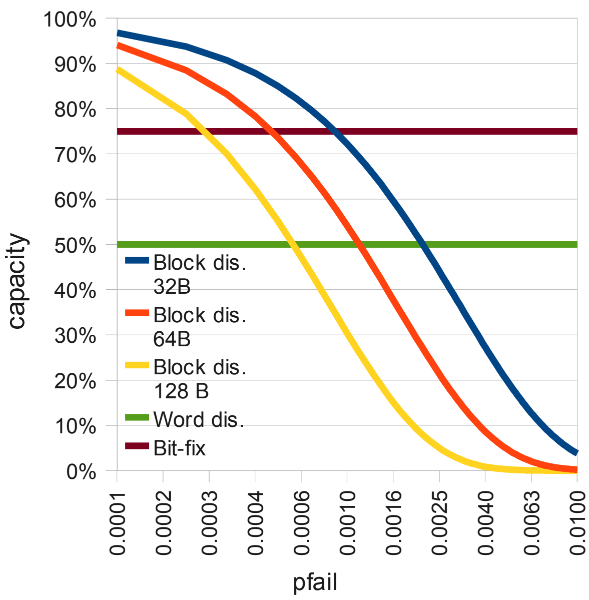


Figure 3: Capacity as a function of p_{fail}

faults in the cache is in turn dependent on the probability of cell failure - p_{fail}. Using the following expression we can determine the capacity for a given p_{fail}.

$$capacity = (1 - p_{fail})^{k} \tag{1}$$

In the above expression *k* is the number of bits that, when faulty, the block in the cache is disabled. For our work *k* = number of bits in a block + tag bits + ECC bits + dirty bit + valid bit. This allows block disabling to protect against all faults that would compromise correctness. ECC bits are also included so that soft error protection is not compromised by low voltage operation or by process variation faults. Other bits, such as LRU bits, are ignored since they do not affect correctness. Using equation 1 we plot Figure 3 that shows capacity for block disabling, word disabling and bit-fix for varying p_{fail}. For block disabling we plot three lines for varying block sizes: 32B, 64B and 128B.

Since *k* in equation 1 is dependent on the number of bits per block, reducing the block size results in increased capacity. This happens because when the block size is small, a single fault disables less cache area compared to when the block size is bigger. Figure 3 shows that the effect of block size in capacity is significant. When p$_{fail}$ is 0.001 for example, a 32B block cache will have 72% capacity while a 128B cache will only have 30%. These results suggest that for a block disabling scheme, selecting a smaller block size is crucial.

When we compare block disabling to word disabling we can see that depending on p$_{fail}$, either mechanism can have higher capacity. For p$_{fail}$ up to 0.002, block disabling(with 32B block) has higher capacity. For p$_{fail}$ values higher than 0.002, word disabling offers more capacity. For p$_{fail}$ up to 0.0009, block disabling offers more capacity than bit-fix as well. From this analysis, it is not clear which mechanism performs better since capacity depends on p$_{fail}$.

4.2 Yield Analysis

As described in Section 2.1, the word disable mechanism can only tolerate up to n/2 faulty words in each subblock of n words. This means that when the number of faults in a subblock is greater than n/2 the mechanism cannot operate and the chip has to be tossed(assuming no other deconfiguration mechanisms are in place). Bit-fix cannot operate when more than 10 bit-pairs are faulty in a block. In contrast, block disabling can operate despite high numbers of clustered faults. This is a potential disadvantage for the word disable and bit-fix mechanisms. To determine the probability of the word disabling mechanism to be inoperable we used the following expression.

$$p_{word\_dis\_fail} = 1 - (1 - p_{subblock\_fail})^{blocks \times 2} \tag{2}$$

Where *blocks* is the number of blocks in the cache and $p_{subblock\_fail}$ is the probability that a subblock will have more than n/2 faulty words(n is the number of words in a subblock). $p_{subblock\_fail}$ is given by the following equation.

$$\sum_{i=n/2+1}^{n} \binom{n}{i} (p_{wf})^i (1-p_{wf})^{n-i} \tag{3}$$

Where $p_{wf} = 1-(1-p_{fail})^{32}$ is the probability that a word will be faulty (assuming 32 bit words). Note that the above equations do not take the tag bits into account since for the word-disabling scheme, the tag bits are assumed to be built using reliable 10T cells and are therefore always fault free.

For bit-fix, we determine the probability that mechanism is inoperable using the following expression.

$$p_{bitfix\_fail} = 1-(1-p_{block\_fail})^{blocks} \tag{4}$$

A block fails when it contains more than 10 faulty bit pairs. The probability of having more than 10 faulty bit pairs in a block is given by the following.

$$\sum_{i=11}^{a} \binom{a}{i} (p_{bpf})^i (1-p_{bpf})^{a-i} \tag{5}$$

Here, *a* is the number of bit pairs in a block and $p_{bpf} = 1-(1-p_{fail})^2$ is the probability that a bit pair will be faulty.

Using equation 2 we plot Figure 4 which shows the probability of cache failure for 32Kb,64B block L1 cache equipped with the word disable mechanism. As the p$_{fail}$ increases, the probability of whole cache failure increases dramatically. After p$_{fail}$ 0.0015, yield starts to be significantly affected(1/100 caches will be faulty). The 64B block block disabling configuration has a capacity advantage over word disabling for p$_{fail}$ up to 0.0012. This means that word disabling has the capacity advantage for only a small fraction of p$_{fail}$ before yield starts to be affected(illustrated with gray in Figure 4). For p$_{fail}$=0.0022, word disabling offers more capacity than both block

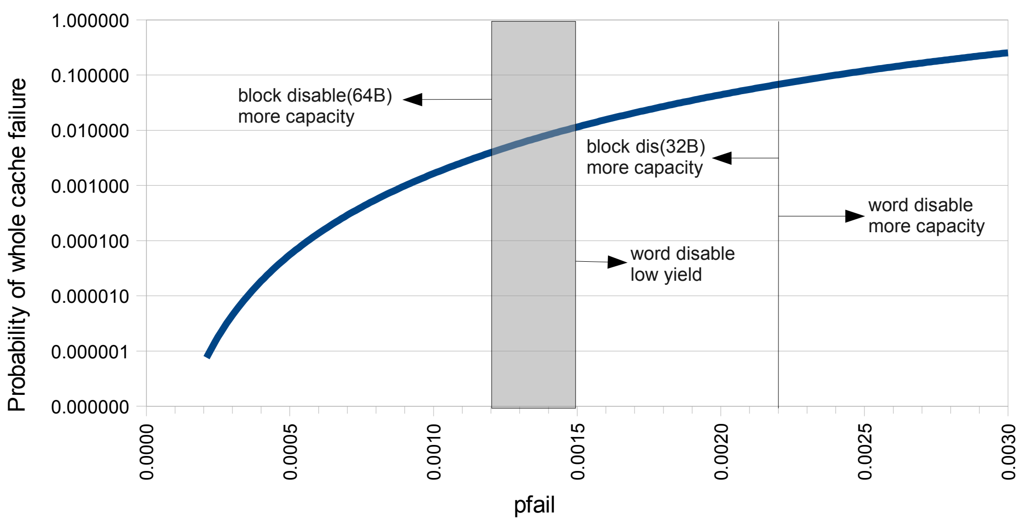


Figure 4: Probability of cache failure for word disabling

disabling configurations(32/64B block). However, at this p_{fail} the probability of whole cache failure is 6.7%. This means that for the probabilities of cell failure that word disabling has a capacity advantage, yield is significantly affected due to the limitations of the mechanism. This, however, is not a problem for the bit-fix mechanism. As stated in [32], for p_{fail}=0.001, only 1 out 1 billion caches are expected to fail.

In this section we have established that for a wide range of p_{fail} block disabling offers more capacity than word disabling or bit-fix. Furthermore, for high probabilities of failure, block disabling continues to operate whereas word disabling may render the cache inoperable. However, when p_{fail} is high, block disabling may suffer severe performance degradation if many(or all) blocks are disabled in a frequently accessed set. In the following section we will determine analytically the likelihood of such a scenario.

4.3 Probability of Set Failure

We define $p_{setfail}$ as the probability that of cache set having all of its blocks disabled due to faults. When a set failure happens, performance is likely to degrade, especially if the failing

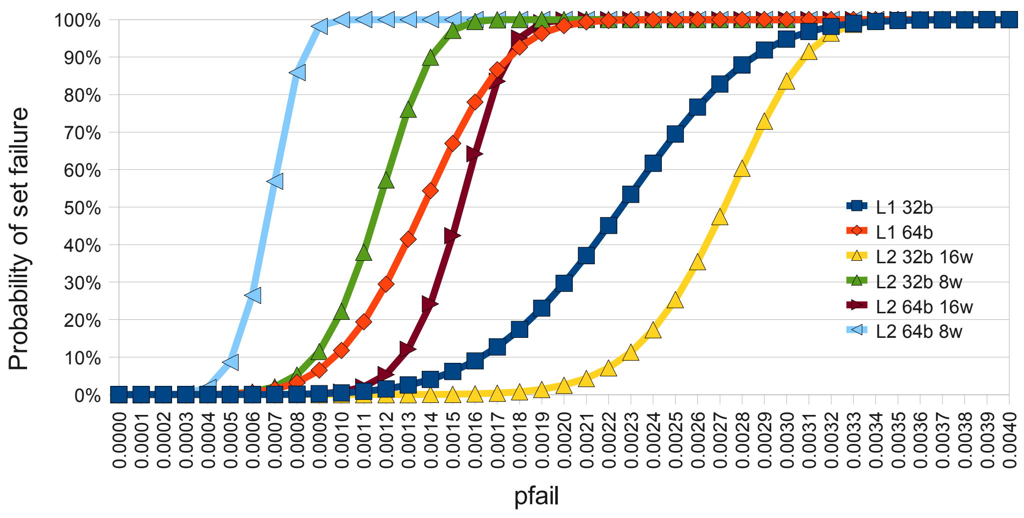


Figure 5: Probability of set failure as a function of p$_{fail}$

set happens to be frequently accessed. This is a potential shortcoming of the block disabling mechanism. Using the following expression we can determine how likely set-failures are.

$$p_{setfail} = 1 - (1 - p_{sf})^{sets} \tag{6}$$

Where *sets* is the number of sets in the cache, and $p_{sf} = {p_{bf}}^{ways}$ is the probability of a single set having all blocks disabled. p_{bf} is the probability that a block is faulty and can be easily obtained using equation 1. Using equation 6 we produced Figure 5 which shows the probability of set failure for varying cache configurations. A description of each configuration can be found in Table 2. As the figure illustrates, decreasing the block size and increasing the associativity results in smaller probability of set failure. When p$_{fail}$ is 0.001 the 1 out of 250 L1 caches with 32B blocks will experience whole set failure. For L2 caches however, using a 32B block alone is not sufficient. The 8-way L2 configuration will experience 1 set failure in every 5 caches which may not be acceptable in terms of performance variation. To remedy this we can use a 16-way L2 cache. With the 16-way, 32B block configuration, L2 caches virtually never experience set failure.

Configuration	Block Size	Associativity
L1 32b	32 Byte	8-way
L1 64b	64 Byte	8-way
L2 32b 16w	32 Byte	16-way
L2 32b 8w	32 Byte	8-way
L2 64b 16w	64 Byte	16-way
L2 64b 8w	64 Byte	8-way

Table 2: Description of cache configurations

	Number of failed sets			
Configuration	0	1	2	3
L1 32b	0.9955	0.0045	1.01E-005	0.0000
L1 64b	0.8824	0.1105	0.0068	0.0002
L2 32b 16w	1.0000	4.03E-006	0.0000	0.0000
L2 32b 8w	0.7778	0.1953	0.0245	0.0020
L2 64b 16w	0.9930	0.0069	2.44E-005	5.72E-008
L2 64b 8w	0.0005	0.0041	0.0154	0.0387

Table 3: Probability of multiple set failures for p_{fail}=0.001

It is also interesting to know if it is likely to have multiple set failures in the same cache. To determine this we use the following expression:

$$\binom{sets}{x}(p_{setfail})^{x}(1-p_{setfail})^{sets-x} \tag{7}$$

Where *x* is the number of failed sets. Using this expression we calculated the probability of having 0,1,2 and 3 set failures in the same cache for all configurations in Table 2 for p_{fail}=0.001. The results are shown in Table 3. The results show that for L1 caches and L2 16-way caches the probability of multiple set failures is small. For 8-way L2 caches, however, multiple set failures are likely. For this reason we believe it is highly preferable to use 16-way L2 caches in combination with block disabling as multiple set failures may significantly affect performance.

The above results indicate that unless the correct cache configuration is chosen, set failure is likely to occur in a block-disabling cache. However, a set failure on a set that is rarely accessed during program execution will not affect performance. To assess the likelihood of set failure occurring on a heavily accessed set we use equation 6. We change *sets* in the equation to be the number of heavily accessed sets in the cache. In order to determine the number of heavily

	32 B cache			64 B cache		
	IL1	DL1	L2	IL1	DL1	L2
ammp	22	40	3	33	30	3
applu	30	8	0	54	64	0
apsi	42	40	0	40	47	0
art	17	5	0	23	64	0
bzip	14	12	0	15	12	0
crafty	27	24	1	48	32	1
eon	34	39	0	44	35	0
equake	42	21	0	22	17	0
facerec	19	17	0	16	37	0
fma3d	26	33	0	33	33	0
galgel	12	2	1	7	64	1
gap	40	26	0	35	21	0
gcc	17	13	0	16	64	0
gzip	34	15	0	21	13	0
lucas	20	4	4	11	4	4
mcf	16	2	3	10	64	3
mesa	40	33	1	42	26	1
mgrid	26	7	0	55	64	0
parser	26	31	0	35	31	0
perlbmk	31	17	1	34	16	1
sixtrack	29	28	0	18	45	0
swim	29	2	0	41	64	0
twolf	30	19	0	29	46	0
vortex	24	31	3	39	26	3
vpr	26	14	3	19	29	3
wupwise	23	31	0	15	19	0
average	**26**	**19**	**1**	**29**	**37**	**1**

Table 4: Number of heavily accessed sets per cache

accessed sets we performed simulations for 2 cache configurations: 1) I-cache/D-cache 8-way, 64B block; L2 8-way, 64B block. 2) I-cache/D-cache 8-way, 32B block ; L2 16-way, 32B block. Our simulation environment is described in detail in chapter 5. We consider a set as heavily accessed(or hot) if the number of accesses for that set is greater than 1% of the total number of accesses for the cache containing the set. Table 4 shows the number of heavily accessed set for the aforementioned two cache configurations and for each of the 26 SPEC2000 benchmarks that we simulated.

Using the averages in Table 4 and equation 6, we plot Figure 6 which shows the probability of having a set failure on a hot set. The probability of set failure happening on hot set is of course smaller than the probability of set failure happening anywhere in the cache. However, the figure

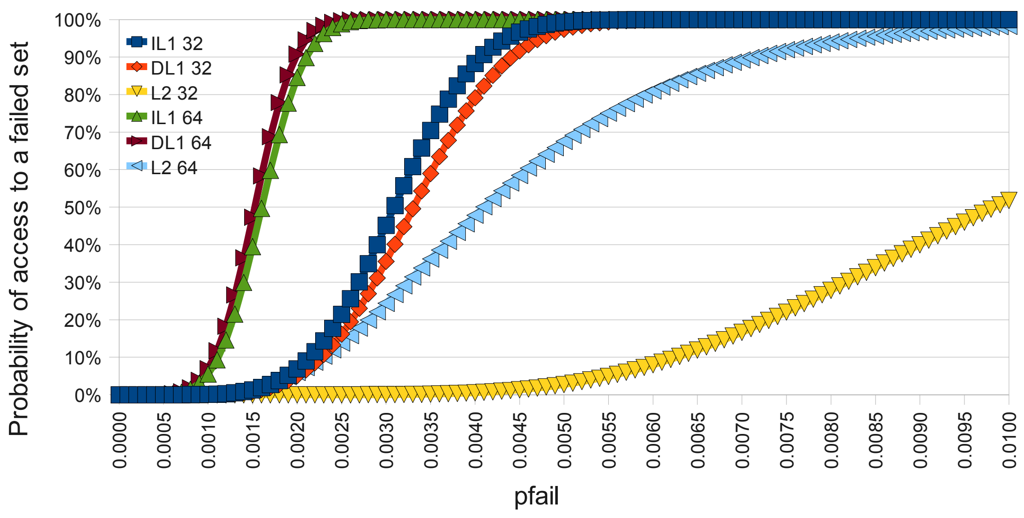


Figure 6: Probability of set failure occurring on a frequently accessed set

shows that for the 64B block L1 caches, hot set failure is likely. Specifically, for the 64B-IL1 cache configuration, 3 out of 100 caches will experience hot set failure. For the 64B-DL1 configuration, hot set failure happens at a rate of 4/100 caches. In contrast, the 32B caches experience hot set failure at a rate of 3/1000 caches. The L2-64B cache is also much less likely to have hot set failure (1/1000 caches) and in L2-32B caches, hot set failure virtually never happens. The above results indicate that clustering of faults in hot sets is likely for the L1 caches if the block size is 64 bytes. This can be remedied by using a 32B block.

The above analysis considers the average number of hot sets across all 26 SPEC benchmarks. This approximates the probability of having set failure on a hot set for a typical program. However, most computers run multiple programs(e.g. operating system processes, user applications). Different programs are likely to have different hot sets. As result, when running multiple programs the number of hot sets increases and the probability of hot set failure increases as well. Given a cache that has one set failure, the probability of one program accessing that set is $p_{sfa} = \frac{program\ hot\ entries}{sets\ in\ cache}$. When a number of programs m is using the cache, the probability

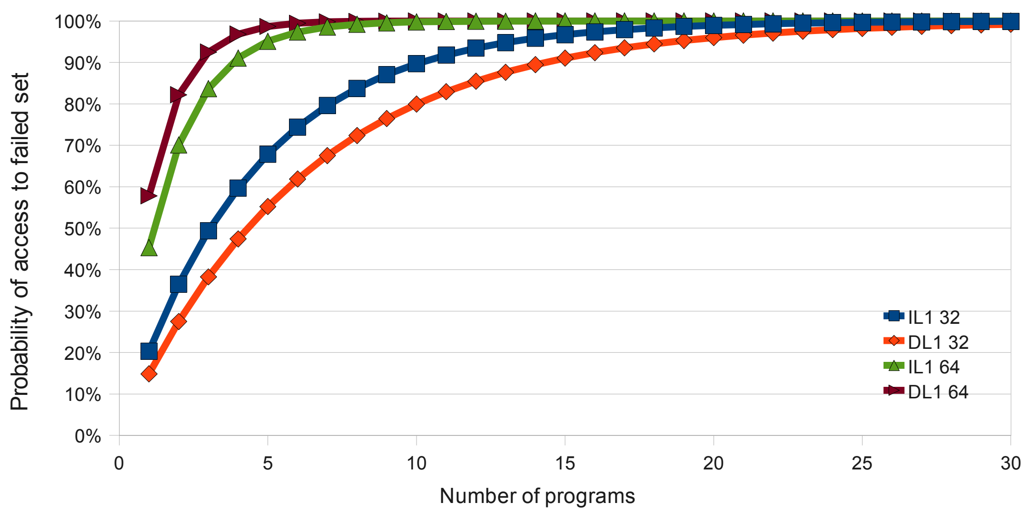


Figure 7: Probability of accessing a failed L1 set for varying number of programs

that one of the programs will access the failed set is[1] $p_{sfa-m} = 1 - (1 - p_{sfa})^m$. Using this expression and the averages in Table 4 plot Figures 7 and 8. Figure 7 shows, for a varying number of programs, the probability of accessing a failed L1 set. As the figure shows, even for a small number of programs, accessing a failed set is very likely. This means that for L1 caches if a set failure occurs, it is very likely that it will affect a hot entry for some program. Figure 8 shows the results for L2 caches. The figure shows that even when a very large number of programs is considered, the probability of set failure on a hot set is small(2.4% for 32B block caches and 4.7% for 64B block caches at 100 programs). This happens because the ratio of hot sets and total cache sets(p_{sfa}) is very small for L2 caches.

4.4 Extending the Methodology for Other Mechanisms

In this section we demonstrate how the analytical methodology described in this chapter can be extended to other mechanisms. Specifically, we will perform analytical estimation of cache capacity for a reliability mechanism similar to the one proposed by Roberts et al. [25]. This

[1]This analysis assumes that the positions of hot sets for each program are random.

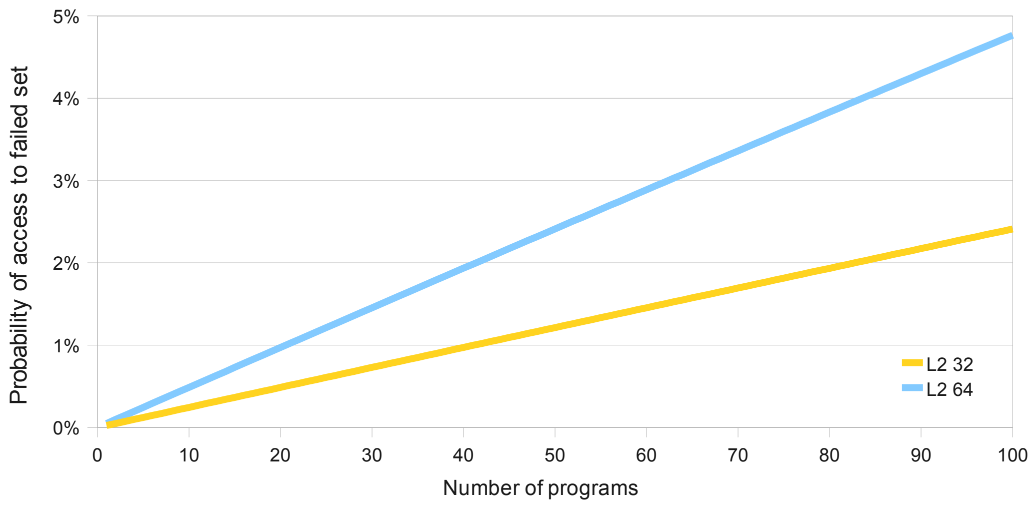


Figure 8: Probability of accessing a failed L2 set for varying number of programs

mechanism, which we will call incremental word disabling, allows pairs of blocks that are fault free to operate at full capacity even at low voltage operation. Additionally, pairs of blocks that contain a half-block with more than 4 faulty words are disabled so that the whole chip does not have to be discarded. Block pairs that contain faults, but do not have to be disabled, operate at half capacity as in the word-disabling scheme. To estimate the capacity of this scheme we use the following:

$$p_{bpff} + (1 - p_{bpff} - p_{bpd})/2 \tag{8}$$

where p_{bpff} represents the probability that a block-pair is fault-free, and p_{bpd} the probability that a block-pair is disabled. Therefore, expression $1 - p_{bpff} - p_{bpd}$ corresponds to the fraction of block pairs that operate at half capacity. To calculate p_{bpff} we use the following expression: $p_{bpff} = (1 - p_{fail})^{k \times 2}$ where k is the number of data bits in a block. p_{bpd} is calculated from $1 - (1 - p_{hbf})^4$ where p_{hbf} represents the probability that a half-block will contain more that 4 faulty blocks and is obtained using Eq. 3.

Fig. 9 shows that the incremental word-disabling mechanism for a 32KB 64B/block cache performs well. At low probabilities of failure, the number of fault free block-pairs is high with

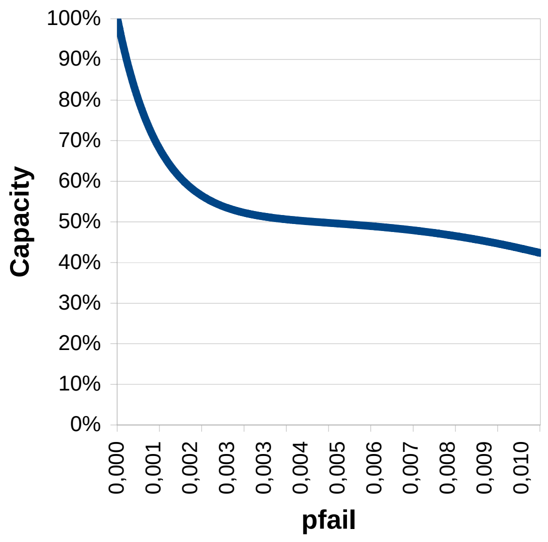


Figure 9: Capacity as a function of p_{fail} for the incremental word-disabling scheme

capacity over 50%. As the number of faulty cells increases, more block-pairs will contain faults and capacity begins to saturate at 50%. When we move to higher probabilities of failure, more block-pairs are disabled which decreases capacity to a value below 50%. The analysis shows that the incremental word-disabling scheme degrades more gracefully than word-disabling while completely avoiding the whole cache failure scenario.

We do note however, that this scheme may not be easy to implement. A block pair can be in three states: fault-free, disabled, half capacity. A different access path is required for the fault-free and half capacity blocks since the later requires the block to pass through the word-disabling shifting network. This can increase the cache access time non-determinism and may complicate implementation.

4.5 Overview

In this chapter we described an analytical methodology for estimating capacity, yield and clustering of faults. This analysis indicates that with a 32B block configuration, block disabling has a capacity advantage over word disabling and bit-fix for a wide range of p$_{fail}$. Also, our yield analysis shows that for very high failure rates($p_{fail} > 0.001$), word disabling is likely to be inoperable since it cannot handle more than a fixed number of faulty words in each subblock. Furthermore, we have seen that a 64B block cache is more likely to suffer hot set failures (clustering of faults in a frequently accessed set) whereas a 32B block cache is more resistant. We have also seen how the methodology described in this chapter can be easily extended to study other reliability mechanisms.

The analysis in general leads us to believe that when using block disabling, cache configuration is an important factor that cannot be overlooked. Also, the capacity benefits from using block disabling indicate that it may be a preferable solution over word disabling and bit-fix.

Chapter 5

Experimental Evaluation and Comparison

In this chapter we will evaluate block disabling, word disabling and bit-fix experimentally. Experimental evaluation is necessary because analytical evaluation does not take into account all the parameters that may affect performance in a real processor. For example, more capacity may not necessarily translate to improved performance if the capacity requirements of a program are low. Furthermore, probability analysis does not take into account performance mechanisms that may be present in a processor -such as a victim cache or a prefetcher- that may lessen or worsen the effects of faults in the cache.

5.1 Experimental Framework

For our experiments we used the validated, cycle accurate simulator *sim-alpha* [9] that models a high-performance out of order superscalar processor. The simulator is extended to support cache block disabling. Details of all modifications performed on the simulator can be found in Appendix B. Table 5 contains the processor parameters that are constant for all runs while Table 6 shows configuration specific parameters. For configurations that use a victim cache, we used a two 16

Parameter description	Setting
Pipeline depth	15 stages
Line Predictor	6.5 KB
RAS	16 entries
Branch Predictor	8 KB gshare (15 bits history)
Fetch/Decode/Issue/Commit	up to 4/4/6/4 instr. per cycle
Issue Queue	40 INT entries, 20 FP entries
Functional Units	4 INT ALUs, 4 INT mult/div, 1 FP ALUs, 1 FP mult/div
Reorder buffer	128 entries

Table 5: Processor parameters that are constant for all configurations

Configuration	L1(I+D) size, associativity, block, latency	L2 size, associativity, block, latency	Victim$	Prefetching	Fig
High Voltage - Frequency:3GHz, DRAM latency:255 cycles					
64B	32KB, 8-way, 64B,3	2MB, 16-way, 64B, 20	no	no	10
64B-Victim Cache	32KB, 8-way, 64B,3	2MB, 16-way, 64B, 20	yes	no	10
64B-Prefetch	32KB, 8-way, 64B,3	2MB, 16-way, 64B, 20	no	yes	10,12
64B-Both	32KB, 8-way, 64B,3	2MB, 16-way, 64B, 20	yes	yes	10
32B	32KB, 8-way, 32B,3	2MB, 16-way, 32B, 20	no	no	11
32B-Victim Cache	32KB, 8-way, 32B,3	2MB, 16-way, 32B, 20	yes	no	11
32B-Prefetch	32KB, 8-way, 32B,3	2MB, 16-way, 32B, 20	no	yes	11,12
32B-Both	32KB, 8-way, 32B,3	2MB, 16-way, 32B, 20	yes	yes	11
64B word-disable/bit-fix	32KB, 8-way, 32B,4	2MB, 16-way, 32B, 23	no	yes	13
Low Voltage - Frequency:600MHz, DRAM latency:51 cycles					
64B-No Faults	32KB, 8-way, 64B,3	2MB, 16-way, 64B, 20	no	no	14
64B-No Faults-Prefetch	32KB, 8-way, 64B,3	2MB, 16-way, 64B, 20	no	yes	14
64B	32KB, 8-way, 64B,3	2MB, 16-way, 64B, 20	no	no	15,18
64B-Prefetch	32KB, 8-way, 64B,3	2MB, 16-way, 64B, 20	no	yes	16,18
64B-Prefetch/V$	32KB, 8-way, 64B,3	2MB, 16-way, 64B, 20	ye	yes	18,20
32B	32KB, 8-way, 32B,3	2MB, 16-way, 32B, 20	no	no	15,19
32B-Prefetch	32KB, 8-way, 32B,3	2MB, 16-way, 32B, 20	no	yes	17,19,21
32B-Prefetch/V$	32KB, 8-way, 32B,3	2MB, 16-way, 32B, 20	yes	yes	19
64B word-disable/bit-fix	16KB, 4-way, 64B,4	1.5MB,12-way,64B, 23	yes	yes	20
32B word-disable/bit-fix	16KB, 4-way, 32B,4	1.5MB,12-way,32B, 23	no	yes	21

Table 6: Configuration dependent parameters

entry victim caches, one for each L1 cache(I+D). [1] For configurations that use prefetching, we used a sequential tagged prefetcher[23] for the L1 caches and a correlating prefetcher [22] in the L2. More details about the prefetching mechanisms can be found in Appendix A. The word-disable/bit-fix configurations employ word-disabling for the L1 caches and bit-fix for the L2. For convenience, Table 6 shows for each configuration, the figure that uses it.

[1]For simplicity we assume that the victim caches are fault free. Future work will investigate the performance effects of disabling parts of the victim cache. Preliminary analysis leads us to believe that disabling few entries of the victim cache will not significantly affect performance.

The performance of block-disabling is evaluated with many simulation runs since faults can occur randomly at any cell. In particular, block-disabling configurations are evaluated with 30 random fault maps for each cache(IL1, DL1 and L2). Each fault map contains an entry for each set in the cache that it corresponds to. For example, for an L2 cache with 4096 sets, each L2 fault map will contain 4096 entries. Each entry holds the number of faulty blocks for its associated set. The number of faulty blocks per set is determined randomly using as input the cell probability of failure -p$_{fail}$- and the number of bits associated with each block. For our experiments, the cell probability of failure is assumed to be 0.001, the same as in [32]. According to the projections in [19], cell failure rates greater than 0.001 are not likely even for 12nm technologies. Each block is associated with its data bits($blocksize \times 8$), the tag bits for that block, ECC bits for that block (1 bit per byte), a valid bit and a dirty bit. Whenever one of these bits is faulty, the associated block is considered faulty and its disable bit is set. This grouping of bits allows us to protect from faults in both the data and tag array of the cache. Also, by including the ECC bits, we make sure that soft error protection is not compromised by low voltage or process variation faults. In the next sections we report, for each block-disabling configuration and benchmark, the average and minimum values for 30 runs using different fault maps.

To simulate the operation of word disabling, we reduced the cache capacity and associativity by half and increased the cache access time by one cycle. Similarly for bit-fix we reduced cache capacity and associativity by 25% and increased the cache access time by 3 cycles. Since performance does not depend on the location of faults for these mechanisms, we did not use fault maps and instead performed one experiment per configuration required. In the next sections, word disabling and bit-fix are used together. Specifically, word disabling is used for the L1 caches and bit-fix is used for L2. This configuration was proven to be the best performing in [32].

Mechanism	Area required(bytes)	Area assumed fault free	Other overheads
Block disable 64B	4224	4224	N/A
Block disable 32B	125248	8448	N/A
Word disable/bit-fix	2048	118848	Shifting networks

Table 7: Comparison of block disabling, word disabling and bit-fix in terms of area overhead

Table 7 shows the area overheads for block disabling - using 32B and 64B block- and the word-disable/bit-fix configuration. The area overhead for the 64B block disabling configuration consists of one extra bit per block for all caches. When using 32B block the tag array doubles. This is reflected in the area cost of the 32B configuration. The area cost for the word-disable/bit-fix configuration is one bit per data word for the L1 caches. Since bit-fix stores the correction bits inside the cache there is no additional area overhead. However, word disabling and bit-fix require the tag array to be fault free. As a result the tag array bits must be protected using either ECC or reliable transistors that take up more area. Additionally, word disabling and bit-fix require a shifting network to remove/repair faults from the cache. Overall, the least expensive mechanism in terms of area overhead is block disabling using 64B block while the most expensive is block disabling with 32B block. It is important to note, however, that the 32B block disabling configuration is much simpler to implement than word-disable/bit-fix.

For all experiments that we present in the next sections, we run all 26 SPEC CPU 2000 benchmarks for 100M committed instructions using reference inputs. The simulation regions were selected using an in-house SimPoint-like[10] tool.

5.2 Fault Free Operation at High Voltage

In this section we compare various cache configuration schemes when operating at high voltage and no faults are present. Our goal is to select the optimal baseline configuration. Additionally, we compare our optimal configuration to a cache equipped with word disabling and bit-fix.

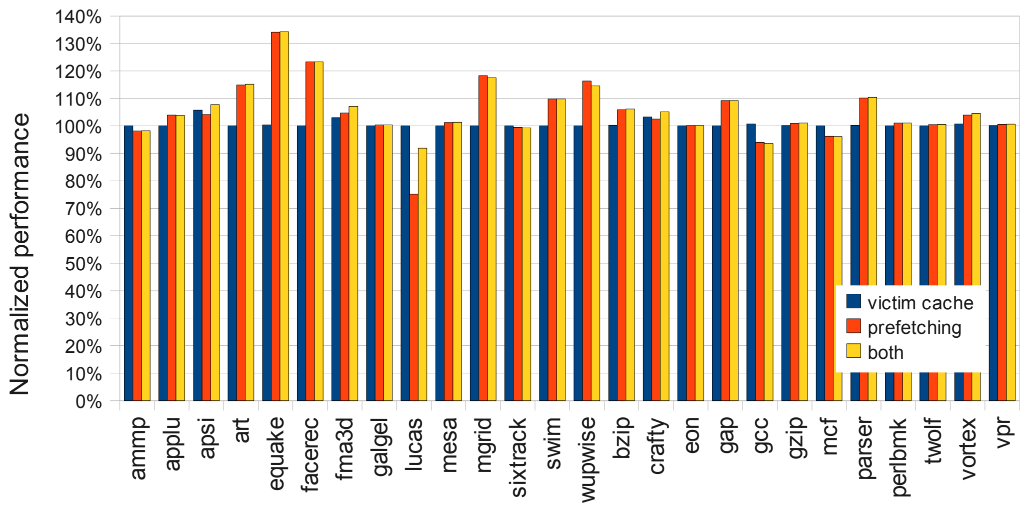


Figure 10: Performance improvements from using a victim cache, prefetching and the combination of both for a 64B block cache. Results are normalized to a 64B block cache configuration with prefetching and victim caching disabled.

Since the word disabling and bit-fix mechanisms increase cache access time(even when operating without faults), we would like to assess if the increased access time has significant impact on performance.

For the selection of the optimal baseline cache configuration we consider three cache parameters: block size(32B or 64B), victim cache and prefetching. Figure 10 shows the performance benefit from using victim cache, prefetching and the combination of the two for 64B block caches. The Y axis shows performance(IPC) normalized to a 64B block cache without victim cache or prefetching and the X axis shows the 26 SPEC 2000 benchmarks. The benchmarks are divided in floating point(left) and integer(right) and then sorted alphabetically. The figure shows that prefetching is beneficial for most benchmarks(14 benchmarks benefit). Some benchmarks are insensitive or have little performance increase or degradation and just three benchmarks experience performance degradation greater than 1%(lucas,gcc,mcf). Overall, by using prefetching average performance across all benchmarks increases by 4.9%. Victim caching is less beneficial with only three benchmarks (apsi, fma3d and crafty) seeing performance increase. The average

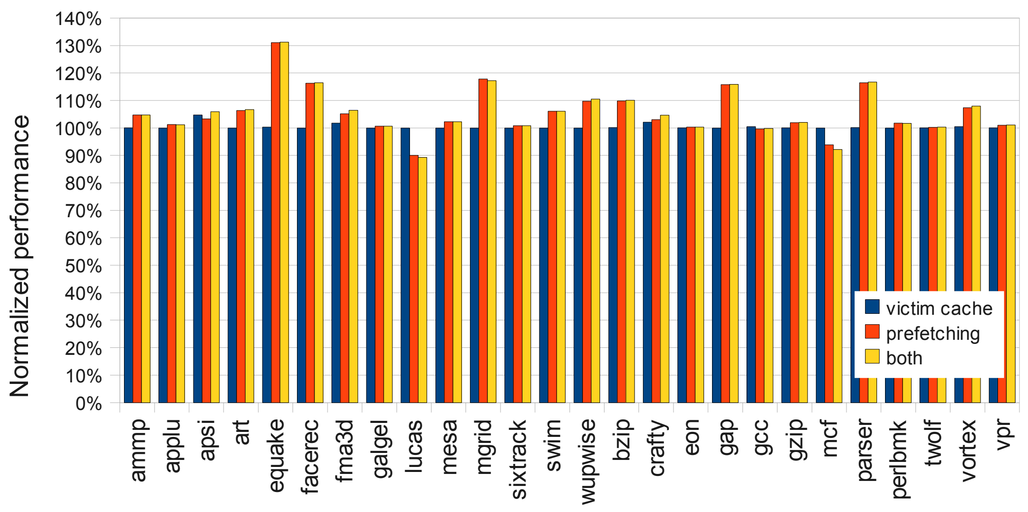


Figure 11: Performance improvements from using a victim cache, prefetching and the combination of both for a 32B block cache. Results are normalized to a 32B block cache configuration with prefetching and victim caching disabled.

performance improvement from victim caching for all benchmarks is 0.5%. This leads us to believe that, at least for our processor configuration, victim caching is not beneficial to performance. When victim caching and prefetching are combined, average performance increases by 5.8%. This is due to the benchmark lucas where victim caching helps alleviate the performance loss caused by prefetching. In this case, prefetching causes pollution (replacement of useful blocks) in the L1 data cache and the victim cache helps by providing some of the replaced blocks when they are later requested. The average performance difference between using prefetching alone or using prefetching combined with a victim cache is less than 1%. This led us to omit victim caching from our baseline configuration.

Figure 11 shows a similar comparison for a 32B block cache. The figure shows on the Y axis, performance normalized to a 32B block cache without victim cache or prefetching while the X axis holds the 26 SPEC benchmarks. Similarly to the 64B cache, the figure shows that prefetching is beneficial to performance(5.6% average improvement) while victim caching is not(0.3% average improvement). As a result, we omit the victim cache from the 32B block cache baseline as well.

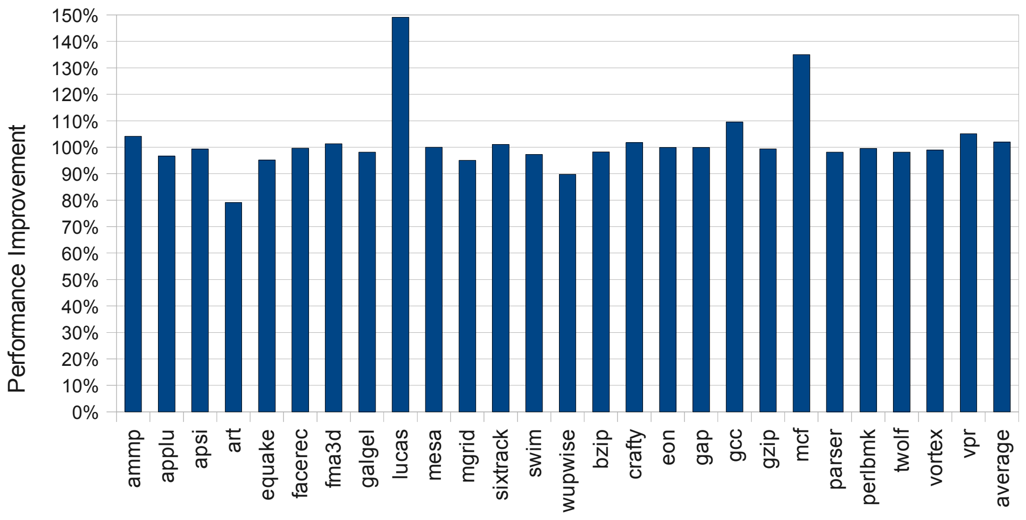


Figure 12: Performance improvements from using a 32B block cache over a 64B block cache

Figure 12 shows the performance improvement of using a 32B block cache over a 64B block cache. Both caches in this comparison utilize prefetching but not victim caching. The graph shows on the X axis the 26 benchmarks and on the Y axis the IPC improvement of using 32B block cache over a 64B block cache. Bars that are above 100% show performance benefit in favor of the 32B cache whereas bars below 100% show performance benefit for the 64B cache. From the figure we can see that for 3 benchmarks, performance is significantly better when using a 32B block(lucas, gcc, mcf) while two benchmarks(art, wupwise) benefit more from the 64B block. Overall, the performance improvement from using the 32B block is 1.8%. However, the 32B block cache is more costly to implement since double the area has to be allocated for storing block tags. We calculated that cache area(data bits+tag bits) increases by 5.1% as a result of reducing the block size from 64B to 32B. Since the performance benefit from the smaller block size is small, we decided to use the 64B cache as our baseline.

Figure 13 compares our baseline cache configuration to the word disabling/bit-fix configuration. In this scheme, word disabling is used for the L1 caches(I+D) and bit-fix id used for the L2 cache(this configuration was the best performing in [32]). The figure shows the 26 SPEC

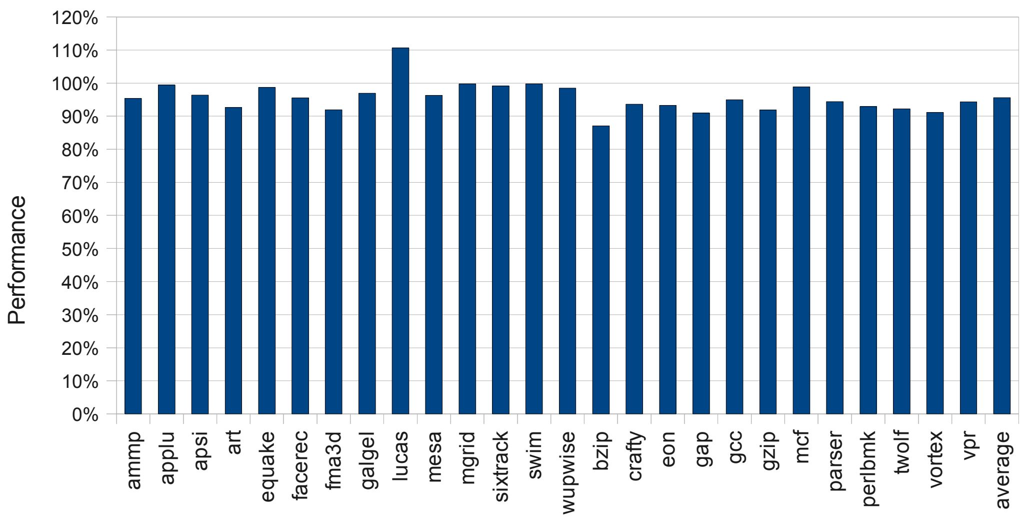


Figure 13: Performance degradation from using word disabling and bit-fix

benchmarks on the X axis and on the Y axis the performance of the disabling/bit-fix configuration normalized to the performance of our baseline cache. Both the disabling/bit-fix configuration and the baseline cache have prefetching enabled. The figure shows that performance suffers when word disabling and bit-fix are used. The average performance degradation across all benchmarks is 4.4%. The performance degradation is caused by the increased cache access times that the word disabling and bit-fix mechanisms require (one and three cycles respectively). This performance degradation comes at no benefit in high voltage operation where no faults are present.

5.3 Low Voltage Operation without Faults

Having established our baseline configuration in high voltage operation, we move to examine our cache configuration parameters for low voltage operation. When operating at low voltage, processor frequency decreases while memory frequency remains constant. As a result, the latency to access the main memory in terms of processor cycles decreases. This leads some tradeoffs in the cache hierarchy to shift. In our case, when the main memory access time decreases, prefetching becomes less beneficial since the penalty of cache misses becomes smaller. This is illustrated in

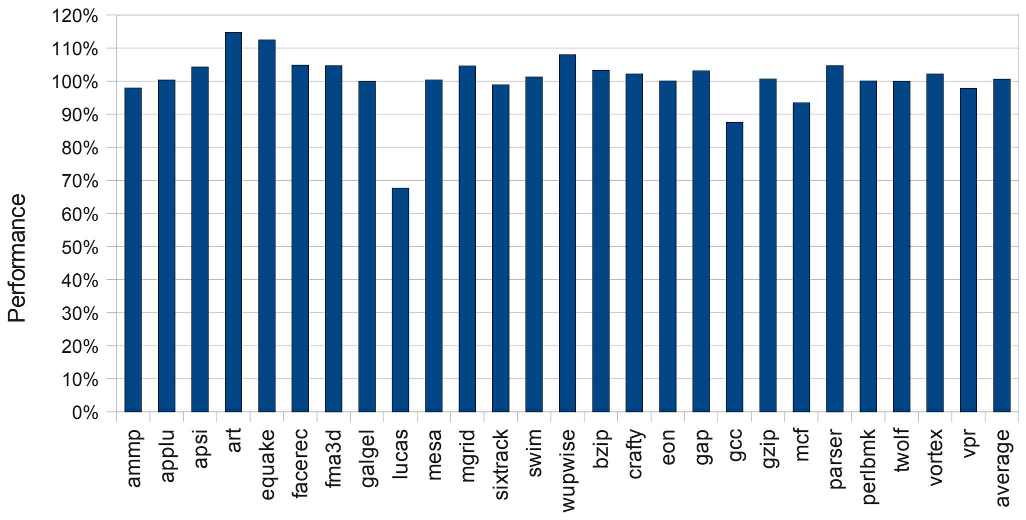


Figure 14: Performance improvement due to prefetching at low voltage operation

Figure 14. In this figure, the performance of the cache equipped with prefetching is normalized to the performance of a cache without prefetching. A 64B block configuration was used for this comparison. [2] Most benchmarks experience small performance improvements due to prefetching while for three benchmarks(lucas, gcc and mcf) performance is significantly reduced. The average performance improvement for all benchmarks is only 0.6%. These results indicate that for low voltage operation prefetching in not as useful as in normal operation. Because of this, we chose to switch prefetching off during low voltage operation in order to save energy. It is possible that a prefetching mechanism can be devised that is beneficial to performance at low voltage mode. Creating such a mechanism, however, was not in the scope of this work.

5.4 Low Voltage Operation with Faults Present

Block disabling deconfigures the cache reducing capacity and associativity depending on the number and location of faults. Because of this, the optimal cache configuration without faults may

[2]As with high voltage, there is very little overall performance benefit of using smaller block size when no faults are present so we use the 64B block configuration as baseline.

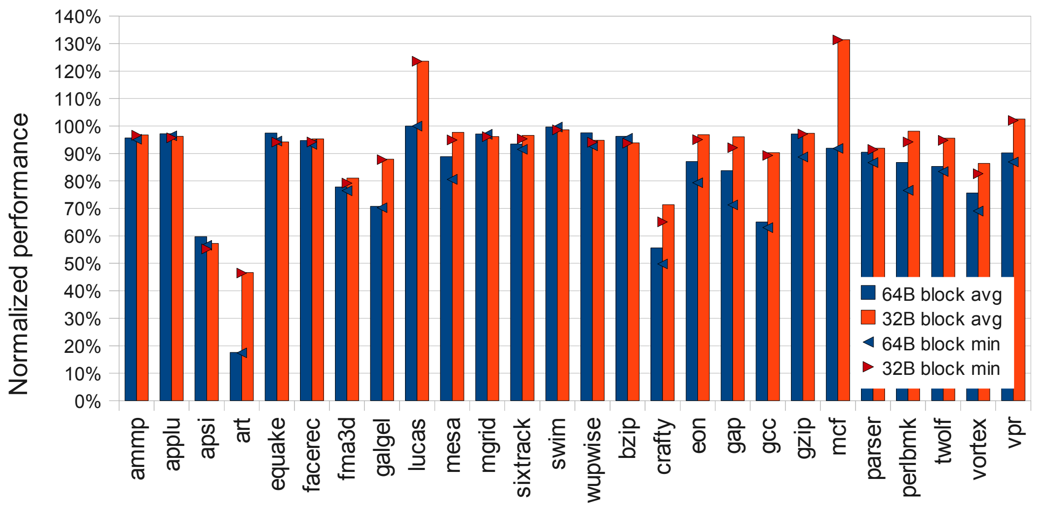


Figure 15: Block disabling with 64B block and 32B block. Results are normalized to a 64B block cache configuration without faults.

not be optimal in the presence of faults. In this section we will evaluate our baseline cache configuration in combination with block disabling and select the cache parameters that are beneficial for operation in the presence of faults. Our optimal block disabling configuration is then compared against a word disabling/bit-fix configuration.

5.4.1 Block Disabling 32B VS 64B

We begin by comparing our baseline scheme for low voltage operation(64B block) with a 32B block scheme. In both these configurations prefetching is switched off. We have seen in Chapter 4 that by using a smaller block, more capacity is available when block disabling is used. Additionally, we predicted analytically that the smaller block size would make performance more deterministic as it made clustering of faults in highly accessed sets more unlikely. However, it remained to be seen if these benefits would actually translate to better performance.

Figure 15 compares a 64B block and a 32B block cache configurations, both utilizing the block disabling mechanism for protection against faults. The figure shows, for each benchmark on the X-axis, the average and minimum performance of 30 random runs for the two configurations. Performance is normalized to a 64B cache without faults. From the figure it is clear that, when operating under faults, the 32B configuration offers superior performance. For the majority of benchmarks, performance is greatly increased when a 32B block is used. This happens because, as described in Chapter 4, the 32B configuration has more capacity available. Furthermore, performance is more deterministic for the 32B configuration with the minimum values of the random runs following more closely at the averages. Across all benchmarks, 64B block-disabling suffers a 16% drop in performance in the presence of faults while for 32B block-disabling, performance only drops by 8%. The average difference between the minimum and average values is 3.4% for the 64B configuration and 1.3% for the 32B configuration meaning that performance is significantly more deterministic for the smaller block configuration.

The above results clearly indicate that for block disabling, using a smaller block configuration is preferable. However, using a smaller block may not always be feasible since it increases the total cache area. When minimum area overhead is the key goal, the 64B configuration may be preferable. For this reason, we consider both the 32B and 64B configurations as alternatives to be used depending on whether the design goal is minimum area overhead or maximum performance. In the following subsection we will describe how to improve both configurations to increase performance in the presence of faults.

5.4.2 Block Disabling with Prefetching and Victim Caching

The analysis in Section 5.3 indicated that little was to gain from prefetching at low voltage mode. In the presence of faults however, this can change. Cache capacity decreases significantly when faults are present in a block disabling cache. Program access patterns that could be serviced efficiently without faults may cause misses when capacity is reduced due to faults. Consider the

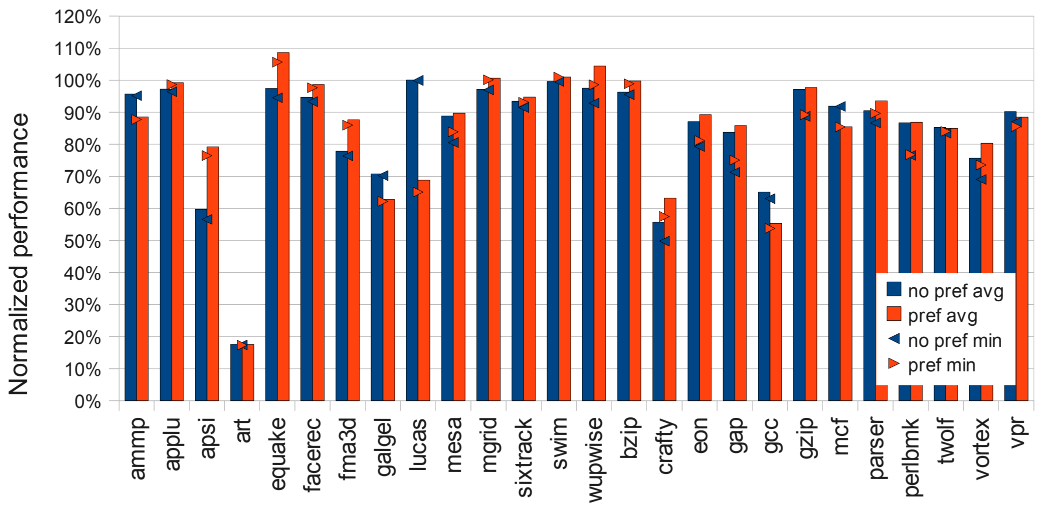


Figure 16: Block disabling with 64B block, with and without prefetching. Results are normalized to a fault-free 64B block cache configuration without prefetching.

following example. Throughout execution, a program frequently requests data from 7 blocks in the same cache set. Since the cache associativity is 8, after an initial miss the data will be constantly present in the cache and subsequent request will result in hits. In low voltage mode, one of the blocks of that set has a fault and as a result is disabled. Now, this set will consistently produce misses. Since the misses are consistent, an intelligent prefetching mechanism can identify the miss pattern and issue prefetches before absent blocks are demanded. On the other hand, issuing prefetches into sets that have reduced associativity because of faults may cause further misses. This can happen if the prefetches replace useful data in the cache.

To determine if prefetching is beneficial for our block disabling scheme we performed experiments. We evaluate prefetching with block disabling both for the 64B block and the 32B block cache configurations. Figure 16 shows the results for the 64B block caches. Both the average and the minimum of 30 random runs is shown. Results are normalized to a 64B block, fault-free cache without prefetching. The figure shows that prefetching is beneficial for some benchmarks(15 out of 26 benchmarks benefit) while for other benchmarks(6 out of 26) performance suffers due to

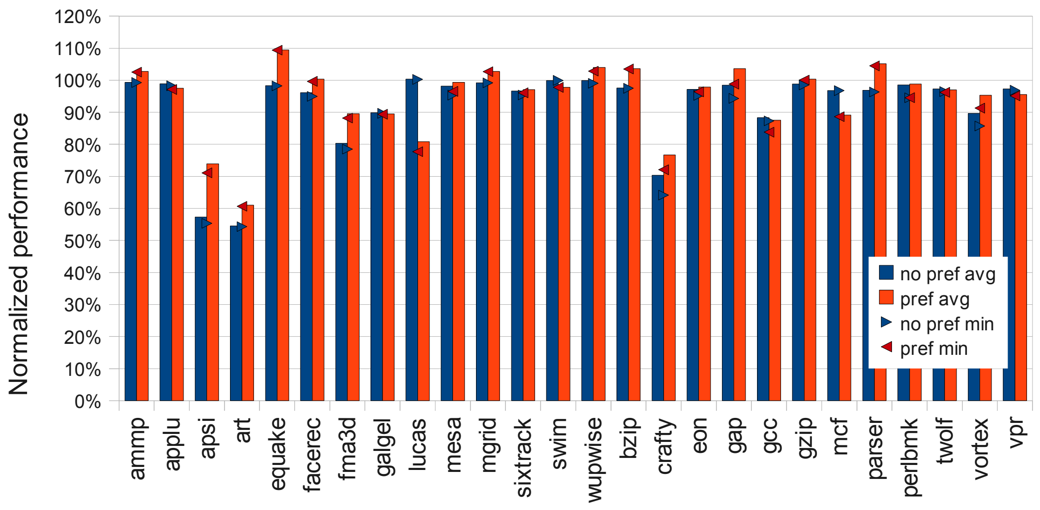


Figure 17: Block disabling with 32B block, with and without prefetching. Results are normalized to a fault-free 32B block cache configuration without prefetching.

prefetching. If we recall the results from Section 5.3, we can see that for benchmarks lucas, mcf and gcc performance drops even in the absence of faults when prefetching is used. However, for benchmarks ammp, galgel and vpr performance drops as a result of the combination of prefetching and faults. In these cases prefetching degrades performance as prefetching requests replace useful data in cache sets that have reduced associativity due to faults. On average, the performance benefit from using prefetching is less than 1%. However, this is not the case for the 32B block cache. Figure 17 shows the average and minimum of 30 runs for a 32B cache, with and without prefetching. Results on this figure are normalized to a 32B block cache without prefetching and without faults. As the figure shows, prefetching is for the majority of benchmarks beneficial to performance. Two benchmarks - lucas and mcf - experience performance degradation with prefetching enabled. We performed analysis similar to that in Section 5.3 but for a 32B block cache and found that for these two benchmarks perform badly when prefetching is enabled even in the absence of faults. In contrast, benchmark *apsi* sees little performance improvement from prefetching in fault free operation but when faults are introduced performance increases when prefetching is enabled.

Benchmark	Fault-free	Faults present	Difference
ammp	0.001	0.002	0.000
applu	0.022	2.039	2.017
apsi	0.831	9.075	**8.244**
art	0.002	0.003	0.001
bzip	0.002	0.003	0.000
crafty	2.949	14.014	**11.065**
eon	0.013	3.233	3.220
equake	0.002	0.241	0.238
facerec	0.002	0.010	0.008
fma3d	5.034	12.735	**7.701**
galgel	0.001	0.001	0.000
gap	0.006	0.947	0.941
gcc	2.336	5.614	3.278
gzip	0.003	0.003	0.001
lucas	0.000	0.000	0.000
mcf	0.001	0.002	0.001
mesa	0.011	0.798	0.787
mgrid	0.017	0.056	0.039
parser	0.081	0.267	0.186
perlbmk	0.012	0.920	0.908
sixtrack	0.007	0.768	0.761
swim	0.010	0.011	0.001
twolf	0.022	1.424	1.402
vortex	1.613	4.295	2.682
vpr	0.003	0.004	0.001
wupwise	0.007	0.029	0.022

Table 8: I-cache misses per 1K instructions with and without faults

This happens because at fault-free operation this benchmark has few IL1 cache misses so there is little need for prefetching. When faults are introduced, cache misses increase substantially and there is opportunity for prefetching to be useful. This effect is also true for benchmarks *fma3d* and *crafty*(for these benchmarks prefetching does not help significantly in fault-free operation). Table 8 shows the difference in misses per 1K instructions in fault-free and faulty operation for the instruction cache for all SPEC2000 benchmarks(using the 32B block configuration with prefetching enabled). As the table shows, the benchmarks that benefit from prefetching in the presence of faults see a significant increase in I-cache misses due to faults. Overall, for the 32B block cache, prefetching increases performance by 2.3% compared to not using prefetching. Prefetching is more useful in the 32B block configuration than in the 64B block configuration because, as we have seen in Section 4.3, it is less likely for faults to cluster in the same set in the smaller block

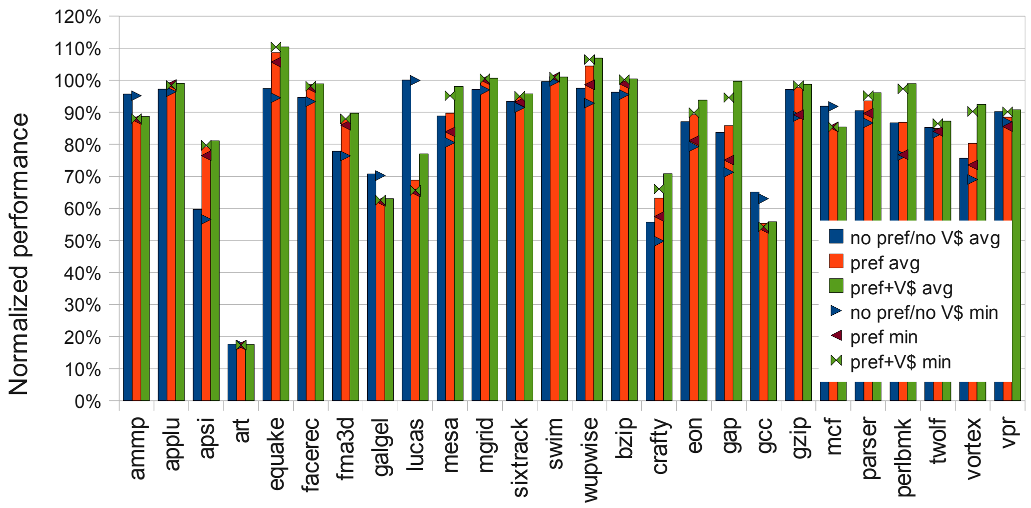


Figure 18: Block disabling with 64B block, with prefetching(pref) and victim caching(V$). Results are normalized to a fault-free 64B block cache configuration without prefetching or victim caching.

configuration. As a result, prefetching is less likely to replace useful data since associativity will likely be higher.

In [16] it was shown that, for a 64B cache configuration using block disabling, using a victim cache helps reduce performance non determinism caused by the random placement of faults in the cache. The victim cache helps by relieving pressure from frequently accessed sets that happen to have reduced associativity due to faults. Although we have seen in Section 5.2 that victim caching is not substantially beneficial to performance for our baseline configuration, we reexamine its use in the presence of faults and prefetching. Figure 18 compares block disabling, block disabling with prefetching and block disabling with prefetching and victim caching for a 64B cache configuration. These results are normalized to a 64B, fault free cache. The figure shows that victim caching together with prefetching is preferable than using prefetching alone. This happens because the victim cache helps reduce the misses caused when the prefetching mechanisms issue prefetches into sets with limited associativity. When compared to the baseline block disabling configuration(prefetching and victim caching disabled), the pref+victim cache configuration performs

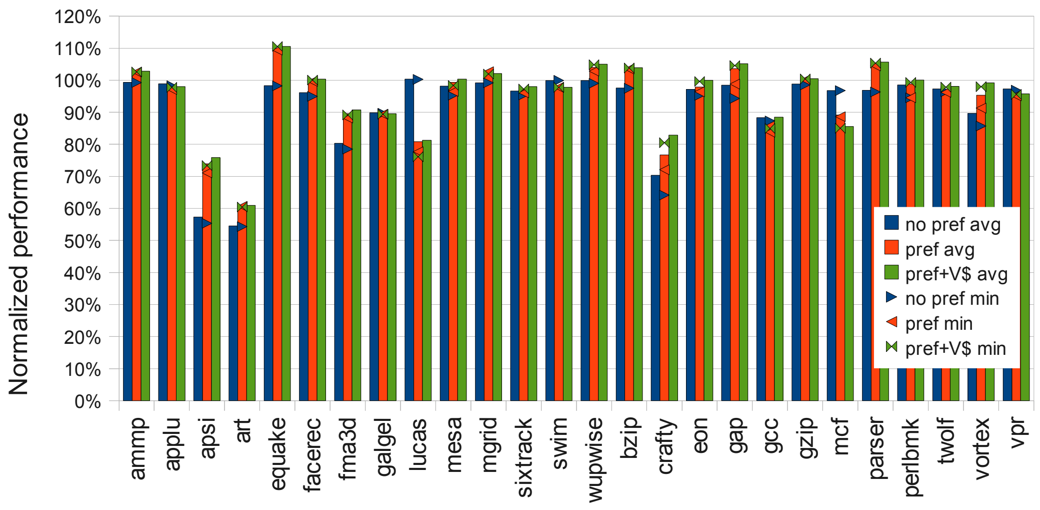


Figure 19: Block disabling with 32B block, with prefetching(pref) and victim caching(V$). Results are normalized to a fault-free 32B block cache configuration without prefetching or victim caching.

better(4% average improvement) whereas by using prefetching alone performance improvement is insignificant. Also, the victim cache helps with performance variation. On average the difference between average and minimum performance for each benchmark is 3.4% for the baseline block disabling and 1.6% when a victim cache is used. This confirms, as in [16], that the victim cache is helpful in terms of reducing performance variability.

Similar analysis was performed for the 32B cache configuration and the results are shown in Figure 19. These results are normalized to a 32B cache without victim caching or prefetching enabled. The figure shows that there is little performance gain from using a victim cache together with prefetching. Average performance improvement is less than 1%. However, performance predictability is increased. On average the difference between average and minimum performance when no victim cache is present is 1.5%. For the victim cache configuration the difference is 0.9%. The improvement in performance predictability is not as pronounced as in the 64B block configuration. This happens because, as we have seen from the analysis in Section 4.3, the 32B cache configuration is more resilient to clustered faults and as a result, less prone to performance variations.

The above results indicate that the optimal block disabling configuration for a 64B cache combines prefetching and victim caching while for 32B caches prefetching alone could be sufficient. Since the hardware cost for a victim cache is small, we chose to include a victim in the 32B block configuration as well since it increases performance predictability and acts as an additional safeguard against performance variation. In the following section we will compare these two configurations against a cache that is equipped with word disabling and bit-fix.

5.5 Comparison of Block Disabling, Word Disabling and Bit-fix in Low Voltage Operation

In Section 5.2 we have seen that, due to mechanism overheads, a cache equipped with word disable and bit fix suffers significant performance degradation in fault free operation. This is not the case for block disabling. In this section we compare the two best performing block disabling configurations(32B and 64B) against a cache equipped with word disable for the L1 caches and bit-fix for the L2 cache.

Figure 20 compares word-disable/bit-fix against block disabling for a 64B cache. Both the block disabling cache and the word-disable/bit-fix cache have prefetching and victim caching enabled. Results are normalized to a faultless 64B cache with prefetching and victim caching disabled(which is the best performing configuration for low voltage operation without faults). From the figure we can see that for some benchmarks(13/26), block disabling performs better than word-disable/bit-fix. For other benchmarks(5/26), word-disable/bit-fix has the advantage. For three benchmarks (art, galgel and gcc) word-disable/bit-fix performs significantly better. On average, the two mechanisms perform equally. Average performance degradation compared to the fault free cache is 11.6% for both configurations. If we recall the analysis in Section 4.1 we can see

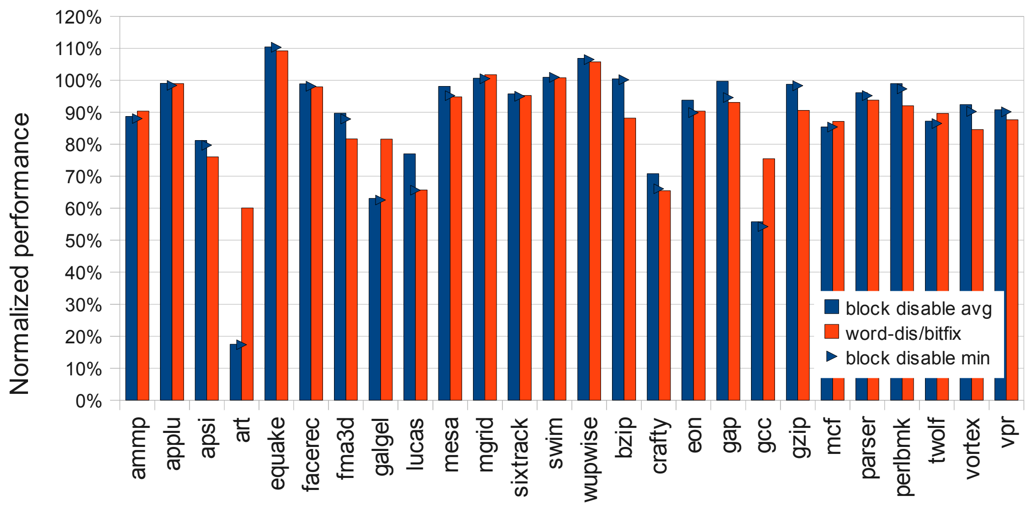


Figure 20: Comparison of block disabling and word-disabling/bit-fix for a 64B cache. Results are normalized to a fault-free 64B block cache configuration without prefetching or victim caching.

that for p_{fail} 0.001, block disabling has a capacity advantage over word disabling. However, for this p_{fail}, bit-fix has a capacity advantage over block disabling. As a result, neither configuration has a clear advantage over the other and this is reflected in the results.

Figure 20 compares word-disable/bit-fix against block disabling for a 32B cache. Both the block disabling cache and the word-disable/bit-fix cache have prefetching and victim caching enabled. Results are normalized to a 32B cache without faults and with prefetching and victim caching disabled. The figure shows that for 32B caches, block disabling outperforms the word-disable/bit-fix configuration. For the majority of benchmarks, block disabling performs better and for the rest it closely follows word-disable/bit-fix. Average performance degradation is 4.3% for block disabling and 11.5% for word-disable/bit-fix. Note that for word-disable/bit-fix performance is almost unchanged between the 64B and 32B configurations. This happens because capacity and associativity remain the same regardless of the block size.

From the above results it is clear that when high performance is required, block disabling using 32B block is the preferred solution. However, if minimum cost is the goal, block disabling using

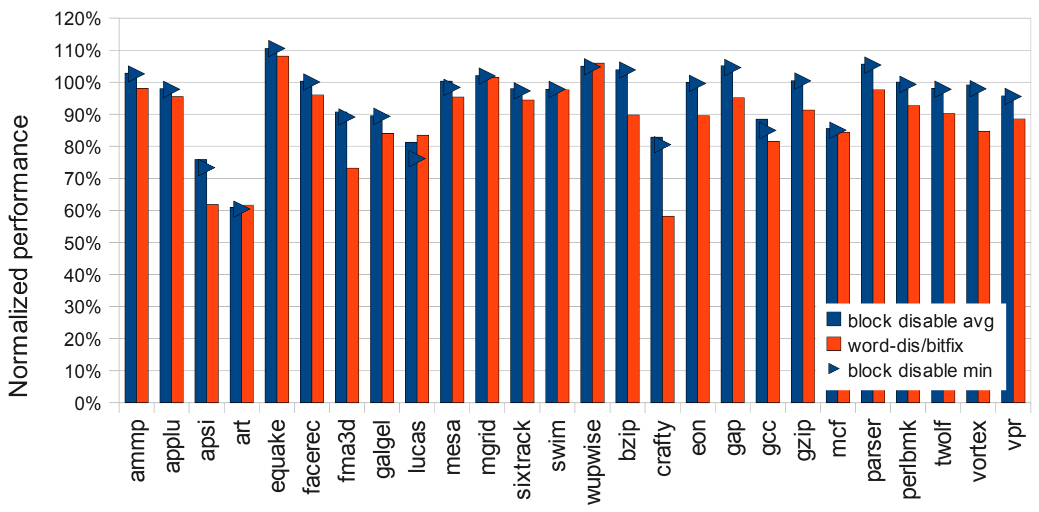


Figure 21: Comparison of block disabling and word-disabling/bit-fix for a 32B cache. Results are normalized to a fault-free 32B block cache configuration without prefetching or victim caching.

64B block may be preferred as it performs similarly to word-disable/bit-fix, is simpler and can be implemented with minimal area overhead.

5.6 Implications of Set Failure

The results we have presented so far for block disabling where produced using 30 randomly created fault maps. This approach is useful for capturing average behaviour. However, as we have seen in Section 4.3, it is possible for a set to be completely faulty(set failure). In the randomly created fault maps that we produced set failure did not happen. To assess the performance impact of set failure we created specific fault maps where the most frequently accessed set is completely faulty(no blocks available). These fault maps have faults only in the most useful(frequently accessed) set. We present results for each of the caches separately(in each set of experiments we disable the most useful set of only one cache). For the experiments below, we used our best performing block disabling configuration for 64B block caches (victim caching and prefetching where enabled).

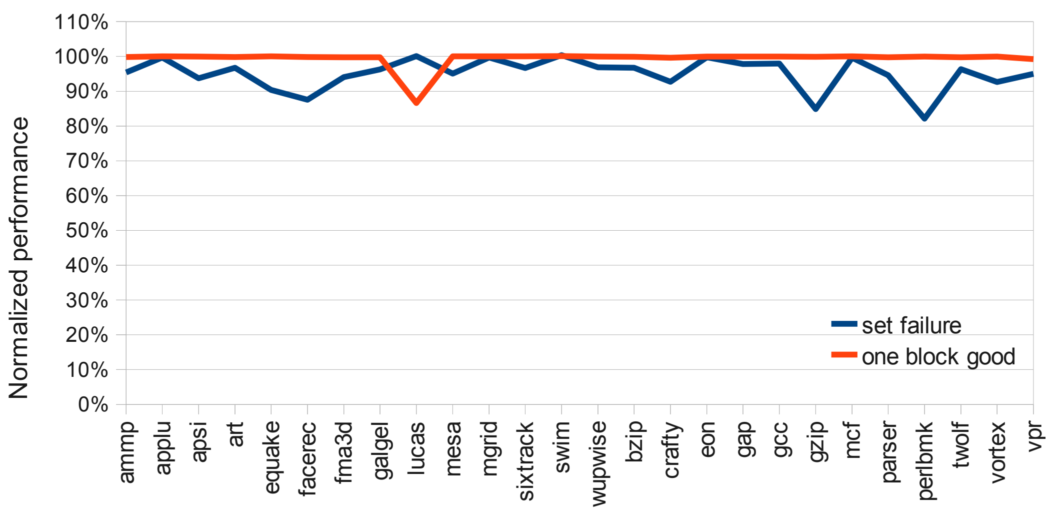


Figure 22: Effect of set failure on the L1 data cache

Figure 22 shows the effect of set failure for the DL1 cache. As the figure shows, set failure in the DL1 cache can significantly affect performance. Out of 26 benchmarks, four have more than 10% performance degradation. For benchmark *perl*, performance drops by more than 15%. The figure also shows what happens if one block is fault free instead of disabling all blocks in the set(indicated as *one block good* in the figure). As the figure shows, when one block is operational most of the performance degradation is recovered. Benchmark *lucas* is the odd case where having one block good actually degrades performance. In this case, having one block available does reduce DL1 misses. However, the L2 prefetching mechanism is affected since the address stream that reaches the L2 cache changes and this results in bad prefetching behaviour(many unnecessary prefetches).

Figure 23 shows the performance degradation from set failure in the IL1 cache. For the IL1, performance degradation due to set failure is more pronounced. Out of 26 benchmarks, 11 benchmarks experience more than 10% performance degradation, 5 benchmarks drop more than 20% and for two benchmarks(art and bzip) performance drops nearly by 50%. This happens because the IL1 hit ratio is very high and nearly all misses caused by faults cause pipeline stalls. As in the

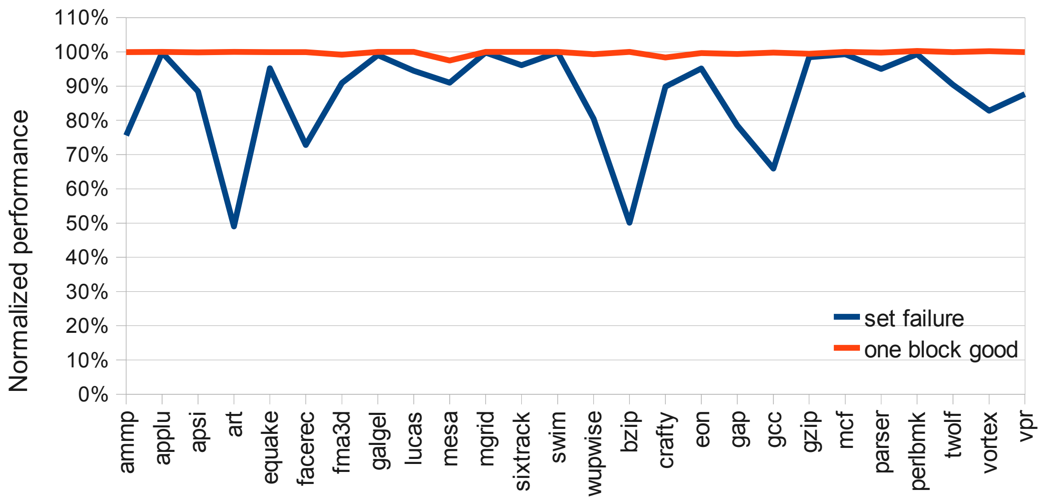


Figure 23: Effect of set failure on the L1 instruction cache

DL1 cache, by allowing one block in the set to be fault-free most of the performance degradation is recovered.

For the L2 cache(Figure24), only 3 benchmarks (lucas, mcf and vpr) see performance degradation due to set failure. This happens because L2 accesses are spread out in all sets(no one set has many accesses). As in the other caches, when one block is working, the performance degradation is recovered.

The above results indicate that set failure can significantly affect performance for the L1 caches. Since for a 64B L1 cache the probability of set failure is significant (as we have seen in Section 4.3), we believe it is important to provide protection against it. One way to protect against set failure is to make sure that one block is available in each set. As we have seen in the above figures, when one block is available most of the performance degradation is recovered. Having one block available can be achieved using spares. The cost of this approach is small since few spares are needed as the probability of multiple set failures happening at the same chip is very small. Alternatively, a 32B block cache can be used which reduces the probability of set failure

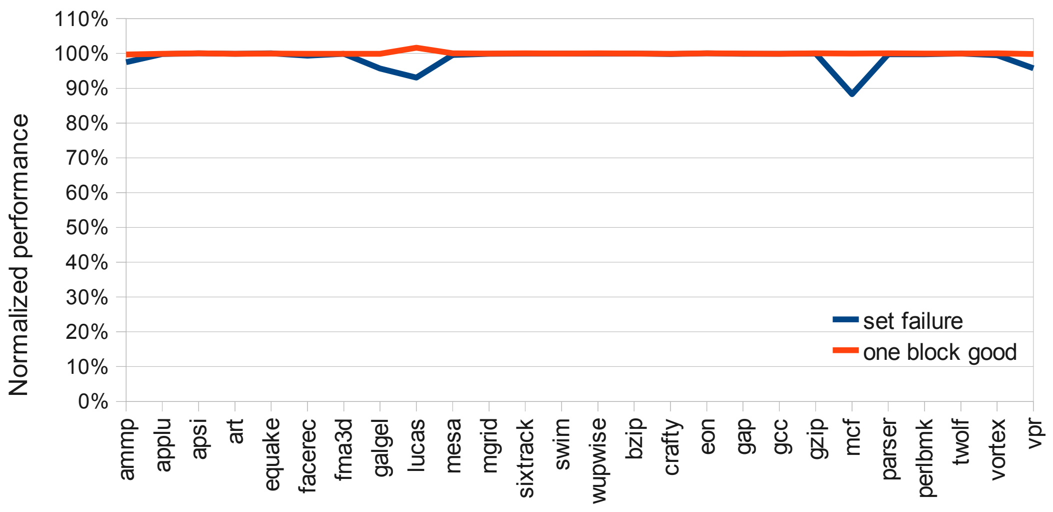


Figure 24: Effect of set failure on the L2 cache

significantly. For the L2 caches, set failure is more unlikely. At the same time, if set failure happens the effects are not as pronounced as in the L1 caches. This leads us to believe that set failure for the L2 caches is not as critical.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this work we presented our approach for providing reliable cache operation in the presence of many faults. We have seen that most of the previously proposed mechanisms that can handle large numbers of faults in the cache are costly in terms of area and complex to implement. We proposed to use a very simple reliability mechanism, block disabling, and combine it with careful selection of cache parameters and performance enhancing mechanisms such as prefetching and victim caching. This approach is both simple to implement -since the technology required is already present in modern processors- and has little area overhead. We compared our approach against two recently proposed mechanisms that can handle high fault rates in caches: word disabling and bit-fix.

Using probability analysis, we showed that block disabling capacity is highly dependent on the cache block size. By using a smaller block, block-disabling capacity was shown to be higher than word disabling and bit-fix for a wide range of cell probability of failure(p_{fail}). Furthermore, we have seen that when p_{fail} is high, word disabling suffers from whole cache failure and yield is affected. In contrast, our block disabling approach does not suffer from this problem. Block disabling may suffer from performance degradation if many faults happen to occur in a frequently accessed set. Using analysis we showed that by selecting a smaller block size and increasing cache associativity, the probability of performance degradation due to clustered faults is greatly reduced.

Using simulation experiments we showed that block disabling with a 32B block cache performs better than using 64B block cache since more capacity is available and performance variations are less likely. Additionally, we have shown that performance mechanisms that are not particularly useful in fault-free operation can be of great assistance when faults are present. Specifically, using a victim cache in combination with block disabling is beneficial to performance when a 64B block cache is used. A victim cache is also helpful for all configurations to reduce performance variability caused by clustered faults. We have also seen that prefetching is not useful in fault-free low voltage operation since memory access time is reduced making L2 misses less costly(in terms of performance). In the presence of faults, however, we have seen that prefetching becomes useful. This happens because reduced associativity caused by faults increases cache misses. Furthermore, we have shown that prefetching is more useful for the 32B block cache configuration than for the 64B block configuration. This happens because prefetching is less likely to cause pollution as the associativity in the 32B configuration is higher.

Our best performing block disabling configurations where compared against a cache equipped with word disabling and bit-fix. We showed that our 32B block disabling configuration consistently outperforms word disabling and bit-fix for roughly equal hardware cost. Furthermore, our 64B block disabling configuration performs similarly to word disabling and bit-fix, requires less area overhead and is simpler to implement.

6.2 Future Work

One direction for future work would be to achieve the benefits of smaller block size but without the associated extra tag cost. We have seen from our analysis that the best performing configuration is to use block disabling with a cache configuration that uses 32B blocks. This, however, is costly since it requires that the tag array is doubled. It would be useful if the benefits of the smaller block size could be achieved without paying the extra area cost. One possible way of doing this is through the use of subblocking. By disabling subblocks, the same capacity can be achieved as using a smaller block size without extra tag cost. However, implementing this method efficiently is not straightforward. For example, an address may request data that is present in the cache(the tag exists) but the subblock that contains the data is disabled. One way to deal with this problem is to swap the positions of subblocks inside the block upon a subblock miss caused by disabling.

In Chapter 5 we have seen that prefetching is beneficial to performance when faults are present. In our experiments we have used two prefetching mechanisms designed and tuned for fault free operation. A promising direction for future work is to tune or redesign the prefetchers to further help with performance degradation caused by faults. Ideally, we do not want costly hardware, such as the prefetcher, to be devoted specifically for dealing with faults. To avoid this, the prefetcher should be able to switch configurations when operating at high or low voltage(with and without faults) and be optimally beneficial to performance in both modes.

Another interesting direction for future work is to explore the relation between low voltage operation and power/energy gains. We know that by operating below Vccmin we can save power. However, by operating below Vccmin we introduce faults in the caches. To deal with these faults we use reliability mechanisms that impact performance(e.g. block disabling disables parts of the cache). Because performance is affected, power savings from low voltage operation are reduced(e.g. processor cycles are wasted waiting for cache misses). The more we lower voltage the more power we can save, but at the same time we introduce more faults that may hinder performance. Future work can examine this relation with the goal of finding the optimal voltage where the highest power savings can be achieved.

Appendix A

Selecting the Prefetching Strategy

A.1 Description of Baseline Prefetching Mechanism

One of the goals of this work is to evaluate block disabling in the context of a modern, high-end cache hierarchy. For this reason we decided to include a state of the art prefetcher to our cache configuration. We chose the prefetcher proposed in [24], winner of best paper award in the 2009 Data Prefetching Competition. In this section we provide an overview of how this mechanism works.

The prefetching configuration used in [24] is comprised of an L1 sequential tagged prefetcher(STP) and an L2 PDFCM prefetcher. The L1 sequential tagged prefetcher [23] is used to prefetch blocks in the L1 data. It requires one extra bit per cache block called the prefetch bit. When a miss occurs in the L1 data cache(DL1), a prefetch is issued to fetch the next block in the cache. When this block is fetched from the lower level cache into DL1, its matching prefetch bit is enabled. A subsequent hit to a block marked as prefetch will initiate further prefetch requests for the next *degree* cache blocks. Additionally, after a hit to a block marked as prefetch, the prefetch bit is disabled.

The following is an example of how the mechanism works. Assume we have a miss on cache block A. The prefetching mechanism will issue a prefetch request for block A+1. When block

A+1 is inserted in the cache its prefetch bit will be set to 1. Assume that after a series of cache events block A+1 is requested. Since its prefetch bit is enabled, the prefetching mechanism will issue additional *degree* prefetch requests. If *degree* is 4, the mechanism will issue requests blocks A+2,A+3,A+4 and A+5. This policy allows the tagged prefetcher to respond to new misses in the cache (by issuing a prefetch for each cache miss) and to further prefetch blocks that appear to be useful (by issuing further prefetches for a requested prefetch block). This mechanism is inexpensive as it requires only a single bit per block and is shown in [23] to be cost-effective.

For the L2 cache, the configuration in [24] used a PDFCM prefetcher [22]. PDFCM is a correlating prefetcher that keeps a compressed history of addresses for each memory instruction that misses in the cache or hits on a prefetched block. Prefetch hits and misses are referred as training addresses. The mechanism uses the address history to identify miss patterns and issue accurate, useful prefetches. To perform these functions, the PDFCM prefetcher uses a history table(HT) and a delta table(DT). The history table is indexed using the PC of the memory instruction accessing the cache and holds, in each field, the PC tag of the memory instruction, the last training address issued by this memory instruction, the hashed sequence of deltas issued by this instruction(compressed history) and two confidence bits.

The compressed history is used to index the delta table in order to find out the next delta. When a memory instruction misses in the cache or hits a prefetched block, the HT is accessed using as index the PC of the instruction. The corresponding HT entry is then updated using the address of the accessing memory instruction. A new history is calculated using the last address stored in the entry and the new address carried by the memory instruction. The last address is then replaced with the new training address. Next, the new history is used to access the DT in order to give a prediction about what the next address for this instruction will be. A prefetch is then issued using this address. On cycles where the cache is idle, the mechanism calculates new miss address using the last address to be prefetched and the DT. These addresses are used to prefetch new blocks in the cache. A maximum of *degree* prefetches can be issued this way, where *degree* is a value that can be either static or controlled dynamically based on program behaviour.

In the configuration used in [24], degree is increased when the number of L1 accesses in a period of 64K cycles(one epoch) exceeds the number of L1 accesses in the previous period. Similarly, when the number of L1 accesses in an epoch is less than the number of cycles in the previous epoch the degree is decreased. This heuristic attempts to capture fluctuations in performance and adapt the prefetching degree accordingly.

A.2 Our Prefetching Configuration

In contrast to configuration used in [24], our simulator accurately models an out of order pipeline with mispredictions, wrong path execution and speculative cache accesses. As a result, the parameters used to tune the prefetchers in [24] where not suitable for achieving good performance in our case. For this reason we performed changes to the prefetchers as well as tuning specific to our simulation environment. The changes in the prefetching mechanisms are described below.

In [24], the PDFCM prefetcher is trained using addresses that are either L2 demand misses(not issued by the STP prefetcher) or first references to blocks prefetched by PDFCM. We found this policy useful when PDFCM is used without an L1 prefetcher present. When we combined STP and PDFCM, we found that performance suffered. This happens because the L1 prefetcher(STP) interferes with the miss stream that the L2 prefetcher(PDFCM) receives. As a result, the L2 prefetcher is not properly trained and produces less useful prefetches. This is more of a problem for our simulation environment since we allow wrong path cache accesses. As a solution to this problem, we modified PDFCM to be trained using both demand and prefetch L2 misses. By using this approach, PDFCM receives the whole miss stream from L1 and can produce more meaningful prefetches. Useless L1 prefetches(prefetches that are not consistent over time) are filtered automatically by PDFCM using confidence counters.

In [24], the PDFCM prefetch degree is controlled by tracking increases and decreases in L1 accesses. We found this heuristic to be inaccurate in our simulation environment. For our implementation, we changed the degree controller to reflect prefetch usefulness. Specifically, PDFCM prefetch degree is increased when a number of prefetched blocks are requested in L2. Similarly when a number of prefetch blocks are replaced, the prefetch degree is reduced. We found this approach to perform better than the method used in [24].

Additionally, we found that the adaptive degree used in [23] for STP performed better than the static degree used in [24]. The static degree prefetches 4 blocks after a hit occurs on a prefetched block. The adaptive degree prefetches *degree* blocks on a prefetch hit where *degree* is controlled by an adaptive degree controller similar to the one we used for PDFCM. In both cases a miss in the cache triggers a prefetch for the next line in the cache. In our environment we found that this may degrade performance in benchmarks that are unfriendly to prefetching. For this reason we modified the STP prefetcher so that it completely turns off if many replacements of prefetched blocks occur.

Figures 10 and 11 show the performance of 64B and 32B caches equipped with prefetching relative to a cache without prefetching. An STP prefetcher was used for the L1 caches and PDFCM was used for L2. The prefetchers in these results contain all of the modifications that we described above. As the figures show, prefetching is beneficial for many benchmarks and rarely deteriorates performance. Further enhancements could be made to the prefetching mechanisms. For example, the prefetchers could be trained only with non-speculative cache accesses or with speculative accesses for which we have high confidence(using confidence bits from the branch predictor or indirect jump predictor). For the requirements of our work, however, we found the performance of the prefetching scheme adequate.

Appendix B

Simulator Code Changes

This appendix describes the changes that we made to the sim-alpha simulator as part of this work. Below is a list of all major code changes performed.

- **Block disabling**. To simulate block disabling we use a fault map that is generated externally. Each fault map corresponds to one cache(e.g. IL1, DL1, L2) and contains one entry per set of the cache. Each entry holds the number of available blocks for that set. The fault map is given as input to the simulator and is read during the cache initialization method. For each cache set to be created, instead of creating *associativity* blocks, we create the number of blocks specified in the fault map.

- **Variable cache latency**. By default, sim-alpha does not allow changing the access latency to the IL1 cache. To properly simulate word disabling, we needed to increase the access time to IL1. we achieved this by adding extra latency whenever a branch misprediction or load-store trap happens. The extra latency is controlled by a user specified parameter.

- **Bypassing disabled sets**. Due to block disabling, a set may become completely faulty(no blocks available). To continue operation in the presence of a disabled set we implemented bypassing. When a block is to be inserted in a disabled set, we insert it in the fast hit buffer instead. The fast hit buffer in fault free operation holds the data for the last block to hit in the cache. When operating under faults, the fast hit buffer performs two functions. It can hold the last block to hit in the cache or a bypassed block. By using this approach multiple subsequent accesses to a disabled set do not cause multiple cache misses. However, a hit to another block will evict the bypassed block and a later request for the bypassed block will result in a miss.

- **Counting cache accesses per set**. For the needs of the analysis in Section 4.3 we needed to know the number of accesses for each set in the cache. We achieved this easily using an array for each cache. Each array has an entry for each set in the cache and is updated when the corresponding set is accessed.

- **Skip write misses**. We modified the simulator so that when a write command misses in the L2 cache it will not initiate a DRAM access. This only happens if the L1 and L2 block sizes are the same.

- **Prefetching**. The description of our prefetching configuration is described in Appendix A.

Bibliography

[1] Abella, J., Carretero, J., Chaparro, P., Vera, X., and González, A. Low vc-cmin fault-tolerant cache with highly predictable performance. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2009), ACM, pp. 111–121.

[2] Ansari, A., Gupta, S., Feng, S., and Mahlke, S. Zerehcache: armoring cache architectures in high defect density technologies. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (2009), pp. 100–110.

[3] Asenov, A., Brown, A., Davies, J., Kaya, S., and Slavcheva, G. Simulation of intrinsic parameter fluctuations in decananometer and nanometer-scale mosfets. *Electron Devices, IEEE Transactions on 50*, 9 (sept. 2003), 1837 – 1852.

[4] Bhavnagarwala, A., Tang, X., and Meindl, J. The impact of intrinsic device fluctuations on cmos sram cell stability. *Solid-State Circuits, IEEE Journal of 36*, 4 (apr 2001), 658 –665.

[5] Borkar, S., Karnik, T., Narendra, S., Tschanz, J., Keshavarzi, A., and De, V. Parameter variations and impact on circuits and microarchitecture. In *DAC '03: Proceedings of the 40th annual Design Automation Conference* (New York, NY, USA, 2003), ACM, pp. 338–342.

[6] Bowman, K., Brooks, D., Wei, G.-Y., and Wilkerson, C. Tutorial on design variability: Trends, models and design solutions. In *Tutorial at MICRO* (Nov. 2008).

[7] Bowman, K., Duvall, S., and Meindl, J. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *Solid-State Circuits, IEEE Journal of 37*, 2 (feb 2002), 183 –190.

[8] Chang, J., Huang, M., Shoemaker, J., Benoit, J., Chen, S.-L., Chen, W., Chiu, S., Ganesan, R., Leong, G., Lukka, V., Rusu, S., and Srivastava, D. The 65-nm 16-mb shared on-die l3 cache for the dual-core intel xeon processor 7100 series. *Solid-State Circuits, IEEE Journal of 42*, 4 (2007), 846 –852.

[9] Desikan, R., Burger, D., Keckler, S., and Austin, T. Sim-alpha: a validated execution driven Alpha 21264 simulator. Tech. Rep. TR-01-23, CS Dept., University of Texas at Austin, 2001.

[10] Hamerly, G., Perelman, E., and Calder, B. How to use simpoint to pick simulation points, 2004.

[11] JOUPPI, N. P. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. *SIGARCH Comput. Archit. News 18* (May 1990), 364–373.

[12] KOH, C.-K., WONG, W.-F., CHEN, Y., AND LI, H. Tolerating process variations in large, set-associative caches: The buddy cache. *ACM Trans. Archit. Code Optim. 6*, 2 (2009), 1–34.

[13] KULKARNI, J. P., KIM, K., AND ROY, K. A 160 mv, fully differential, robust schmitt trigger based sub-threshold sram. pp. 171–176.

[14] KULKARNI, J. P., KIM, K., AND ROY, K. A 160 mv, fully differential, robust schmitt trigger based sub-threshold sram. In *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design* (New York, NY, USA, 2007), ACM, pp. 171–176.

[15] LADAS, N., SAZEIDES, Y., AND DESMET, V. Performance implications of faults in prediction arrays. In *2nd HiPEAC Workshop on Design for Reliability (DFR 2010).*

[16] LADAS, N., SAZEIDES, Y., AND DESMET, V. Performance-effective operation below vcc-min. In *ISPASS-2010: 2010 IEEE International Symposium on Performance Analysis of Systems and Software* (2010), pp. 223–234.

[17] LEE, H., CHO, S., AND CHILDERS, B. R. Performance of graceful degradation for cache faults. In *ISVLSI '07: Proceedings of the IEEE Computer Society Annual Symposium on VLSI* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 409–415.

[18] LIANG, X., WEI, G.-Y., AND BROOKS, D. Revival: A variation-tolerant architecture using voltage interpolation and variable latency. pp. 191–202.

[19] NASSIF, S. R., MEHTA, N., AND CAO, Y. A resilience roadmap. In *DATE* (2010), pp. 1011–1016.

[20] PATTERSON, D. A., GARRISON, P., HILL, M., LIOUPIS, D., NYBERG, C., SIPPEL, T., AND DYKE, K. V. Architecture of a vlsi instruction cache for a risc. In *Proceedings of the 10th annual international symposium on Computer architecture* (New York, NY, USA, 1983), ISCA '83, ACM, pp. 108–116.

[21] POUR, A. F., AND HILL, M. D. Performance implications of tolerating cache faults. *IEEE Transactions on Computers 42*, 3 (Mar. 1993), 257–267.

[22] RAMOS, L. M., BRIZ, J. L., IBÁÑEZ, P. E., AND VIÑALS, V. Data prefetching in a cache hierarchy with high bandwidth and capacity. *SIGARCH Comput. Archit. News 35* (September 2007), 37–44.

[23] RAMOS, L. M., BRIZ, J. L., IBÁÑEZ, P. E., AND VIÑALS, V. Low-cost adaptive data prefetching. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing* (Berlin, Heidelberg, 2008), Euro-Par '08, Springer-Verlag, pp. 327–336.

[24] RAMOS, L. M., BRIZ, JOSÉ LUIS IBÁÑEZ, P. E., AND VIÑALS, V. Multi-level adaptive prefetching based on performance gradient tracking. In *DPC-1: The 1st JILP Data Prefetching Championship* (2009), IEEE JILP.

[25] Roberts, D., Kim, N. S., and Mudge, T. N. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. *Microprocessors and Microsystems - Embedded Hardware Design 32*, 5-6 (May 2008), 244–253.

[26] Sarangi, S., Greskamp, B., Tiwari, A., and Torrellas, J. EVAL: Utilizing processors with variation-induced timing errors. pp. 423–434.

[27] Sasan, A., Homayoun, H., Eltawil, A., and Kurdahi, F. A fault tolerant cache architecture for sub 500mv operation: resizable data composer cache (rdc-cache). In *Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems* (New York, NY, USA, 2009), CASES '09, ACM, pp. 251–260.

[28] Sazeides, Y., Kourouyiannis, C., Ladas, N., and Desmet, V. Protecting prediction arrays against faults. In *IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE).*

[29] Sohi, G. S. Cache memory organization to enhance the yield of high performance VLSI processors. *IEEE Transactions on Computers 38*, 4 (Apr. 1989), 484–492.

[30] The International Technology Roadmap for Semiconductors. Edition 2009. Tech. rep., ITRS, 2009.

[31] Tiwari, A., Sarangi, S. R., and Torrellas, J. Recycle: : pipeline adaptation to tolerate process variation. pp. 323–334.

[32] Wilkerson, C., Gao, H., Alameldeen, A. R., Chishti, Z., Khellah, M., and Lu, S.-L. Trading off cache capacity for reliability to enable low voltage operation. pp. 203–214.