Archipelago: A Polymorphic Cache Design for Enabling Robust Near-Threshold Operation

Amin Ansari Shuguang Feng Shantanu Gupta Scott Mahlke

Advanced Computer Architecture Laboratory
University of Michigan, Ann Arbor, MI 48109
{ansary, shoe, shangupt, mahlke}@umich.edu

ABSTRACT

Extreme technology integration in the sub-micron regime comes with a rapid rise in heat dissipation and power density for modern processors. Dynamic voltage scaling is a widely used technique to tackle this problem when high performance is not the main concern. However, the minimum achievable supply voltage for the processor is often bounded by the large on-chip caches since SRAM cells fail at a significantly faster rate than logic cells when reducing supply voltage. This is mainly due to the higher susceptibility of the SRAM structures to process-induced parameter variations. In this work, we propose a highly flexible fault-tolerant cache design, Archipelago, that by reconfiguring its internal organization can efficiently tolerate the large number of SRAM failures that arise when operating in the near-threshold region. Archipelago partitions the cache to multiple autonomous *islands* with various sizes which can operate correctly without borrowing redundancy from each other. Our configuration algorithm – an adapted version of minimum clique covering – exploits the high degree of flexibility in the Archipelago architecture to reduce the granularity of redundancy replacement and minimize the amount of space lost in the cache when operating in near-threshold region. Using our approach, the operational voltage of a processor can be reduced to $375mV$, which translates to 79% dynamic and 51% leakage power savings (in $90nm$) for a microprocessor similar to the Alpha 21364. These power savings come with a 4.6% performance drop-off when operating in low power mode and 2% area overhead for the microprocessor.

1. INTRODUCTION

With aggressive silicon integration and clock frequency increase, power consumption and heat dissipation have become key challenges in the design of high performance processors. Growing power consumption reduces device lifetimes and expedites early stage failures [38]. It also affects the cost of thermal packaging, cooling, electricity, and data center air conditioning [25]. Dynamic voltage scaling (DVS) is a widely used technique to reduce the power consumption of microprocessors, exploiting the fact that dynamic power quadratically scales with voltage and linearly with frequency. However, the supply voltage of a microprocessor cannot be reduced below a certain threshold without drastically sacrificing clock frequency. Lowering this minimum achievable voltage can dramatically improve the lifetime, energy consumption, and battery life of medical devices, laptops, and handheld products.

The minimum achievable voltage for DVS is set such that under the worst-case process variation, the processor operates correctly [11]. The motivation for our work comes from the observation that large SRAM structures are limiting the extent to which operational voltages can be reduced in modern processors. This is because SRAM delay increases at a higher rate than CMOS logic delay as the supply voltage is decreased [39]. Furthermore, with increasing systematic and random process variation in deep submicron technologies, the failure rate of SRAM structures rapidly increases in the near-threshold regime. Ultimately, the minimum sustainable V_{dd} of the entire cache structure – and consequently the core as a whole – is determined by the one SRAM bit-cell within the entire system with the highest required operational voltage. This forces designers to appropriate a large voltage margin in order to avoid on-chip cache failures.

An SRAM cell can fail due to the following reasons: a read stability failure, a write stability failure, an access time failure, or a hold failure [3]. Figure 1 depicts the bit error rate (BER) of an SRAM cell based on the operational voltage in a $90nm$ technology node [30]. The minimum operational voltage of 64KB L1 and 2MB L2 caches is selected to ensure a high expected yield, 99% in

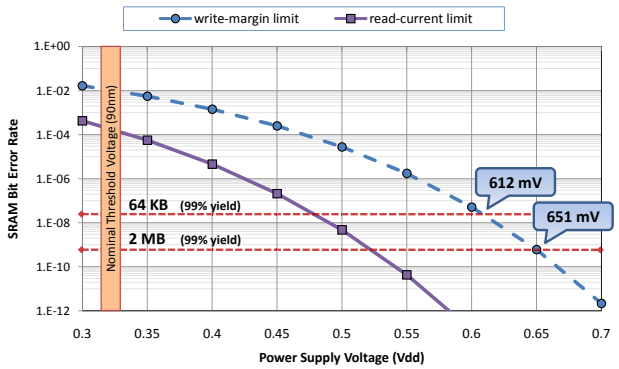


Figure 1: Bit error rate for an SRAM cell with varying V_{dd} values in $90nm$. For this technology, the write-margin is the dominant factor and limits the operational voltage of the SRAM structure. Here, the Y-axis is logarithmic, highlighting the extremely fast growth in failure rate with decreasing V_{dd}. The two horizontal dotted-lines mark the failure rates at which the mentioned SRAM structures (64KB and 2MB) can operate with at least 99% manufacturing yield.

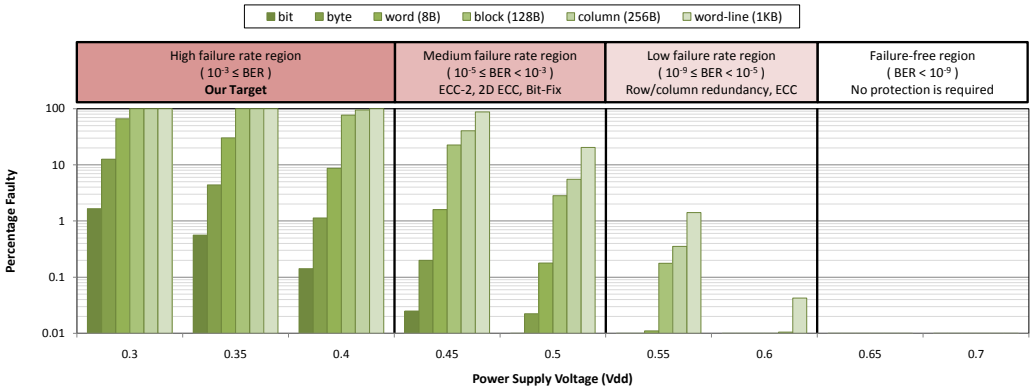


Figure 2: Percentage of faulty bits, bytes, words, blocks, columns, and word-lines for a 2MB L2 cache while varying the supply voltage. Here, the Y-axis is logarithmic, highlighting the rapid growth in faulty units when decreasing V_{dd}. The top part of this figure depicts our conceptual division of this V_{dd} range into four different regions based on the protection difficulty. For each region, corresponding bit error rates and also several applicable protection techniques are also shown. In order to operate correctly in the failure-free region, no protection mechanism is required. However, as can be seen, our target is the high failure rate region which causes an avalanche of failures for on-chip caches.

Figure 1. Due to the higher sensitivity of a bit-cell to parametric variations at lower $V_{dd}s$, the failure rate of an SRAM cell increases exponentially when V_{dd} gets decreased. This exponential increase in the number of faulty cells makes the protection of the on-chip caches much more difficult when operating in the near-threshold region. As can be seen in this figure, the write margin mostly dictates the minimum V_{dd} and it is expected to operate with $V_{dd} \geq 651mV$ due to the dominating failure rate of the 2MB L2 cache. This minimum operational voltage is consistent with predicted and measured values ($\sim 0.7V$) reported in [8].

In the literature, several techniques have been proposed to improve dynamic and/or leakage power of on-chip caches as well as the entire processor [39]. Section 4.1 summarizes some of these low-power design techniques. Most of these methods exploit the structure specific sleep modes and/or power-aware resource allocation to avoid facing with the failures. Consequently, for low V_{dd} values (e.g., $\leq 651mV$), the amount of power saving for these methods is restricted due to the arising failures in the SRAM structures.

In contrast, the objective of our work is to enable DVS to push the core/processor operating voltage down to the near-threshold region (e.g., low power mode) while preserving correct functionality of on-chip caches. An alternative to this approach is dual-V_{dd}s where core and caches operate at different voltages. However, dual-V_{dd} imposes a high cost in terms of area and design complexity. Voltage-level converters must be added to allow signal sharing between different voltage islands. Furthermore, high voltage memory elements generate noise that victimizes the neighboring low voltage logic circuits, necessitating either shielding or extra noise margins [40].

In this work, we target ultra-low voltage operation ($V_{dd} \leq 400mV$) in the near-threshold region which causes an extreme bit-cell failure rate ($> 10^{-3}$) for the on-chip caches – presented as the high failure rate region in Figure 2. This figure shows the percentage of *faulty* (i.e., containing at least one faulty bit-cell) bits, bytes, words, blocks, columns, and word-lines for a 2MB L2 cache for different V_{dd} values. Figure 2 is generated assuming a uniform failure distribution based on the relation between bit-cell failure rate and V_{dd} in Figure 1. In this figure, at $V_{dd} = 350mV$, almost all blocks are faulty while 30% of the words are faulty for the 2MB L2 cache. As can be seen and also discussed earlier for $V_{dd} \geq 651mV$, almost no failure is expected (i.e., *failure-free region*). As can be seen, for the $V_{dd} \leq 400mV$, finer granularities of redundancy are required to allow a high utilization of the spare elements since a large fraction of word-lines, columns, blocks, and even words would be faulty. This increases the complexity of the design since simple disabling (e.g., block or way disabling) or coarse grain redundancy (e.g., row or column redundancy) techniques cannot be efficiently applied.

In this work, we propose Archipelago (*AP*), a cache capable of reconfiguring its internal organization to efficiently tolerate the large number of SRAM failures that arise when operating in the near-threshold region. AP allows fault-free operation by partitioning the cache into multiple autonomous *islands* with various sizes. Each island is a group of physical cache word-lines that can operate correctly without using any word-line outside of their group. Each group has a sacrificial word-line which is divided up to multiple redundancy units. These spare units are *directly/indirectly* employed to achieve fault-free operation of the other word-lines in the same group. Sacrificial word-lines do not contribute to the effective cache capacity since they do not store any independent data. The clustering of the cache to these autonomous islands is done by a configuration algorithm which is described in Section 2.3. This adapted version of the minimum clique covering algorithm tries to partition the cache to the least number of islands/*groups* to minimize the number of sacrificial word-lines required for guaranteeing the fault-free operation of the cache. AP enables greater power savings than prior approaches and requires only a single power supply. The overhead of the approach is a small performance penalty (4.6%) when operating in near-threshold mode and a negligible area overhead (2%) for the microprocessor. We apply AP to L1-D, L1-I, and L2 caches to evaluate the achievable power reduction for the overall microprocessor system. A thorough comparison of AP with several well-known proposals is done in Section 4.3.

The primary contributions of this paper are: **1)** A flexible and highly reconfigurable architecture that can be leveraged to protect regular SRAM structures against high failure rates; **2)** In order to minimize the amount of redundancy required for protecting the cache, we model the cache fault pattern with a proper graph structure to guide a minimum clique covering configuration algorithm for near optimal group formation; **3)** To our knowledge, Archipelago is the first low overhead, fault-tolerant architectural technique which

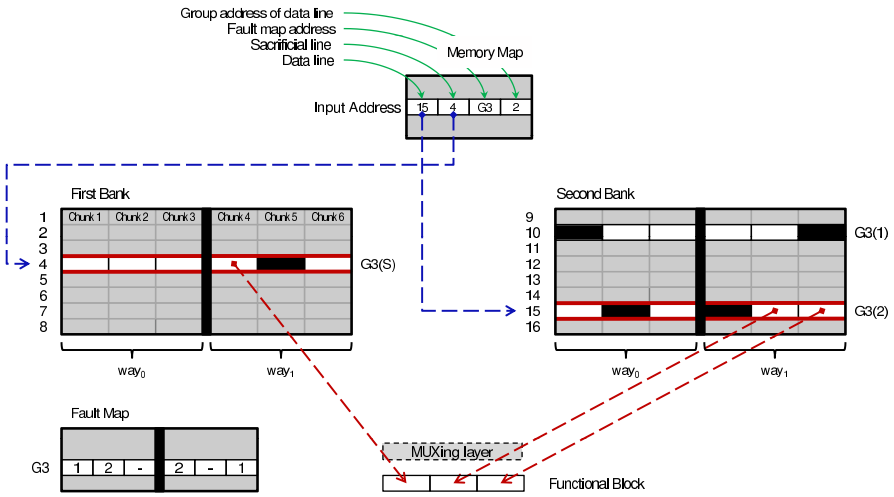


Figure 3: A dual-bank 2-way set-associative Archipelago. Each cache bank consists of eight lines. Cache blocks are divided into three equally sized data chunks. Black boxes represent chunks of data that have at least one faulty bit. The memory and fault maps, which are essential components of the proposed scheme, are also shown.

allows the cache to operate correctly when $V_{dd} \leq 400mV$; and **4)** A design space exploration in $90nm$ to show the actual process of selecting the architecture parameters for both L1 and L2 caches.

2. ARCHIPELAGO

In this section, we first describe the AP architecture that adaptively reconfigures itself to absorb failing SRAMs. Next, we present a configuration algorithm that exploits the intrinsic flexibility of the AP architecture to perform a near optimal redundancy assignment. The coalescence of highly flexible hardware and intelligent configuration enables our proposed solution to minimize the impact of operating at near-threshold region on cache characteristics (e.g., size and latency) with minimal overhead.

2.1 Baseline AP Architecture

In AP, the cache is partitioned into several autonomous islands of varying sizes. Each island is a group of physical cache word-lines that can operate correctly without using any word-line outside of their group. Each group has a sacrificial word-line which is divided up into multiple redundancy units. These spare units are *directly/indirectly* employed to allow a fault-free operation of the other word-lines in the same group. Put simply, considering one of these groups with *n + 1* partially functional cache word-lines, AP allows this group to behave as a set of *n* fully functional cache word-lines. In the remainder of this paper, we refer to every physical cache *word-line*, which may contain multiple cache blocks as a *line*.

Figure 3 is a toy example that depicts a 2-way set-associative AP cache with two banks. Each cache bank has eight lines and each line consists of two blocks of data which are divided into 3 equally sized data chunks. In this figure, lines 4, 10, and 15 form a group (named *G3*) in the cache. Line 4 (labeled G3(S)) is the sacrificial line that furnishes the redundancy needed to accommodate the faulty chunks in lines 10 and 15. In order to minimize the access latency overhead, the sacrificial line (4) and the data lines (10 or 15) should be in different banks[1] so that the sacrificial line can be accessed in parallel to the original data line.

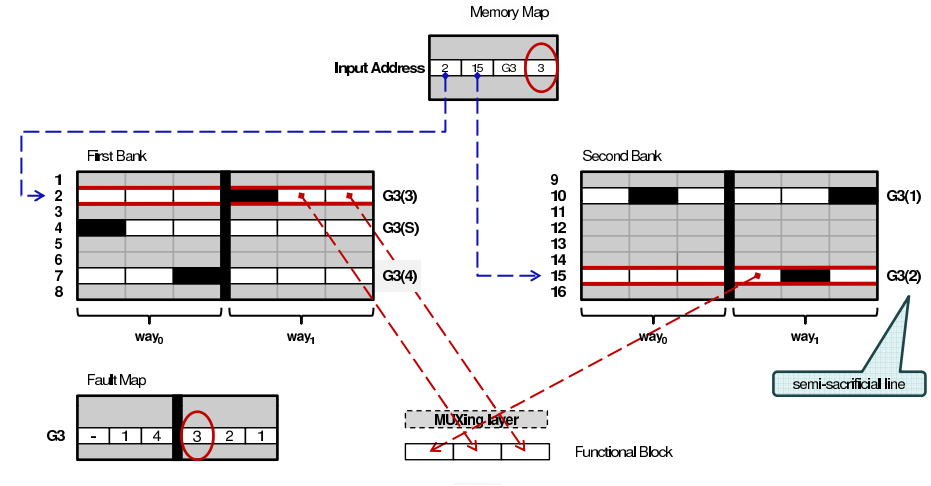
[1]In this work, our approach is described for protecting caches with only two banks. However, it should be clear that our scheme can be naturally extended to caches with any number of banks ≥ 2 without loss of generality.

The figure also shows structures specific to AP architecture (i.e., the memory map, the fault map and the MUXing layer). The cache access can be logically divided into three steps. First, the memory map is read to determine an index for accessing the cache banks. Next, the cache banks and fault map are accessed in parallel. Here, the cache banks provide the actual (from regular lines) and redundant data (from sacrificial line). The fault map indicates which parts of the actual or redundant data should be used to construct the output. Finally, the MUXing layer leverages the information from fault map to assemble the requested data.

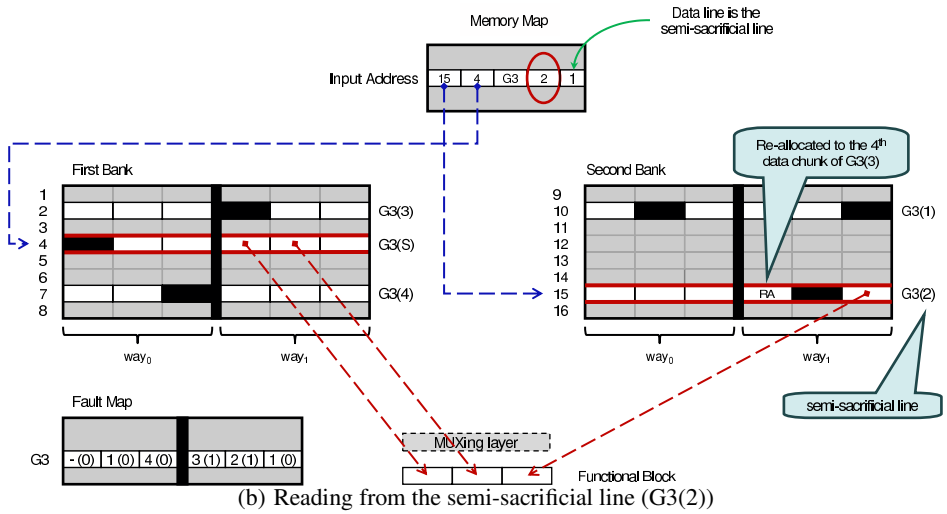
In AP, a memory map is used to provide a level of indirection in cache accesses. Each cache access first indexes into this memory map, which supplies the location of the data line and its corresponding sacrificial line. After these two lines have been accessed from their respective banks (different ones), a MUXing layer[2] is used to compose a fault-free block by selecting the appropriate chunks from each line. Considering the basic design of a *fast-access* cache (e.g., L1 cache), based on the tag comparison results, column decoders – which are placed before our MUXing layer – MUX the corresponding blocks out of read word-lines. On the other hand, it should be clear that for *energy-efficient* caches (e.g., L2 cache) design, each word-line might only contain a single block and access to the tag and data array is sequential. As a result, indexing into the memory map will be done after resolving the tag part of the address.

The MUXing layer receives its inputs indirectly from the *fault map*. For a given data line, the fault map determines which chunks are faulty and should be replaced with corresponding chunks from the sacrificial line. To aid in the encoding and decoding of this information, a unique address is assigned to all lines within a group (*group address*). For instance, in Figure 3, lines 10 and 15 are the first and second lines of *G3*, respectively. For each data chunk in the sacrificial line, the fault map stores the group address of the line to which that data chunk is assigned. In general, for a given group, if the sacrificial line consists of k data chunks, the fault map requires to store $(i_1, i_2, ..., i_k)$. In this notation, i_j is the group address of the line to which the jth data chunk of the sacrificial line is assigned. In our example, the entry which is assigned to the third

[2]As a side note, since read and write are symmetric operations, the only modification in the hardware implementation would be to replace the MUXes in the MUXing layer with pass transistors [40].



(a) Reading a line (G3(3)) from the same bank as the sacrificial line (G3(S))



(b) Reading from the semi-sacrificial line (G3(2))

Figure 4: Two special read-access scenarios. A standard read access is illustrated in Figure 3. Notice the extra bit that has been added to both the memory map and every fault map entry to handle scenario (b). Since the 4*th* data chunk of the semi-sacrificial line is re-allocated, it is marked as RA in scenario (b).

group (*G3*) in the fault map, contains (1 | 2 | - | 2 | - | 1) for way_0 and way_1. Considering only way_1, this indicates that the 4*th* chunk of the corresponding sacrificial line (*G3(S)*) is devoted to *G3(2)*, the 6*th* chunk is dedicated to *G3(1)*, and the 5*th* chunk is not assigned to any line. Finally, the MUXing layer gets its input from a set of comparators that compare the group address of a data line with fault map entries for the same group. For example, group address of line 15 is 2 – read from memory map – which gets compared with *G3*'s fault map entries.

In order for lines in a cache to come together and form a group, they should not have any *collisions*. In our scheme, two lines have a collision if they have at least one faulty data chunk in the same position. For example, if the second data chunk of the 3*rd* and 6*th* lines is faulty, lines 3 and 6 have a collision. Similarly, in Figure 3, lines 10 and 15 are collision-free since they do not have a faulty block in the same position. By definition, collisions are independent of workload running on the system and data stored in the cache. Two lines that have a collision cannot share a single sacrificial line. This means as the number of collisions increase, a larger fraction of cache needs to be turned into sacrificial lines. Group formation is the key component to minimize the number of sacrificial lines across cache. Therefore, the primary objective of AP is to form groups such that there are no collisions between any two lines within a group.

2.2 AP with Relaxed Group Formation

Since every group requires a sacrificial line be dedicated solely for redundancy, AP strives to minimize the total number of groups that must be formed. Given that the number of lines is fixed within a cache, achieving this objective implies that larger groups are preferred over smaller ones. In order to improve the likelihood of forming large groups, we remove the constraint that forces the sacrificial line, from a particular group, to be in a different bank than all the other lines. This allows any set of lines from the two banks to form a group. However, in order to minimize the access latency, we do not allow a group to derive all its lines from the same bank. In other words, each group should have at one line in each bank to allow a parallel access to the original and spare data. Relaxing the mentioned constraint, gives rise to two new read access scenarios in addition to the standard case described in Figure 3.

Handling Different Types of Accesses: In Figure 4(a), lines 2, 4, 7, 10 and 15 are in the same group, with line 4 serving as the

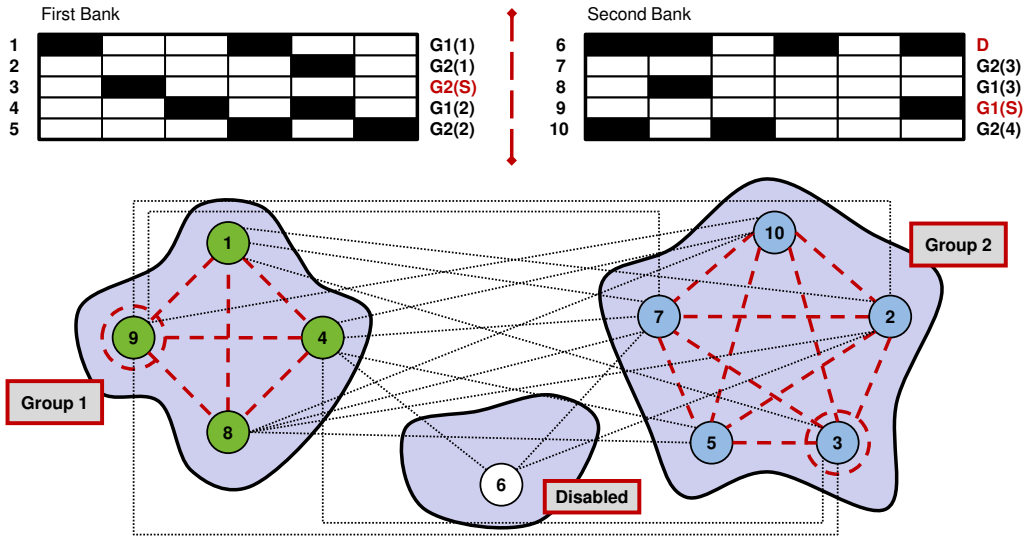


Figure 5: A simplified example of the minimum clique covering process for a given distribution of faults in the cache banks. Here, each bank has only 5 lines. The solver disables the 6*th* line since it has many faulty chunks and, is therefore very expensive to repair. Two cliques are formed by the solver and lines 9 and 3 are designated as sacrificial lines for groups 1 and 2, respectively. Moreover, the conceptual partitioning of the cache into distinct islands is also demonstrated.

sacrificial line. However, when line 2 or 7 is accessed as a data line, a parallel access to line 4 cannot be performed since they are in the same bank. To handle such a scenario, a small and transparent modification to the AP architecture is needed. We arbitrarily select a line from the bank not containing the sacrificial line (but still from the same group) and label it as a *semi-sacrificial* line (line 15 in Figure 4(a)). This semi-sacrificial line can be used to replace faulty chunks from cache lines which are in the same bank as the sacrificial line. However, *in contrast to the sacrificial line, the semi-sacrificial line still contributes to the effective cache space*. In other words, a semi-sacrificial line only acts as a level of indirection for redundancy substitution by borrowing redundancy from its corresponding sacrificial line. Moreover, semi-sacrificial lines guarantee the parallel access to the original and spare data in all possible scenarios. Consequently, the faulty chunks of the lines 2 and 7 are replaced using *G3(2)* instead of the *G3(S)*. With the addition of the semi-sacrificial line, accesses to the cache can be divided into three categories:

1) *Accesses to data and sacrificial lines that reside in different banks*. This is the base case which is demonstrated in Figure 3 and does not require any special consideration beyond what is described earlier.

2) *Accesses to data lines that are the in the same bank as the corresponding sacrificial line*. This case is illustrated in Figure 4(a). Line *G3(3)* is the data line and line *G3(2)* is the semi-sacrificial line which supplies the replacement chunks for the faulty ones in *G3(3)*. Instead of accessing *G3(S)*, the memory map remaps the address of the sacrificial line to the address of *G3(2)*. This case is particularly interesting, since no other parts of the procedure must be adjusted to support this access scenario. The fault map is still used, unmodified, to indicate that the 4*th* data chunk from *G3(3)* is faulty and should be substituted. Therefore, AP replaces this faulty chunk of *G3(3)* by the semi-sacrificial line's 4*th* chunk instead of sacrificial line's 4*th* chunk.

3) *Accesses to a semi-sacrificial line*. This case is demonstrated in Figure 4(b) for which two small modifications to the access procedure are necessary. An additional bit is added to memory map entries indicating whether the accessed data line is the semi-sacrificial line. As can be seen, the 4*th* data chunk of *G3(2)* has been re-allocated to *G3(3)*. Consequently, we artificially mark the 4*th* chunk of the semi-sacrificial line as "re-allocated" (**RA** in Figure 4(b)). When *G3(2)* is accessed, its faulty and also re-allocated data chunks (i.e., the 4*th* and 5*th* chunks) are supplied as expected by the corresponding chunks from the sacrificial line (*G3(S)*). However, this cannot be easily done using the unmodified fault-map since the 4*th* entry of the fault map points to the faulty chunks of data from *G3(3)*. As a result, the system cannot identify that the 4*th* chunk of the semi-sacrificial line should be replaced during an access. In order to tackle this problem, we add an extra bit to every fault map entry which indicates whether the corresponding data chunk should be replaced when accessing the semi-sacrificial line. For instance, in Figure 4(b), since the 4*th* and 5*th* chunks should be replaced, their corresponding bits in the fault map have been set to "1".

2.3 AP Configuration

To maximize the number of functional lines in the cache, we need to minimize the number of sacrificial lines required to enable fault-free operation. As previously discussed, there is a single sacrificial line devoted to every group of lines. This sacrificial line is not addressable as a data line since it does not store any independent data. In other words, sacrificial lines do not contribute to the usable capacity of the cache. Depending on the number of collision-free groups that are formed, the effective capacity of the cache can vary dramatically. This motivates the need to minimize the total number of groups required.

Problem Modeling: Here, we model the problem as a graph in which every group corresponds to a clique. To minimize the number of groups in a given faulty cache – upper part of Figure 5 with two banks, we model the problem as a minimum clique cover (*MCC*) problem [14]. Figure 5 shows the process of forming the groups given a fault pattern for the cache. Each node in the constructed graph, ten in all, is a cache line. There is an edge between two nodes if and only if there is no collision between the corresponding lines. Therefore, it is possible to assign connected nodes

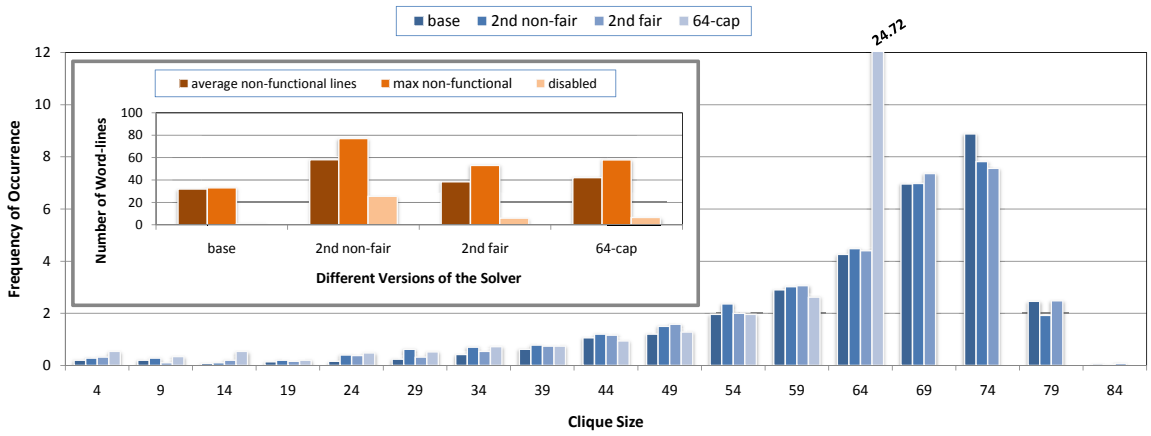


Figure 6: Distribution of the clique size for different versions of the solver based on Monte Carlo simulation. Note that for 64-cap, the size of all cliques is ≤ 64. Here, the number of non-functional lines is the summation of the number of sacrificial lines and the number of disabled lines. The plot in the insert depicts the average number of non-functional cache lines, the maximum number of non-functional lines, and the number of disabled lines while achieving 99% yield.

to the same group. For example, there is an edge between lines 2 and 3 but no edge between lines 1 and 10.

As mentioned earlier, a group is a set of lines for which there is no collision between any pair of lines. By constructing the graph this way, a collision free group forms a clique (i.e., there is an edge between every pair of nodes). As a result, the task of forming groups can be represented as finding the cliques in the constructed graph. However, since we are interested to minimize the number of sacrificial lines, this problem turns to finding the minimum number of cliques that cover the entire graph. In Figure 5, lines 1, 4, 8, and 9 form the first group (*G1*, clique with size 4) while lines 2, 3, 5, 7, and 10 form the second group (*G2*, clique with size 5). Line 6 is disabled (*D*) since it contains 4 faulty chunks and repairing it is not cost effective. In general, a line gets disabled, if its corresponding node in the graph does not belong to any clique with size ≥ 2. Here, the fault map has 2 lines which correspond to the sacrificial lines 9 and 3. As can be seen in this figure, there are 16 faulty chunks out of 60, therefore, at most $10 - (\lceil \frac{16}{6} \rceil) = 7$ cache lines can be kept functional after the grouping. Ultimately, our configuration algorithm can achieve this best case, highlighting the effectiveness of our approach.

MCC Solver: We use the transformation described in [19] to convert the MCC problem to a minimum chromatic number problem. After applying this transformation, the final graph is passed to the DSATUR solver [22] which uses a well-known and efficient approximation algorithm. As shown in [22], the approximation factor for the DSATUR solver is $\leq 6.1\%$ for various graph densities. For the problem size that we target in this work, the run time of the DSATUR algorithm is always less than 5*ms*. In contrast, the full backtrack-based solver (e.g., optimal solver) takes several days to solve *one instance* of our configuration problem which makes it infeasible to be used in practice.

Nevertheless, the original solver's answer is not directly applicable to our problem. Since the solver is free to form a group in any possible way that minimizes the number of cliques, it is possible to have a clique with nodes in only one bank. This latter case is not a feasible solution because the solution disallows parallel access to the spare elements. As a result, we apply a set of modifications to the original solver for making it suitable to our application:

1) We force the solver to pick the second line of the group from a different bank from the first line of the group. This small modification assures us that at least one line is selected from each bank and allows us to take care of the parallel access problem discussed above.

2) An artifact in the DSATUR solver algorithm can sometimes cause it to disable a large fraction of cache lines. More specifically, it picks the nodes for coloring only based on the degree of saturation which is proportional to the reciprocal of the node degree in our constructed graph [22]. Because of this bias, all the lines from one bank might be selected while there are many unassigned lines in the other bank. In such a scenario, the unassigned lines have to be disabled which results in a large fraction of disabled lines. In order to solve this problem, we modify the original DSATUR algorithm to pick the lines from both banks more evenly. This was done by giving artificial priority to the lines in a bank that has more *unassigned lines* at the beginning of each assignment phase.

3) As will be discussed in Section 3, minimizing the area overhead of the fault map carries a major significance for our scheme. The size of each entry in the fault map is proportional to the $\log_2(max\{clique\ size\})$. Therefore, to reduce the size of the fault map, an upper bound (e.g., 64) can be placed on the maximum clique size (*MCS*). By adding this feature, all the cliques which are found by the DSATUR solver can be forced to have a size smaller than or equal to *MCS*.

The main plot in Figure 6 depicts the distribution of clique sizes for different versions of the solver: base (base DSATUR solver), 2nd non-fair (base solver + modification (1)), 2nd fair (base solver + modifications (1,2)), and 64-cap (base solver + modifications (1,2,3)). This data is generated using 1000 iterations of a Monte Carlo simulation for a 2MB L2 cache with 2048 lines. It should be clear that the size of the fault map is proportional to the $(Num.\ of\ Cliques) \times \log_2(MCS)$ which implies a small number of large cliques is preferable. Furthermore, since $(Num.\ of\ Cliques) \times (Average\ Clique\ Size)$ is equal to the total number of word-lines, a constant value, *MCS* should be as close as possible to the $Average\ Clique\ Size$. As a result, the most desirable distribution of clique sizes would be a tight distribution around large clique sizes. As can be seen in Figure 6, a majority of cliques fall into the narrow region of 60 to 80 nodes. This tight distribution shows the efficiency and proper balancing of the group sizes in the process of group formation by the different versions of the solver. The smaller plot in this figure demonstrates different characteristics of these 4 versions of the solver. In this plot, the number of non-functional lines is the summation of the number of

sacrificial lines and the number of disabled lines. As can be seen, the second modification (i.e., fairness) can effectively reduce the average number of disabled lines from 25.5 to 6.1.

Another observation is that the constraint $MCS = 64$ increases the maximum number of non-functional lines by 9% while it reduces size of each fault map entry by 17% – from 7 bits to 6 bits. For each cache instance, the number of lines in the fault map array is equal to the number of sacrificial lines (e.g., 2 in Figure 5). However, due to the presence of process variation in a large population of fabricated chips, different fault patterns should be expected. As we described earlier, in our evaluation, we employ a Monte Carlo simulation to generate a population of 1000 cache instances and the total number of fault map lines is determined based on the maximum number of sacrificial lines while achieving a 99% yield.

Hardware Configuration: In order to configure AP, the memory and fault maps need to be filled. The initial step involves solving the MCC configuration problem for a given cache fault-pattern. Given the MCC solution, each line – a node in the graph – can be classified as: **1)** *Data line:* For each data line, a new memory map entry should be allocated. Each memory map entry has 5 fields (Figure 4(b)): The first field is the data line address. If this line is in the same bank as its respective sacrificial line, the second field will set to the address of the respective semi-sacrificial line. Otherwise, it will get the address of the sacrificial line. The third, fourth, and fifth fields should be set to the data line's group number, group address, and "0", respectively. **2)** *Sacrificial line:* Although no change in the memory map is required, a fault map entry should be allocated for each sacrificial line. Each fault map entry has a field per data chunk – 6 fields in Figure 4. For faulty data chunks of a sacrificial line or the ones which are not assigned to any faulty data chunks, corresponding fields in the fault map should be set to "-". For other data chunks, corresponding fault map fields should be set to the group addresses of the data/semi-sacrificial lines to which those data chunks are assigned. **3)** *Semi-sacrificial line:* This is similar to the first case, except the last field of the memory map entry should be set to "1". **4)** *Disabled line:* Nothing needs to be done in this case.

Low Power Mode Operation: As we saw in this section, Archipelago leverages a simple architecture which mainly consists of two relatively small array structures. However, in order to achieve a robust sub-400mV operation, a sophisticated configuration algorithm is used to customize the hardware based on the existing fault pattern in a particular cache. This hardware/software co-design allows to keep the architecture simple when modeling the configuration with well-established algorithmic problems. The first time a processor switches to low power mode, the built-in self test (BIST) module scans the cache for the potential faulty cells. After determining the faulty chunks of each cache line in low power mode, the processor switches back to high power mode and constructs the mentioned graph and solves the MCC problem using the modified DSATUR solver. As mentioned before, the solver time for a 2MB L2 cache is less than 5*ms* on an Intel Core™2 Duo machine. This solution contains the information that is required to be stored in the memory and the fault maps. This configuration information can be stored on the hard-drive and is written to the memory map and fault map at the next system boot-up. In addition, the memory map, fault map and the tag arrays need to be protected using, for example, the well studied 10T cell [8] which has about 66% area overhead for these relatively small arrays. These 10T cells are able to meet the target voltage in this work for the aforementioned memory structures without failing. However, as we will discuss in Section 4.3, 10T cells are not a cost-effective option for protecting the entire L1 and L2 caches.

Table 1: The target system configuration

Parameters	Value
Technology	90 nm (1.2 V nominal V_{dd})
Clock frequency	1.9 GHz [23]
L1 Cache	2 banks 64KB data, 2 banks 64KB instruction, split, 2-way, 4 cycles, 1 port, 64B block size
L2 Cache	2 banks 2MB Unified, 8-way, 10 cycles latency, 1 port, LRU, 128B block size
Registers	80 integer, 72 floating point
ROB (re-ordering buffer)	128 entries
LSQ (load/store queue)	64 entries
Instruction fetch buffer	32 instructions
Integer/FP issue queue	32/32 entries
FU (functional unit)	4 int ALU, 4 int mult/div, 2 memory system ports
FPU (floating point unit)	4 FP ALU, 1 FP mult/div
Main memory	225 cycles (high power), 34 cycles (low power)
Branch predictor	combined (bimodal and 2-level), RAS (return address stack) has 32 entries
BTB (branch target buffer)	2048 entries, 2-way associative, BHT (branch history table) has 4096 entries

Cache Addressing Mechanism after Capacity Reduction: In our design, the memory map can be used to remap the *original address* of the sacrificial lines to other functional lines. As a result, a small fraction of the sets will have the functionality of two sets. In other words, two sets different sets will be located in the same line. For those dual-set word-lines, associativity is reduced by half. These dual-set lines are distributed evenly across the cache to prevent biased miss rate for an address sub-range. The tag comparison and replacement logic needs straight-forward modifications to make this work, details of which are omitted in the interest of space.

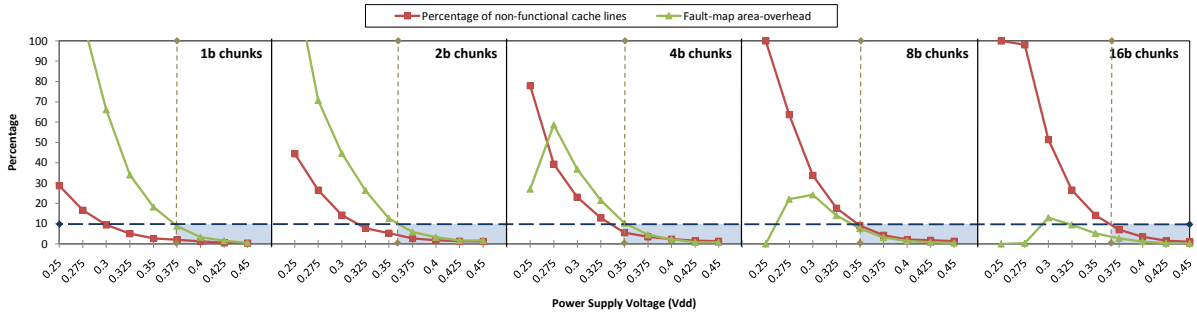
High Power Mode Operation: In the high power mode, our scheme is turned off in order to minimize the unwanted overheads: **1)** All the cache lines are functional and there is no sacrifice of the cache capacity. **2)** Assuming clock gating, there is a negligible overhead for the dynamic power due to the switching in the bypass MUXes which consists of the MUXing layer and an additional MUX for bypassing the memory map. In other words, in high power mode, the access path of AP includes an additional bypass MUX compared to a baseline cache. **3)** Leakage power overhead remains the same. However, power gating techniques can be used for general leakage mitigation [18].

3. EVALUATION

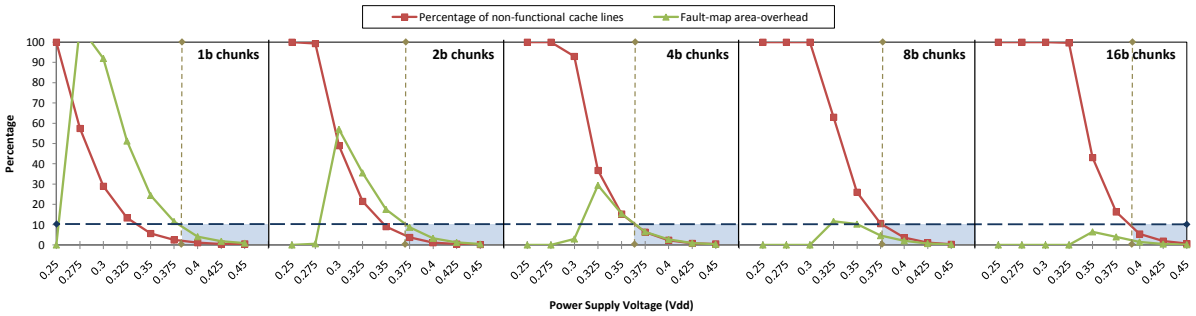
This section evaluates the potential of AP in reducing the power of the processor while keeping the overheads as low as possible. Comparisons with related work are presented in the next section of the paper.

3.1 Methodology

For performance evaluation, we use SimAlpha, a validated micro-architectural simulator based on the SimpleScalar out-of-order simulator [5]. The processor is configured as shown in Table 1 and is modeled after the DEC Alpha 21364 at an ambient temperature of $40^{\circ}C$ [35, 23]. Dynamic processor power consumption is calculated using Wattch [7] based on the activity factors of individual core structures, and leakage power is computed with HotLeakage [41]. CACTI [31] is leveraged to evaluate the delay, power, and area of the base *cache* structures. To take into account the overheads of the memory map and fault map arrays, we use the SRAM generator module provided by the 90nm Artisan Memory Compiler. The Synopsys standard industrial tool-chain (with a TSMC 90nm technology library) is used to evaluate the overheads of the remaining miscellaneous logic (i.e, bypass MUXes, comparators, subset tag comparison disabling, etc.). Lastly, to find the matching



(a) Percentage of non-functional lines and area overhead of the fault-map for **L1** while varying V_{dd} and data chunk size



(b) Percentage of non-functional lines and area overhead of the fault-map for **L2** while varying V_{dd} and data chunk size

Figure 7: Process of determining the minimum achievable V_{dd} for L1 and L2 caches while limiting the fraction of the non-functional cache lines and also the area overhead of the fault map structure to $\leq 10\%$. Moreover, in these 10 sub-plots, vertical dotted lines show the minimum achievable V_{dd} while data chunk size varies from 1 *bit* to 16 *bits*.

frequency for a given V_{dd}, we use the alpha-power model described in [27].

For a given set of cache parameters (e.g., V_{dd}, chunk size, MCS, etc.), a Monte Carlo simulation with 1000 iterations is performed using the modified DSATUR solver described in Section 2 to identify the portion of the cache that should be disabled. As discussed earlier, solutions generated by the MCC solver target a 99% yield. In other words, only 1% of manufactured and configured on-chip caches are allowed to exhibit failures when operating in low-power mode. On the other hand, the target yield directly impacts the size of the fault map. Its size is set based on the maximum number of cliques formed across all cache instances in the Monte Carlo process when ignoring as many worst-case situations as the target yield allows.

3.2 Design Space Exploration

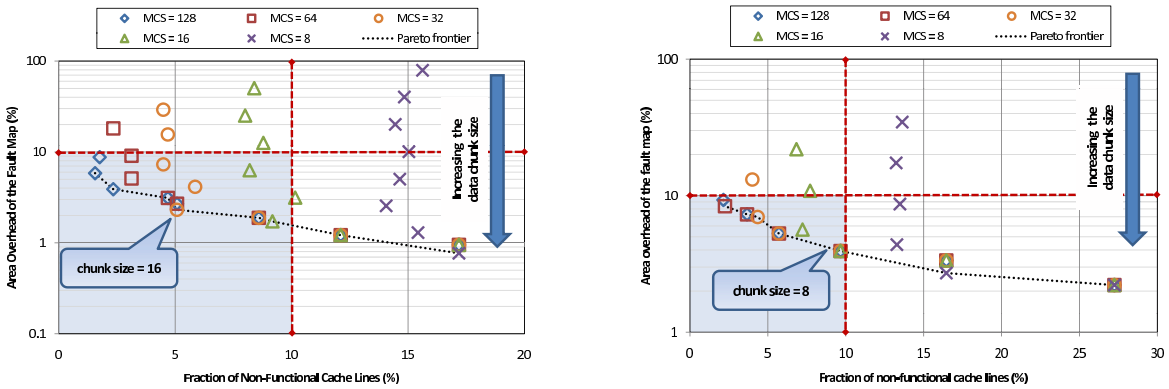
Figure 7 shows the process of determining the minimum achievable V_{dd} for our target system. In this figure, chunk size varies from 1 *bit* to 16 *bits* for both L1 and L2 caches. In high-power mode, both fault and memory map arrays remain idle and leak power. It is crucial to minimize the size of these structures. The size of the memory map is essentially fixed by the number of lines in the cache. The fault map size, however, can vary significantly depending on configuration parameters, motivating a closer look at the size of the fault map as an important design factor. Consequently, we limit the area overhead of the fault map to 10% (of the cache area). Furthermore, since cache size has a strong correlation with system performance, we limit our scheme to setting at most 10% of the cache lines to *non-functional*.

As evident in Figure 7, decreasing V_{dd} increases the non-functional portion of the cache and also the area of the fault map array. However, beyond a certain point, the area overhead of the fault map starts decreasing. This phenomena is due to the large fraction of the cache lines that get *disabled* as lowering V_{dd} leads to increasing error rates and a precipitous increase in faulty chunks. As we mentioned earlier, for these disabled lines, no entry in the fault map array is required.

Here, the vertical dotted lines highlight the minimum achievable V_{dd} based on the aforementioned 10% limits on non-functional lines and fault map size. It is notable that for small chunk sizes, area overhead of the fault map is the limiting factor, while for larger chunks, the number of non-functional lines becomes dominant. As can be seen in this figure, the minimum achievable V_{dd} for L1 is lower than L2. This was expected due to L2's longer lines and larger size which makes protection of L2 harder than L1 [40]. Therefore, L2 protection cost dictates the minimum operating voltage of our system. In addition, based on the trend in Figure 7, it should be clear that a chunk size outside of the presented range will only result in a higher minimum V_{dd}. As a result, we select $375mV$ as the minimum V_{dd} (i.e, low-power mode operating voltage) since all other lower voltages violate our 10% limits.

For $V_{dd} = 375mV$, a design space exploration of L1-(D/I) and L2 caches is demonstrated in Figure 8. There are two important parameters to note: **1)** MCS, the maximum allowable clique size (Section 2.3) and **2)** chunk size, varying from $1b$ to $128b$ for L1 and from $1b$ to $32b$ for L2. From Figure 8, it should be clear that the chunk sizes outside of the presented range always violate at least one of our aforementioned 10% limits. For every pair of (MCS, chunk size), a Monte Carlo simulation, targeting 99% yield, is performed with 1000 iterations. This simulation identifies the necessary area overhead of the fault map and the fraction of non-functional lines in the cache. Here, we still limit the fraction of the non-functional lines to 10% while trying to minimize the area overhead of the fault map.

In Figure 8(a), within the shaded region, only points on the black dotted line (Pareto frontier) are considered since they dominate the



(a) **L1** design space exploration

(b) **L2** design space exploration

Figure 8: Design points for different Maximum Clique Size (MCS) and chunk size pairs are shown that can achieve a 99% yield. For each MCS value, corresponding chunk sizes from $\{2^n \mid n \in \{0, 1, ..., 7\}\}$ for L1 and from $\{2^n \mid n \in \{0, 1, ..., 5\}\}$ for L2 are chosen. The shaded boxes represents the region of interest where both the fault-map overhead and the fraction of non-functional lines is limited to $\leq 10\%$. The black dotted line is the Pareto frontier.

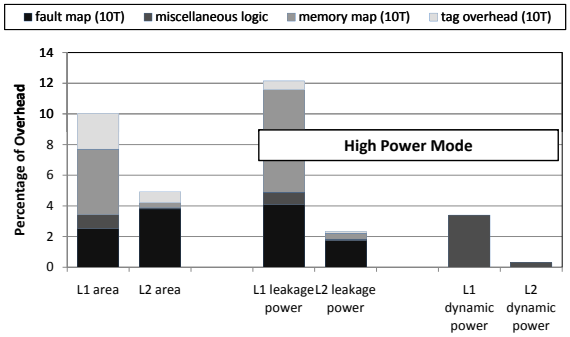
other design points. Note that making the chunk size larger decreases the area overhead of the fault map since the number of entries in the fault map is reduced. However, this reduction is also accompanied by an increase in the fraction of non-functional cache lines, the result of fewer edges in the graph described in Section 2.3. For L1, the design point with $MCS = 32$ and $chunk\ size = 16$ bits is selected, highlighting an interesting trade-off between the area overhead of the fault map and the fraction of non-functional lines. By repeating the same process for L2, as illustrated in Figure 8(b), the design point with $MCS = 16$ and $chunk\ size = 8$ bits is selected.

3.3 Results

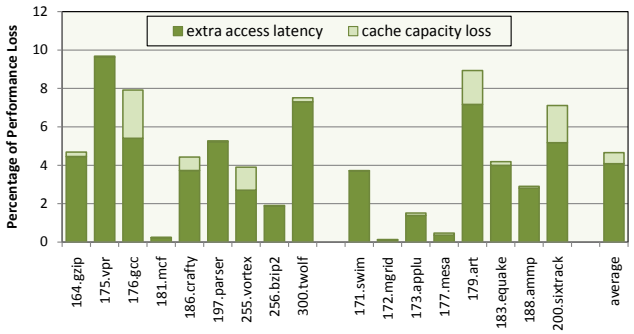
Figure 9(a) summarizes the overheads of our scheme for both L1 and L2 caches. As mentioned before, we also account for the overheads of using 10T SRAM cells [8] for protecting the tag, fault map, and memory map arrays in low-power mode. In addition, the fault map, memory map, and the second bank have their own separate decoders which are accounted for in our evaluation. Leakage overhead in high power mode corresponds to the fault map, memory map, miscellaneous logic, and extra leakage of 10T cells for tag array. Note, the memory map is a far greater contributor to area and leakage power overhead in the L1 than in the L2. The reason behind this is that the L1 has only $\frac{1}{4}$ the lines of the L2, while its overall size is $\frac{1}{32}$. For L2, the fault map is the major component of overhead. Due to its significantly larger size, the L2 cache dominates the processor *leakage* and *area* overheads. Nevertheless, as can be seen in this figure, our scheme has only modest overheads for the L2 cache. Dynamic power overhead in high-power mode can be mainly attributed to bypass MUXes since we assume clock gating for the fault map and memory map arrays. In AP, when in low-power mode, the memory map and MUXing layer are on the critical path of the cache access. Based on our timing analysis, the access delay overhead of L1 and L2 are $0.41ns$ and $0.58ns$, respectively. Based on our system clock frequency (Table 1), this translates to 1 additional cycle latency for L1 and 2 additional cycles for L2 in low-power mode.

As mentioned before, the fraction of non-functional lines is based on a 99% yield in a 1000-run for a Monte Carlo simulation. As a result, the fraction of non-functional lines would be smaller than what is presented in Figure 8 for many chips. This is because across all the fabricated dies, process induced parametric variation causes different cache fault patterns to appear. However, we consider the worst-case capacity loss during our evaluation. In order to evaluate the *worst-case* performance penalty of our scheme in low-power mode, we ran the cross-compiled Alpha binaries of SPEC-CPU-2K benchmark suite on SimAlpha after fast-forwarding to an early SimPoint [37]. We assume one extra cycle latency for L1 and 2 extra cycles for L2. Cache size is also reduced based on the fraction of non-functional lines in Figure 8. On average, a 4.6% performance penalty is seen in low-power mode from which 0.6% is contributed by the cache capacity loss due to the presence of non-functional lines (Figure 9(b)). As can be seen, our strict limit on the fraction of non-functional lines results in minimal impact on performance because of cache capacity loss. However, one should note that low-power mode performance is usually not a major concern. In high power mode, there is no capacity loss since no failures need to be tolerated. Furthermore, based on our CACTI delay results and the frequency of the system (Table 1), there is enough slack on the access time of our L1 and L2 caches to fit the small bypass MUXes (additional $0.07ns$ delay) without adding any extra cycles to the access time of these caches. In other words, there is no performance loss in high-power mode. However, one might have a cache design without any slack available. In that scenario, we add an additional cycle in high-power mode for L1 and L2, which translates into a 3.6% performance loss for SPEC-2K.

Summary of benefits and overheads: Figure 10 shows the savings and overheads for the Alpha 21364 microprocessor using AP for protecting the on-chip caches. As can be seen in Figure 10(b), the overheads of the proposed method are almost negligible. These overheads are evaluated in $90nm$ using the methodology described in Section 3.1. On the other hand, Figure 10(a) depicts the percentage reduction in leakage power, dynamic power, and minimum achievable supply voltage by using AP for protecting the on-chip caches. These results are reported in the $45nm$, $65nm$, $90nm$, and $130nm$ technology nodes. The relation between the supply voltage and the expected SRAM bit-cell failure rate for these four technology nodes are extracted from [40, 24, 34, 13, 33, 9, 6]. Considering the $90nm$ technology node, AP enables DVS to save 79% dynamic power and 51% static power in the near-threshold region. With the aggressive technology scaling, the systematic and random variations are expected to increase [33]. This results in higher sensitivity/vulnerability of SRAM cells to power supply variations. Hence, the percentage of reduction in dynamic power/minimum achievable V_{dd} gradually reduces when heading toward deeper submicron technologies. Nevertheless, a 68% dynamic power reduc-

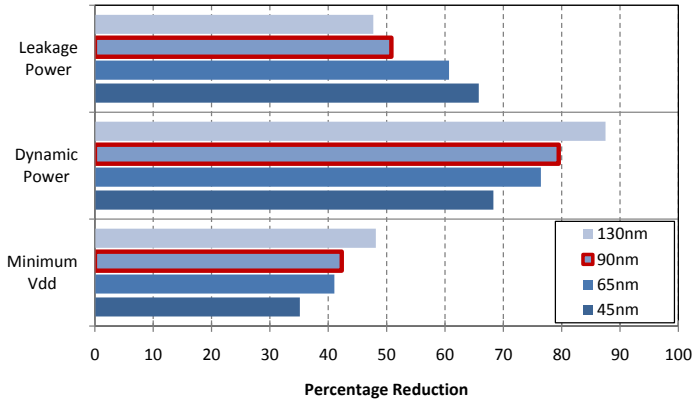


(a) Area, leakage, and dynamic power overheads of our scheme for both L1 and L2 caches. Here, 10T cells are used for protecting the fault map, memory map, and tag arrays.

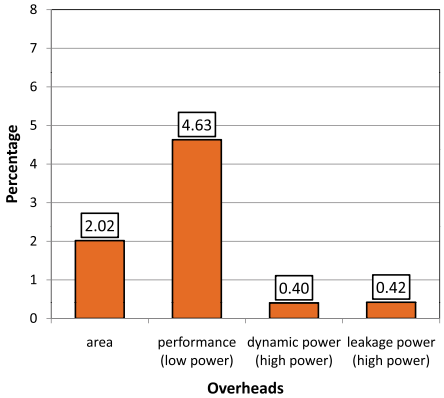


(b) Performance loss in low-power mode. Since the fraction of non-functional lines is limited to less than 10%, the access latency overhead is the dominant factor in performance penalty.

Figure 9: Area, power, and performance overheads of our scheme.



(a) Percentage of reduction in leakage power, dynamic power, and minimum achievable V_{dd} for the baseline processor using 4 different technology nodes.



(b) Overheads of using AP for protecting on-chip caches of the mentioned microprocessor in $90nm$.

Figure 10: Low-power mode benefits and overheads for an Alpha 21364 microprocessor system (Table 1) augmented with Archipelago. Here, we account for the dynamic power overhead of accessing the second bank in low power mode for handling failures.

tion can be achieved in $45nm$. In contrast, for leakage power, the percentage reduction gradually increases as the technology scales down. This is mainly due to the drastic differences in correlations between the leakage power and V_{dd} across technology generations. In deeper technology nodes, a reduction in V_{dd} manifests as a much larger saving in leakage power [15]. Therefore, for deeper technologies, even though we achieve less V_{dd} reduction, the net leakage power saving is larger. Due to the excessive increase in the sub-threshold transistor leakage, it is expected that leakage power will dominate the total power consumption of a chip in the future technologies [15].

4. RELATED WORK

Motivations for the AP design come from previous work in two major categories: low-power and fault-tolerant cache techniques. Here, SRAM cell stability efforts are included in the first category since they try to proactively avoid failures. Furthermore, a comparison of AP with several of these conventional and state of the art proposals will be presented.

4.1 Low-Power Cache Techniques

The usage of V_{dd} gating for leakage power reduction by turning off cache lines is described in [20]. This approach reduces the leakage power of the cache by turning off the cache lines that are not likely to be accessed in the near future. Meng et. al. [28] proposed a method for minimizing leakage overhead in the presence of manufacturing variations. In this scheme, they artificially prioritize cache ways with smaller leakage and resize the cache by avoiding sub-arrays that have higher leakage factors. Instead of turning off blocks, drowsy cache [12] is a state preserving approach that has two different supply voltage modes. In order to save power, recently inactive cache blocks periodically fall into a low power mode in which they cannot be read or written.

For low V_{dd} values (e.g., $\leq 651mV$ in $90nm$), the amount of power saving for schemes similar to drowsy cache [12] is restricted due to failures in SRAM structures [34]. As discussed earlier, our objective is to enable DVS to push the processor/core operating voltage down to the near-threshold region while preserving correct functionality of on-chip caches. Wilkerson et. al. [40] proposed two different cache protection schemes for L1 and L2 caches that use several levels of decoding/shifting to take the faulty data chunks out and replace them using ECC protected patches. Due to the strict binding between data and redundancy, their schemes need to disable 50% of L1 and 25% of L2 caches which results into a considerable performance drop-off in low power mode. Recently, Abella et. al. [1] proposed a cache protection scheme based on sub-block disabling which can provide a better performance predictability than [40]. However, since this scheme relies on dis-

abling finer granularities than a cache block, it losses its efficiency when applied to caches other than L1-Data. Chishti et. al. recently proposed another technique [10] that employs multi-bit segmented ECC to also allow soft and hard-error resilience in lower voltages by sacrificing 50% of cache capacity. Although an interesting approach, it can only achieve 30% reduction in the MinV_{dd}.

On the other hand, many variations of SRAM cells such as 8T [30], 10T [8], 11T [29], and ST [24] have also been proposed. These larger SRAM cells are more stable against different sources of parameter variations compared to the conventional 6T cell and allow the SRAM structures to operate at lower voltages while preserving its correct functionality. Most of these cells have a large area overhead which is a significant shortcoming since the extra area does not translate into any performance gains when operating in high power mode.

4.2 Fault-Tolerant Cache Techniques

Single error correction double error detection (SECDED) is a widely used coding scheme for protecting the memory structures against soft-errors. However, in a high-failure rate situation, most coding schemes are not practical because of the strict bound on the number of tolerable faults in each protected data chunk. A 2D error correction coding scheme is presented in [21] that uses two sets of error detection codes on the rows and columns of the data array. As the failure rate sensitivity analysis results show in [21], this scheme is not appropriate for tolerating large number of randomly distributed failures. Further, the overhead of updating all the column codes for *each cache write* is high. Multiple bit error correcting codes like Hamming codes are capable of tolerating high failure rates, but are highly inefficient for on-chip caches. For instance, ECC-2 is a 2-bit error correction scheme based on Hamming code which is not normally used due to its significant coding delay, area, and power overheads [21].

Dual modular redundancy (DMR) schemes are used in many designs for providing memory structure reliability, but they are highly inefficient in terms of area/power overhead [36]. A popular architectural solution is to use redundant rows and/or columns [26]. However, as we discussed in Section 1, for our target failure rate, almost all word-lines/columns can be expected to be faulty from the start (Figure 2). Moreover, since redundant row replacement is based on a decoder modification and using hard-wired fuses, it is generally not applicable for more than 10 extra rows [16]. A similar set of methods are based on the cache block/row/way disabling that, as discussed previously, are also not suitable for high failure rate situations [32]. There are other groups of work that use a re-mapping table to map a faulty block onto one of neighboring functional blocks [17]. These methods impose a high pressure on the L1-L2 communication bus by increasing the L1 miss rate substantially. Furthermore, these methods have two major applicability issues: they are properly applicable only to *direct-mapped* caches [2]; and, they cannot be applied to L2 caches since a read from a faulty block results in a miss that gets its value from main memory with several hundred cycles latency. To target lower failure rates caused by process variation, ZerehCache introduces an interconnection network between row decoder and data array which requires significant layout modifications [4]. In this scheme, an external spare cache is used to provide redundancy; thus, applying the interconnection network allows a limited redundancy borrowing across the statically specified, fixed-size groups.

4.3 Quantitative Comparison to Alternative Methods

In order to illustrate the benefits of our design, we quantitatively compare AP with the baseline 6T SRAM cell, three well-known conventional cache protection methods (row redundancy, 1-bit error correction code (SECDED), and 2-bit ECC), and three state of the art works (ZerehCache (ZC) [4], Wilkerson et. al. [40], and 10T SRAM cell [8]). Table 2 summarizes this comparison – in $90nm$ – based on the minimum achievable V_{dd}, area overhead for the caches, processor clock frequency, normalized IPC, and normalized power. In order to have a fair comparison, the number of redundant rows and coding granularities are set so that the area overheads of the row redundancy, SECDED, ECC-2, and AP are equal/comparable. In this table, different techniques are sorted based on their minimum achievable V_{dd} – targeting 99% manufacturing yield for on-chip caches.

Table 2: Comparison of different protection schemes

Protection scheme	MinV_{dd} (mV)	Cache area overhead (%)	Freq. (MHz)	Norm. IPC	Power norm. to AP
6T cell	651	0.0	920	1.0	4.35
Row redun.	550	5.1	710	1.0	2.62
SECDED	530	6.3	670	1.0	2.35
ECC-2	490	7.4	580	0.96	1.87
ZC [4]	430	10.7	450	0.96	1.31
Wilkerson [40]	420	3.4	430	0.89	1.35
10T cell [8]	380	66	340	1.0	1.17
Archipelago	375	5.2	320	0.95	1.0

Overheads for AP are calculated by considering all extra SRAM structures, decoders, MUXing layer, comparators, bypass MUXes, and other miscellaneous small logics. However, some of the comparisons shown in Table 2 are conservative at best because we overlook: **1)** Area and delay overhead of the programmable decoder for row redundancy. **2)** Area and power overhead of the encoder and decoder for ECC and ECC-2. **3)** Power overhead of the extra logic which is added to the caches in [40]. **4)** A $380mV$ minimum achievable V_{dd} for the 10T cell was derived in $65nm$ [8] and it is clear that in $90nm$ this value would be higher.

Overall, even by overlooking all the mentioned overheads for other schemes, AP can still achieve the lowest V_{dd} and highest power saving among the other methods. The three closest competitors to our work are the 10T cell, Wilkerson [40], and ZerehCache [4]. However, the 10T cell incurs 66% area overhead which acts as a burden in the high power mode. In contrast, our scheme only has 5.2% area overhead and does not considerably influence the normal operation of the system. Comparing to [40], our scheme can achieve a significantly lower V_{dd} and higher power saving. In addition, Wilkerson's work suffers an 11% performance drop-off – for SPEC-2K – in low power mode and 6% in high power mode. Comparing to ZerehCache, our scheme achieves a considerably lower V_{dd}, power consumption, and area overhead. Overall, the inherent efficiency, high degree of freedom in redundancy replacement, and intelligent assignment of the spare elements are the main advantages of AP that allow it to tolerate a higher failure rate compared to the other techniques.

5. CONCLUSION

With aggressive CMOS scaling, dealing with power dissipation has become a challenging design issue. Consequently, a large amount of effort has been devoted to the development of DVS methods to tackle this problem. When decreasing the operational voltage of a modern microprocessor, large on-chip cache structures are the first components to fail. Tolerating these SRAM failures allows DVS to target lower V_{dd} values while preserving the core frequency scaling trend. In this work, we proposed a flexible fault-

tolerant cache design, Archipelago, which benefits from a high degree of freedom in redundancy substitution and an intelligent configuration algorithm for redundancy allocation and group assignment. AP allows fault-free operation in the near-threshold region by partitioning the cache to multiple autonomous islands with various number of word-lines to minimize the cache capacity loss. Our scheme, which relies on simple architecture and sophisticated configuration, enables DVS to reach $375mV$ in $90nm$. This translates to 79% dynamic and 51% leakage power savings for our target system which is modeled after the Alpha 21364. This significant amount of saving comes with 2% area and 4.6% performance overhead for the microprocessor when operating in low-power mode. Finally, we compared our scheme with several conventional and state of the art methods to illustrate its efficiency and effectiveness.

6. ACKNOWLEDGMENTS

We thank the anonymous referees for their valuable comments and suggestions. This research was supported by National Science Foundation grants CCF-0916689 and CCF-0347411 and by ARM Limited.

7. REFERENCES

[1] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. González. Low vccmin fault-tolerant cache with highly predictable performance. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, page To appear, 2009.

[2] A. Agarwal, B. Paul, S. Mukhopadhyay, and K. Roy. Process variation in embedded memories: failure analysis and variation aware architecture. *Journal of Solid State Circuits*, 49(9):1804–1814, 2005.

[3] A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(1):27–38, Jan. 2005.

[4] A. Ansari, S. Gupta, S. Feng, and S. Mahlke. Zerehcache: Armoring cache architectures in high defect density technologies. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, page To appear, 2009.

[5] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *IEEE Transactions on Computers*, 35(2):59–67, Feb. 2002.

[6] D. Bol, R. Ambroise, D. Flandre, and J. D. Legat. Analysis and minimization of practical energy in 45nm subthreshold logic circuits. In *Proc. of the 2006 International Conference on Computer Design*, pages 294–300, Oct. 2008.

[7] D. Brooks, V. Tiwari, and M. Martonosi. A framework for architectural-level power analysis and optimizations. In *Proc. of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[8] B. Calhoun and A. Chandrakasan. A 256kb sub-threshold sram in 65nm cmos. *2008 IEEE International Solid-State Circuits Conference*, pages 2592–2601, Feb. 2006.

[9] B. H. Calhoun and A. P. Chandrakasan. A 256-kb 65-nm sub-threshold sram design for ultra-low-voltage operation. *Journal of Solid State Circuits*, 42(3):680–688, Mar. 2007.

[10] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu. Improving cache lifetime reliability at ultra-low voltages. *Proc. of the 42nd Annual International Symposium on Microarchitecture*, 0, 2009.

[11] D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, pages 7–18, 2003.

[12] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. *Proc. of the 29th Annual International Symposium on Computer Architecture*, pages 148–157, 2002.

[13] H. Fujiwara, S. Okumura, Y. Iguchi, H. Noguchi, H. Kawaguchi, and M. Yoshimoto. A 7t/14t dependable sram and its array structure to avoid half selection. In *Proc. of the 2009 International Conference on VLSI Design*, pages 295–300, Jan. 2009.

[14] M. Garey and D. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[15] M. Hempstead, G. Y. Wei, and D. Brooks. Architecture and circuit techniques for low-throughput, energy-constrained systems across technology generations. In *Proc. of the 2006 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 368–378, 2006.

[16] M. Horiguchi. Redundancy techniques for high-density drams. In *2nd Annual IEEE International Conference on Innovative Systems Silicon*, pages 22–29, 1997.

[17] L. D. Hung, M. Goshima, and S. Sakai. Seva: A soft-error- and variation-aware cache architecture. In *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, pages 47–54, Washington, DC, USA, 2006. IEEE Computer Society.

[18] D. Kannan, A. Shrivastava, V. Mohan, S. Bhardwaj, and S. Vrudhula. Temperature and process variations aware power gating of functional units. In *Proc. of the 2008 International Conference on VLSI Design*, pages 515–520, 2008.

[19] R. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[20] S. Kaxiras, H. Zhigang, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. *Proc. of the 28th Annual International Symposium on Computer Architecture*, pages 240–251, 2001.

[21] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe. Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding. In *Proc. of the 40th Annual International Symposium on Microarchitecture*, 2007.

[22] W. Klotz. Graph coloring algorithms, 2002. Mathematik-Bericht 5, Clausthal University of Technology, Clausthal, Germany.

[23] J. Kowaleski, T. Truex, D. Dever, D. Ament, W. Anderson, L. Bair, , et al. Implementation of an alpha microprocessor in soi. *2003 IEEE International Solid-State Circuits Conference*, 1:248–491, 2003.

[24] J. P. Kulkarni, K. Kim, and K. Roy. A 160 mv, fully differential, robust schmitt trigger based sub-threshold sram. In *Proc. of the 2007 International Symposium on Low Power Electronics and Design*, pages 171–176, New York, NY, USA, 2007. ACM.

[25] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, pages 81–92, Dec. 2003.

[26] J. H. Lee, Y. J. Lee, and Y. B. Kim. SRAM Word-oriented Redundancy Methodology using Built In Self-Repair. In *IEEE International ASIC Conference '04*, pages 219–222, 2004.

[27] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proc. of the 2002 International Conference on Computer Aided Design*, pages 721–725, New York, NY, USA, 2002. ACM.

[28] K. Meng and R. Joseph. Process variation aware cache leakage management. *Proc. of the 2006 International Symposium on Low Power Electronics and Design*, pages 262–267, Oct. 2006.

[29] F. Moradi, D. Wisland, S. Aunet, H. Mahmoodi, and T. Cao. 65nm sub-threshold 11t-sram for ultra low voltage applications. *Intl. Symposium on System-on-a-Chip*, pages 113–118, Sept. 2008.

[30] Y. Morita, H. Fujiwara, H. Noguchi, Y. Iguchi, K. Nii, H. Kawaguchi, and M. Yoshimoto. An area-conscious low-voltage-oriented 8t-sram design under dvs environment. *IEEE Symposium on VLSI Circuits*, pages 256–257, June 2007.

[31] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *IEEE Micro*, pages 3–14, 2007.

[32] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou. Yield-aware cache architectures. *Proc. of the 39th Annual International Symposium on Microarchitecture*, 0:15–25, 2006.

[33] A. Raychowdhury, S. Mukhopadhyay, and K. Roy. A feasibility study of subthreshold sram across technology generations. In *Proc. of the 2005 International Conference on VLSI Design*, pages 417–422, Oct. 2005.

[34] D. Roberts, N. S. Kim, and T. Mudge. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 570–578, Aug. 2007.

[35] K. Sankaranarayanana, S. Velusamy, M. Stan, and K. Skadron. A case for thermal-aware floorplanning at the microarchitectural level. *The Journal of Instruction-Level Parallelism*, 2005.

[36] K. Sasaki. A 9-ns 1-mbit cmos ram. *Journal of Solid State Circuits*, 24:1219–1225, 1989.

[37] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, New York, NY, USA, 2002. ACM.

[38] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proc. of the 2004 International Conference on Dependable Systems and Networks*, pages 177–186, June 2004.

[39] K. Takeda, Y. Hagihara, Y. Aimoto, M. Nomura, Y. Nakazawa, T. Ishii, and H. Kobatake. A read-static-noise-margin-free sram cell for low-vdd and high-speed applications. *2006 IEEE International Solid-State Circuits Conference*, 41(1):113–121, Jan. 2006.

[40] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. Trading off cache capacity for reliability to enable low voltage operation. *Proc. of the 35th Annual International Symposium on Computer Architecture*, 0:203–214, 2008.

[41] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical report, Univ. of Virginia Dept. of Computer Science, Jan. 2003.