A Survey On Techniques Used For Designing Fault Tolerant and Process Variation Aware Memories and Caches

Sayed Aresh Beheshti-Shirazi
George Mason University
Farifax, VA, U.S.
sbehesht@masonlive.gmu.edu

ABSTRACT

Aggressive voltage and frequency scaling schemes applied to memory and cache structures, specially for memory systems fabricated in advanced and scaled geometry nodes that are severely affected by process variation, significantly increases the likelihood of read, write and access failures to/from memory cell array, and reduces the extent of frequency and voltage scaling. To remedy this problem, in the past decade, many researchers have investigated alternative and fault tolerant cache and memory organizations to mitigate the impact of process variation, and to reduce the failure rate of memory array in the results of voltage and frequency scaling. This survey paper discusses and compare many of such cache and memory design techniques.

KEYWORDS

Process variation, fault tolerant cache, fault tolerant memory, voltage and frequency scaling, DVFS

1 INTRODUCTION AND BACKGROUND

The scaling technology has delivered on Moore's prediction and has halved the feature size and nearly doubled the performance of integrated circuits every 18 months, a trend that has continued for over 3 decades. However, in smaller and scaled geometries, the fabricated transistor devices suffer from a phenomenon referred to as process variation. Process variation is variation in the electrical and physical property of fabricated devices due to the physical limitations of the fabrication process. The variation in the fabrication process such as the channel length, width, oxide thickness, and placement of dopants in a channel results in a large variation in threshold voltage [1].

Process variation has become an ever-increasing issue in the manufacturing and design of scaled semiconductor chips. Because of process variation, the scaling of the transistors dimension has a direct effect on the undesired scaling of the voltage supply. As a result, the process variation causes limitations on designing more power-efficient circuits.

With scaling the geometry node, the on-chip power density significantly increases. These problems are becoming worse by the need and the push for operating the Application-Specific Integrated Circuit (ASIC) devices at higher frequencies. At the same time, the contribution of leakage power is no longer negligible, and at scaled geometries, the leakage power becomes comparable or even dominates the dynamic power consumption of the chip. [3, 8–12]

Voltage scaling is a super effective knob to manage and reduce power consumption. This is because both dynamic and static (leakage) power consumption can be super-linearly reduced with a linear reduction in the supplied voltage at the expense of reducing performance. When it comes to logic, as far as the logic is operated (within a safe margin) above the threshold voltage, one should not expect reliability issues and the only concern will be the loss of performance. But when it comes to memory, in addition to the loss in performance, the scaling of the voltage also affects the reliability and predictability of the memory [13, 14, 31, 37, 38]. The reliability and predictability issue of memories at scaled geometries is related to how process variation impacts different transistors within each memory cell. In simple terms, process variation is the result of imperfection and in implementing transistor devices at scaled geometries. In other words, process variation is the direct result of the physical limitation of the manufacturing devices in shaping, doping, and connecting transistor devices. Under process variation, two identical devices at the design stage may end up with varying strength and threshold voltage after manufacturing [4, 5, 23, 31, 44]. In the result of variability (due to process), the read, write, and access time of the memory is not deterministic; in a simplistic model, each of read, write and access time can be modeled as a Gaussian distribution rather than a single value (that could be obtained from fixed-process spice simulation at design time). In such a model, the application of voltage scaling, not only shifts the mean of read/access/write time distribution but also changes the standard deviation of those distributions [35].

Figure 1 captures the results of a Monte Carlo simulation on a Six-Transistor Statistic Random-Access Memory (6T SRAM) cell under process variation in 32nm technology (with a standard deviation of 34mV for the threshold voltage [35]). As illustrated, the probability of cell, cache way, and cache failure exponentially increases with a linear reduction in the supplied voltage. Note that for obtaining this curve, the cycle time of the memory/cache is kept constant (to that of used in higher voltage). Hence, the increase in the probability of failure is directly related to the extent at which the tail of read/write/access distribution (as defined by process variation) extends out of threshold set by the defined cycle time (clock period). Furthermore, it is worth pointing out that depending on the choice of cycle time, different probability of failure curves can be obtained, however, regardless of the choice of cycle time, the trend will exhibit an exponential relation between supplied voltage and error rate.

In the past decade, researchers have proposed many novel and interesting solutions to remedy the sever impact of process variation in scaled geometries, the need for scaling the voltage, and the exponential increase of the failure rate of memory and cache units in scaled geometries. In larger geometries the impact of process variation in the performance of logic and memory cell was understood but was quite limited, resulting in a very tight distribution of

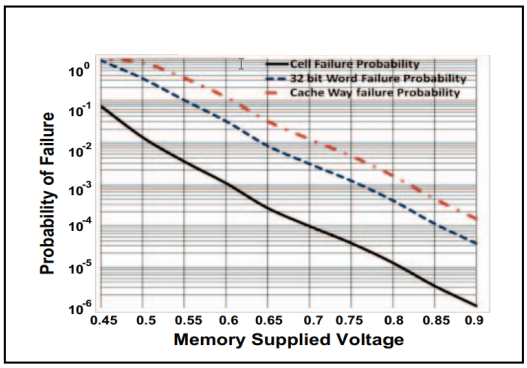

Figure 1: Probability of cell, way and cache failure in 32nm technology. for a 32KB 8 way associative cache organizations [33]

read/write and memory access time. Considering the very limited impact of process variation, the fault rate generated in the result of process variation was comparable or even less than that of stuck-at faults. In conventional memory design to guarantee high production yield in presence of Stuck-at faults and/or tolerating lifetime failures the replacement techniques such as extra rows and columns, were utilized [21, 40]. The memory design was simply altered by modifying and equipping the decoder with the added capability of using the redundant row(s) instead of defective memory row in case of such faults (that could be detected using memory march and testing solutions). However, in scaled geometries, the simple row or cache redundancy techniques became useless as the number of process-variation induced faults significantly out weighted the other rarely occurring faults. This promoted a desire and a need to design memory subsystems capable of tolerating a much larger number of faults (fault tolerance) in scaled geometries. In the rest of this section, I review several of the fault-tolerant cache and memory organizations studies and proposed in our research group in the last decade of research that are capable of tolerating a much large number of process-variation induced faults compare to the baseline memory design based on the simple use of redundancy. This paper is a survey paper on such fault and variation tolerant techniques. The rest of this paper is organized as follows: in section two we divide the various process variation mitigation techniques into three major categories and describe each category in the following sections three to five. Sections three to five describe the body of papers and techniques that could be categorized under each of these categories. In section six we describe the metrics used for evaluation and benchmarking of the suggested solutions. Section seven describes innovative solutions for process variation management. Finally, in section eight this survey paper is concluded.

2 OVERALL PROCESS VARIATION MITIGATION TECHNIQUES

In general, when discussing and analyzing various solutions to mitigate the effects of process variations the techniques can be categorized into three different categories as follow: Architectural-level, Circuit-level, and System and Software level. It is noted that either one or a combination of the aforementioned techniques can be applied during either the design phase of the new memory cell or memory run-time.

- **Circuit-Level solutions:** These solutions are implemented at the circuit level by considering a large margin to remedy the process variations effects [7] while designing the circuits. In this circuit designing approach, both the normal value and the worst case value due to the process variation is considered. The Circuit-based techniques can be applied during both the design phase of a new memory cell or memory run-time. The circuit-level solutions, provide solutions for the circuit that manage read/write to the memory solutions (such as wordline drivers, read/write circuitry, sense amplifier design, etc.) These techniques relying on the cell design, introduce alternative memory cell solutions that are more robust against process variation and/or Dynamic Voltage and Frequency Scaling (DVFS).
- **Architectural-Level solutions:** These solutions are addressing the effects of process variations using architectural techniques. When discussing architectural solutions we are addressing techniques that can take advantage of the attributes of the processor, and processor's elements such as cache, or the relationship between the optimum processing values such as performance, yield, energy or area overhead [26]. The architectural techniques can be applied during both the design phase of a new memory cell or memory run-time. The architectural solutions change the overall architecture of the cache, compared to the conventional design, and add additional units to the architecture that help with the management of the faults at scaled voltages.
- **Software and System Level Solution:** These solutions mitigate the impact of memory process variation in either system or software level. Generally, the system or user is aware of memory fault locations and map the applications to the memory to avoid the faulty locations. An alternative solution in fault-tolerant and approximate software solutions is to map the insensitive data to the faulty location, knowing that rare memory induced faults have a negligible impact on the performance of the overall system, or the software can recover from such faults.

Please see Fig. 2 for the detailed graphical description of Process Variation Mitigation Solutions. In the remainder of this paper, I am going to explain different classes/sub-classes along with a more detailed description of some example papers.

3 CIRCUIT-LEVEL SOLUTIONS:

Based on the above description, the circuit-based techniques to mitigate the process variation effects can be categorized as follow:

3.1 Circuit-Level, Memory Cell Modification:

In this type of technique, the operation is performed at a cell-level to mitigate the effects of Process Variation. For example, as recited by [1] a Built-In Self Test (BIST) circuity is used to detect Process Variation affected faulty cells during the run-time. Another example is a proposed technique disclosed by [17] regulates RESET current by intentionally reducing the RESET current for the cells that are difficult to reset to let them fail temporarily and then use error

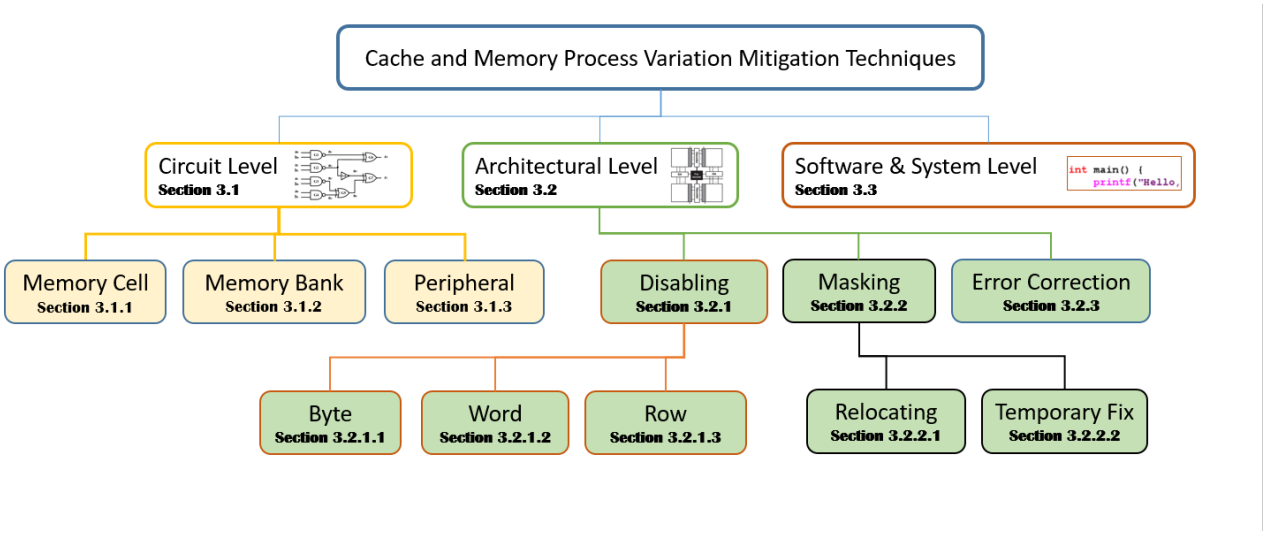

Figure 2: Classification of different techniques used for the mitigation of process variation in memories and caches.

correction schemes to recover them. As a result, the RESET current for the remainder of cells can be decreased which results in lifetime improvement of nonvolatile memory. The technique explored by [17] would also be categorized as Circuit-Level, Peripheral solution because it uses RESET current. Further explanation will be provided in the appropriate section.

3.2 Circuit-Level, Memory Bank:

In this type of technique, the memory process variation is mitigated at the Memory Bank level. For instance, each bank is providing support for adjacent defected banks. As cited in [2, 46] "linking", relocating the defective word to any row in the next bank, allows this architecture to achieve far larger fault tolerance in comparison with prior art solutions relying on cache resizing. In Re-sizable Data Composer Cache (RDC) as recited by [16, 34] banks are arranged in a circular chain, with each bank providing fault tolerance for the previous bank in the chain and the first bank providing tolerance for the last bank. Furthermore, [48] is another research paper exploring the Memory Bank related solution to mitigate the effects of process variation. The paper is using a cache management technique for tolerating process variation by developing a cache migration scheme that utilizes fast banks. The employment and utilization of fast banks through migration would provide a high-performance benefit. The performance is improved by 16% in comparison to the baseline that is determined by the slowest bank. In this section, I will provide a more extensive summary of [48].

In the remainder of this section, I will provide a more extensive description of some selected papers associated with Circuit-Level, Memory Bank technique to mitigate the effects of process variation.

Re-sizable Data Composer Cache (RDC) [16, 34]

In the following paragraphs, I am going to review the techniques explored by [16, 34]. RDC cache is the re-sizing of the cache in scaled voltages and avoiding a write/read from defective locations. The downsizing of the cache was explored in prior literature [2, 46], however, RDC cache refine this methodology to maximize the size of the cache at scaled voltages by partially re-using defective cache rows. The technique introduces a smart defect relocation methodology. It decomposes the data that is targeted for a defective cache way and relocates one or a few words of data to a new location while it avoids a write to the defective bits. Upon a read request, the requested data is recomposed through an inverse operation. The following three features a) compaction of relocated words, b) ability to use defective words for fault tolerance and c) "linking" (relocating the defective word to any row in the next bank), allows this architecture to achieve far larger fault tolerance in comparison with prior art solutions relying on cache resizing [2, 46]. In high voltage mode, the fault-tolerant mechanism of RDC-Cache is turned-off with minimal (0.91%) latency overhead compared to a traditional cache.

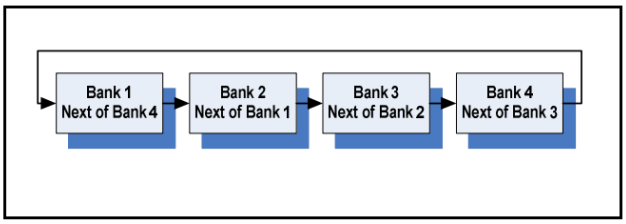

Figure 3: Banks are organized in a circular chain; for each bank, its next-bank will provide fault tolerance [34]

In the RDC approach, banks are arranged in a circular chain, with each bank providing fault tolerance for the previous bank in the chain and the first bank providing tolerance for the last bank. This is illustrated in Fig. 3. RDC-Cache provides a word-level fault tolerance. It generates and keeps a special defect map that has the defect information at a word level granularity. In RDC-Cache the last cache way in each row is used for Fault Tolerance (FT-way). If a cache way contains a defective word, the information that is

mapped to that defective word is relocated and saved in FT-way in its next bank in the circular chain. RDC-Cache uses a mechanism that allows saving the relocated words of two or more ways in one or more rows in a single FT-Way. An FT-way that all its words are used as a destination for relocated words is called “saturated”.

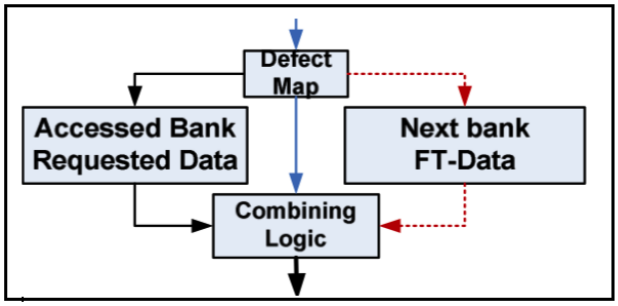

Figure 4: access to a RDC-Cache in low voltage mode [34]

When choosing the destination FT-Way for relocated words, the RDC-Cache first uses unsaturated FT-ways that contain defective word(s). Then it uses the unsaturated previously used FT-ways that are not yet saturated and finally the defect-free FT-ways. This allows us to keep the maximum possible number of defect-free FT-ways. Finally, if these FT-ways are not used they are released and used as ordinary ways in the cache. This increases the final RDC-cache size compare to previously suggested re-sizable caches [2, 46]. The process of associating an FT-Way in the next bank to a defective cache way is referred to as “linking”. The proposed structure allows the linking of any defective way to any FT-way in its next bank.

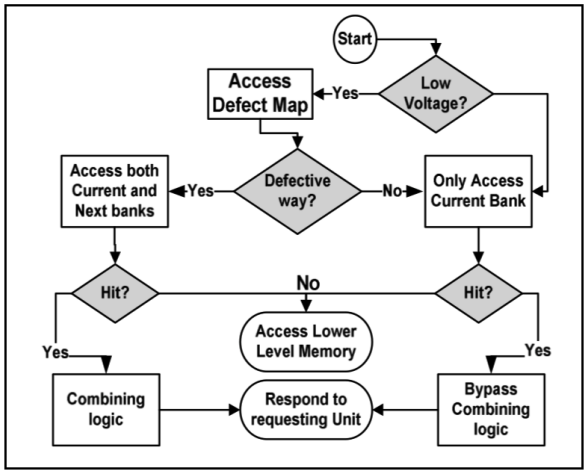

Figure 5: access flowchart to a RDC-Cache in low voltage mode [34]

At low voltage, when reading from a defective cache way, as it is illustrated in Fig. 4, the RDC-Cache first identifies the location of relocated words from the defect map and then accesses both banks (addressed bank and one containing relocated words) at the same time. Then through logical operations (combining logic), based on the defect map of an accessed cache way, FT-way, etc., it combines the information in both cache ways and generates the defect-free fetch group which is sent back to the requesting unit. The usage of another memory bank for remapping of defective words is a means to avoid designing multi-port caches to improve the area and delay of the cache. The access scenario to an RDC-Cache is illustrated in the flowchart in Fig. 5.

A raw defect map is generated at boot time. During the boot time, using the memory BIST unit, the L1 and L2 cache(s) are tested under low voltage conditions. The output of the BIST is a raw defect map containing one bit per each word in the cache. If there are multiple operating points for a different combination of voltage, temperature, and frequency, the BIST operation is repeated for each of these settings. The obtained defect map is then modified and processed to be usable with RDC-Cache.

A raw defect map has to be processed and converted to Relocation Aware Defect Map (RADM) format. Fig. 6 illustrates an example of how RADM fields are generated. This figure illustrates the RADM or two rows that use the same row in the next bank for fault tolerance. The first row [R:0011011] contains a defect in the third word of its second associative way. The second row [R:1101110] contains two defective ways one in the first way and the second one in its third way. The FT-way that is chosen in a row [R:0101110] of the next bank also has one defect in its FT-way. However, the total number of available words is equal to that needed for the tolerance of defects in rows ’0011011’ and ’1101110’. Fig. 6 also shows the RADM for each of these rows. Each RADM entry includes the defect map of the first 3 associative ways, the address of the row containing the FT-way in the next bank, the defect map of the FT-way in the next bank followed by three 2-bit Starting Word Location Indexi (SWLIi) fields. Each SWLIi index points to the location of the first relocated word in the cache way “i”. The ability to use a defective FT-Way for fault tolerance of defective ways in the main bank allows RDC to preserve the non-defective FT-ways to be used only if no other defective FT-way was available. Therefore, if after RADM generation some of the FT-ways were unused they could be released increasing the cache associativity in that row by one and the cache capacity.

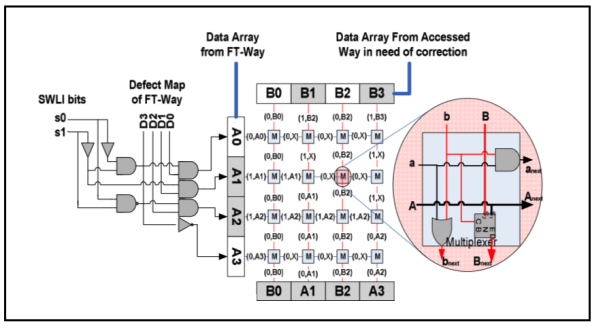

Figure 6: Combinational Logic Unit (CLU) used when reading data from RDC [34]

Reading a cache way containing defective words from RDC-Cache involves reading the addressed bank, reading the FT-way

from the next bank, and then passing the data through a Combining Logic Unit (CLU). The CLU also needs the defect map of the accessed cache way and the FT-way. With this information provided, CLU will process and combine the words in the defective way with those obtained from FT-Way and produce the final defect-free group of words to be sent back to the requesting unit. A simple realization of the combining logic for a 4-way associative cache is illustrated in Fig. 6.

As explained previously, the relocated words to the FT-way are saved in a compact form. This means that one FT-way might be used to store defect-free copies of defective words located in more than one row in the previous bank. The first 2bits of the Starting Word Location Index (SWLI) field in the defect map is used to realize the starting location (offset) of the first relocated word. More than one defective word may be in a cache way, however, all these words regardless of their location in the original cache way are compacted next to each other. For example, in Fig. 6 the words B1 and B3 are defective and they are compacted and saved in locations A1 and A2 in an FT-way. In this case, the SWLI index is "01" meaning the first word is either defective or used for fault tolerance of another cache way. The combination of (s0,s1) bits and defect map of the FT-way could be used to generate an array of bits (a0,a1,a2,a3) that indicate the locations of relocated words in the FT-way. A simple realization of such a circuit is provided in 6. This array of indexes along with defect map of the currently accessed defective cache way (b0, b1, b2, b3), its data (B0, B1, B2, B3) and finally the data of the FT-way (A0, A1, A2, A3) is the input to the Combining Mesh Grid (CMG). CMG is a matrix of M boxes. The functionality of each M box is very simple; M boxes help with routing the data words such that the relocated data words in the FT-way would find the proper location in the final fetch group.

Writing to a defective cache way in the RDC-Cache involves regrouping and compacting the words mapped to defective locations in the accessed way to their corresponding location in their associated FT-way. Before writing the information in the FT- way one should identify in which cache-way in the accessed bank, will the data be saved. Writing to the FT-way involves compacting the defective words together, shifting the compacted words to the appropriate starting word suggested by the SLWI index in the defect map, and then going through a muxing stage to make sure data will not be saved in the defective locations in the FT- way. This process is simply achieved by a Decomposition Logic Unit (DLU) similar to that used for combining. Note that in this case writing to the FT-way is on the critical path of the write operation. Furthermore, writing to the FT-way cannot start until data has propagated through the decomposition matrix (in case of a 4-way associative cache, it is the propagation delay of 4 multiplexers). Normally, the cache is designed so that, the write time is shorter than the read time. Thus, although writing to the FT-way extends the delay of write critical path, the write time is still expected to be much lower than the read time

The probability of a 32 KB RDC-Cache failure is illustrated in Figure 9. This figure also compares the Failure probability of other caches with the same size realized by different fault-tolerant Means. Authors adopt the definition for $V_{cc\text{-}min}$ as the voltage at which 1 out of every 1000 cache instances is defective [46]. With no fault-tolerant mechanism in place, a 32 KB cache composed of 6T SRAMS has a $V_{cc\text{-}min}$ of 0.87 V. Introducing a 1-Bit ECC reduces the $V_{cc\text{-}min}$ to 0.68 V. On the other hand, if memory array is realized via ST-Cells the Vcc-min is effectively reduced to 500 mV. However, an area penalty of 2X in the array size incurs. The Word-Fix Fault Tolerance mechanism suggested in [46] also reduces the $V_{cc\text{-}min}$ lower but close to 500 mV. The Bit-Fix mechanism in [46] further reduces the $V_{cc\text{-}min}$ to 480 mV. However, the cache size in both methods suggested in [46] is lower than that realized by RDC-Cache. Finally, the RDC-Cache realizes the Vcc-min at only 450 mV in 32 nm technology.

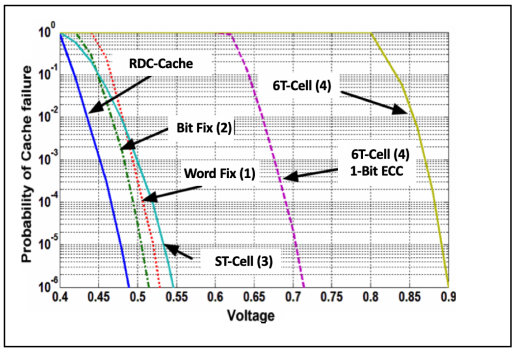

Figure 7: Probability of cache failure for different fault tolerant mechanisms including BitFix in [46], wordFix in [46], Schmitt Trigger 10T RAM in [19], RAM designed with conventional 6T memory cell, and RAM designed with conventional 6T and protected using 1-Bit ECC.

Process Variation-Aware Non-uniform Cache Management in a Three-Dimensional (3D) Die-Stacked Multi-core Processor [48]

The idea presented by the paper is to circumvent the effects of process variation by developing a cache management technique that would overcome the non-equal access times associated with the different memory banks. The cache management technique would migrate data from slow banks to fast banks to decrease the access time. The paper is using a 3D bank. The 3D bank is formed by dividing the bank into several sub-banks that are stacked together to form a 3D bank. Therefore, the interconnection wires inside the bank are all shorter which decreases the access latency of the memory. The energy overhead resulted from the proposed technique is negligible. The idea explored by the paper is to migrate data from slow sub-banks to fast sub-banks. The migration is happened based on either Bank Latency-based Migration Policy or Tiered Migration Policy.

The Bank Latency-Based Migration Policy always moves the data to the fastest sub-bank. For example, if we have 16 sub-banks, all 16 sub-banks in memory are ranked and sorted based on their access latencies in ascending order from Bank-0 to Bank 15. Upon accessing a cache line from a slow sub-bank, it migrates to the fastest sub-bank (Bank-0). The victim cache line from bank-i is moved to bank-(i+1).

The Tiered Migration Policy divides the sub-banks in one rank of memory into multiple tiers based on their speed. Therefore, each tier has multiple sub-banks with similar speed. Instead of always migrating data to the fastest sub-bank the data is migrated to the

fastest tier, and the victim cache line from the fastest tier is moved to the tier below. As a result, the impact on Bank-0 is now evenly divided between the plurality of sub-banks.

Paper further goes on to note that an increase in the number of tiers would not provide an extraordinary performance benefit. Therefore, it concludes that the two-tier enabled migration scheme is adequate and it achieves significant performance improvement. The energy overhead resulted from the proposed technique is negligible.

3.3 Circuit-Level, Peripheral level:

In this type of technique, the memory process variation is mitigated by employing a "Peripheral level" solution. This means adding an auxiliary part to the main memory to mitigate the effects of process variation. Different techniques are introducing different peripheral devices used to achieve the above-mentioned goal. One of these techniques is a proposed technique disclosed by [17] that regulates RESET current by intentionally reducing the RESET current for the cells that are difficult to reset to let them fail temporarily and then uses error correction schemes to recover them. As a result, the RESET current for the remainder of cells can be decreased which results in lifetime improvement of nonvolatile memory.

Moreover, as recited by [35] a small one step charge pump is employed to allow selective wordline overdriving. The detailed summary of the paper [35] is presented in the Architectural solutions section. Furthermore, the technique presented [1] is another example of using a peripheral circuitry, such as BIST to detect process variation impacted cells.

Furthermore, as recited by [22] the paper is employing a Physical Status Register (PSR) and augments the cache to localize the effects of process variations. As a result, it provides a finer granularity.

In more advanced technique, as suggested by [36] the BIST circuitry is employed in combination with the Fault-Tolerant Memory (FTM). At run time BIST tests the memory arrays at lower voltages and writes the defect map into BLB. BIST generates a different BLB for operation at each PVT corner. Defect maps for previously tested voltages could be saved by the system in non-volatile memory to avoid running the BIST in the future or can be rerun on demand. BLB should also be updated by possible defects in the result of the operating environment changes.

In IDC [36], a conventional cache is augmented with two auxiliary blocks. The BLB is a defect map for the cache and the IDC which is a small auxiliary cache that acts as dynamic redundancy to tolerate defects within the most recently visited, yet defective cache lines, in the conventional cache.

Furthermore, [29] employs a plurality of circuit-level techniques to change the access latency of selected cache lines based on the criticalities of the load instructions that access them.

In the remainder of this section, I will provide a more extensive description of some selected papers associated with Circuit-level, Peripheral technique to mitigate the effects of process variation.

Fine-grained Fault Tolerance for Process Variation-aware Caches [22]

In this paper, the effect of continuous scaling in CMOS fabrication process has been analyzed. As noted the highly persistent scaling makes the circuits more susceptible to effects of processes variation. Which results in delay, defects and/or leaky circuits. The paper [22] is specifically targeting spatially uncorrelated defects resulted from Random Dopant Fluctuation (RDF).

To lessen the impacts of process variations on caches, the paper [22] proposes an architectural technique at finer granularity by augmenting cache with a set of Physical Status Registers (PSR) to localize the effects of process variations at a word level. The technique proposed by the paper is an example of augmenting Cache with an additional resource which was one of the innovative solutions discussed in section four of the instant survey paper. The technique discussed provides 10% less performance loss with 90% of cases. As per area overhead considering additional SRAM cell area the proposed technique gives the area overhead of 3%.

In this technique, the Faulty words are disabled or shut down completely and access to those words is bypassed to a small set of word-length buffers. This technique is shown to be effective in reducing the performance penalty due to process variations and in increasing the parametric yield up to 90% when subjected to the performance constraints. The new fine granular technique presented in this paper [22] provides resilience at much higher defect densities which are predicted in the future process technologies due to large deviations in V_t.

Individual 32-bit words of a cache block are augmented with an extra bit indicating those as either defective or not defective. These defective words are substituted by another set of word-length fault buffers tagged with the faulty-word location in the cache.

The PSR contains one bit for each word in a block/cache-line. These word-level defect-bits are multiplexed with the word offset part of a memory address. Hence, a particular cache access is considered defective if the bit corresponding to the accessed word is set. PSR multiplexing is done in parallel with tag matching, therefore, the overhead of this scheme is only the area for PSR and their energy consumption while no significant addition in the critical path.

Because the defective portions of cache lines are never used, they may be turned off to avoid leakage power loss using previously proposed leakage power reduction techniques like gated-VDD [4]. Rendering healthy words usable with PSR has a significant benefit, compared to other techniques, as will be demonstrated in the next section. PSR-array is configured using a March test that writes a cache with test patterns and then reads them to detect defective words. Based on this test, each bit in the PSR is set or cleared. If this configuration is predicted to be invariant of operating conditions and aging, PSR can be fused in the cache permanently or the configuration can be stored in flash memory to be accessible on system boot.

In conclusion, the paper [22] discloses a fine-grained fault-tolerant cache architecture to mitigate process-variations artifacts on SRAM-based caches. The proposed technique makes efficient use of residual cache capacity by turning off or disabling only defective cache words and supplementing those with a set of fault buffers. Fault buffers-augmented cache is shown to recover parametric chip yield up to 90% sustaining a maximum performance loss of 10% while achievable parametric yields of other schemes do not surpass 60%.

Inquisitive Defect Cache (IDC) [36]

The technique used in IDC [36] is an architectural technique to address process variation using a dynamic redundancy, where the redundant rows are used only to mitigate process-variation-induced errors withing the window of execution of a program within the cache. The IDC cache-organization dynamically maps the in-use defective locations in a processor cache to an auxiliary parallel memory, creating a defect-free view of the cache for the processor. This allows the limited yet available redundancy resources to be used for tolerating a large number of defects as at each point of time, the available redundancy is used to mitigate errors in a subsection (most recently visited) of the cache.

Fig. 8 illustrates the IDC cache organization. In IDC, a conventional cache is augmented with two auxiliary blocks. The BLB is a defect map for the cache and the IDC which is a small auxiliary cache that acts as dynamic redundancy to tolerate defects within the most recently visited, yet defective cache lines, in the conventional cache. Both of these auxiliary structures are kept at high supply voltage when the voltage of the main cache is scaled. This is to guaranty reliable storage of defect information in the defect map and to avoid process-variation-induced defects in the smaller cache used as dynamic redundancy. The size of the added memory structures, compared to the main cache, is significantly smaller.

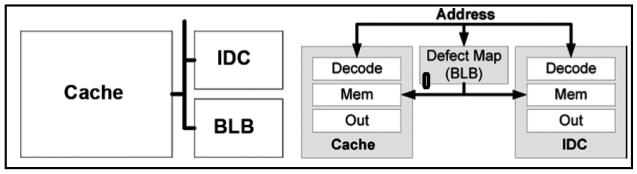

Figure 8: The organization of Inquisitive Defect Cache [36]

Generating the defect map BLB could be accomplished at manufacturing time, however testing for multiple vectors significantly increases the chip cost. Another approach is to generate the defect map at boot time. In this method, at manufacturing time, the memory is only tested for manufacturing defects at the nominal voltage process corners, fixing such errors using the available redundancy and leaving the process variation induced defects to the combination of the FTM and BIST unit. At run time BIST tests the memory arrays at lower voltages and writes the defect map into BLB. BIST generates a different BLB for operation at each PVT corner. Defect maps for previously tested voltages could be saved by the system in non-volatile memory to avoid running the BIST in the future or can be rerun on demand. BLB should also be updated by possible defects in the result of the operating environment changes.

The number of words that are read from a cache at each read cycle is referred to as a Fetch Group (FG). The size of the IDC cache could be much smaller than the total number of defective FGs in a cache. In a sense, the IDC could be considered as a group of redundancy rows with the main difference being that mapping to redundancy is fixed and one time, whereas mapping to the IDC is dynamic and changes with changes in the addressing behavior of the program. Furthermore, in the IDC case, the defective FGs within the WoE of the program in the cache is mapped to IDC allowing much higher coverage than that of a redundancy scheme.

The conceptual view of an IDC enabled cache structure is illustrated in Fig. 9, where IDC, Cache, and BLB represent the Inquisitive defect cache, cache, and defect map BLB, respectively. The WoE and "Int" are conceptual structures (do not exist but drawn to ease the understanding of IDC structure and its operation) representing the WoE of the program in the cache and intersection of the WoE and BLB, respectively. The BLB marks the defective FGs in the cache. The WoE marks the in-use section of the cache (FGs that the processor has recently visited). The "Int" is in the results defective FGs in the WoE which should be mapped to IDC. In Fig. 9, the IDC and cache have associativity of 2 and 4, respectively. If the IDC size and associativity are chosen properly, the WoE of a program that is mapped to the cache (after its first pass) should experience very few defective/disabled words and could be virtually viewed as a defect-free segment in the cache by the processor.

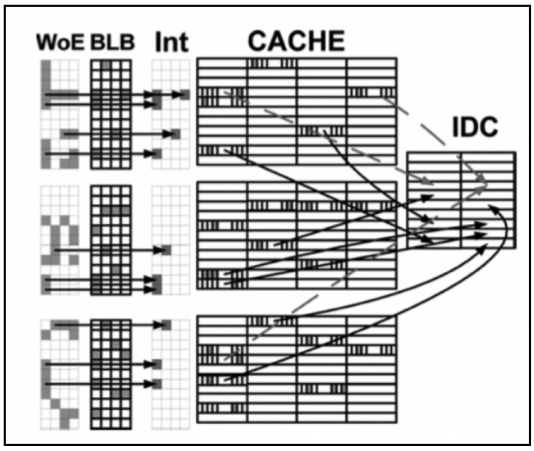

Figure 9: The organization of Inquisitive Defect Cache [36]

If an FG is defective, its information could only be found in the IDC since every read/write access to defective words is mapped to the IDC. The BLB should be designed such that its access time is smaller than the cache cycle time. Considering the small size of the defect map when one bit per FG is chosen, and also the fact that the defect map voltage will not be scaled makes it fairly easy to meet this design constraint. Fig. 10 compares the minimum cycle time of a 16KB IDC to that of a traditional cache of the same size. As illustrated, the IDC allows the system to operate at lower cycle times that are can be up to 25% shorter at sub-500 mV supply.

Process variation in embedded memories [29]

The paper [29] is purposing an adaptive cache management policy based on non-uniform cache access along with a latency compensation approach that employs several circuit-level techniques to modify the selected cache line access latency based on the level criticalities of the load instructions that access them.

The paper addresses on-chip data caches three different aspects "Exploiting Variable access latency," "Selective latency compensation," and "Experimental evaluation." Furthermore, it presents the simulation platform and default values of major simulation parameters. it discusses the impact of process variation on cache memories.

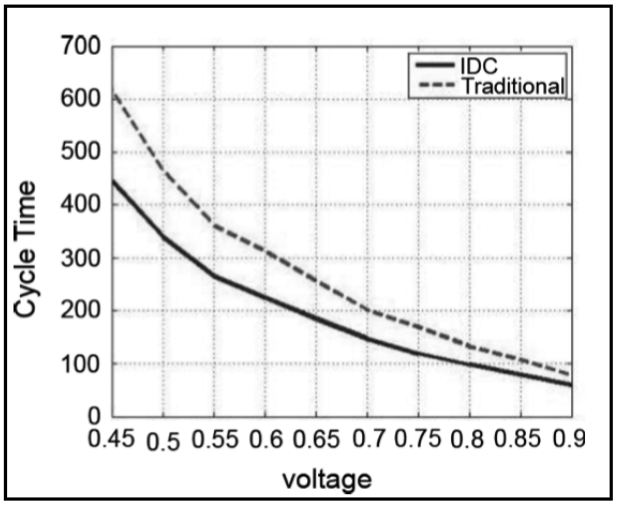

Figure 10: Minimum cycle time improvement when IDC is used to realize a faster [36]

Parameter	Value
Fetch Queue size	2 instructions
Fetch width	2 instructions/cycle
Decode width	2 instructions/cycle
Issue width	2 instructions/cycle (out-of-order)
Commit width	2 instructions/cycle (in-order)
RUU size	128 instructions
LSQ size	32 instructions
Functional units	2 Integer ALUs, 1 Integer multiply/divide, 1 FP add, 1 FP multiply/divide
Branch predictor	Combined, Bimodal 2K table, 2-Level, 1K table, 8 bit history, 4K choser
BTB	1K-entry, 4-way
Mispredict penalty	18 cycles
L1 data cache	16K, 4-way (LRU), 32B blocks, 2 read ports and 1 write port 2 cycle latency (without variation), 4 cycle latency (with variation)
L1 Inst. cache	16K, 2-way (LRU), 32B blocks, 2 cycle latency, 2 read ports and 1 write port
L2 cache	Unified, 2MB, 8-way (LRU), 64B blocks, 8-cycle latency
Memory	160 cycles
ITLB	16-entry, 4KB block, 4-way, 30-cycle miss penalty
DTLB	32-entry, 4KB block, 4-way, 30-cycle miss penalty

Figure 11: Default Configuration Parameters [29].

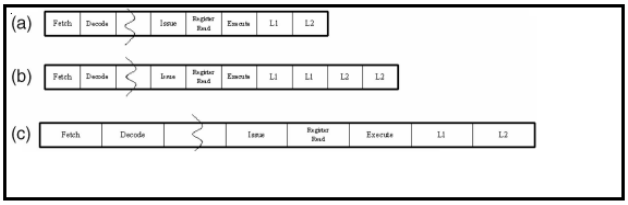

Figure 12: Different Scenarios assuming an original cache access latency of two cycles. (a) Pipline progress without process variation. (b) Increasing number of cache access cycles. (c) Increasing clock cycle time. [29].

Presents the technical details of variable latency exploitation and selective latency compensation techniques, that are respectively purposed by the authors.

In the implementation, for both L1 data cache and L1 instruction cache the paper has assumed two read papers and one write port. The availability of the port is checked and if a port is available, the port is assigned to the requested load instruction. For the sake of simplicity, if the port is not available the requested load instructions are blocked. The default configuration parameters are shown in Fig. 11. The paper uses six SPEC 2000 CPU integer benchmarks and six SPEC 2000 CPU floating-point benchmark. ([29])

As illustrated by Fig. 12, The paper's suggested approach is compared against three different pipeline structures. The first one is the ideal case without process variation as disclosed by Fig. 12(a) In this scenario, the cache access is assumed to be pipe-lined into two single-cycle stages. The second scenario as depicted by Fig. 12(b) discloses one solution to combat the effects of process variation which impacts both these pipeline stages during cache access. The delay of either pipeline stage can exceed that of the nominal cycle time due to process variation. The third scenario as depicted by Figure 19.C discloses a different solution to process variation effect by increasing the cycle time. Please note that solutions in both Figure 19b and Fig. 12(c) are based on the worst-case design paradigm and can lead to significant losses in performance.

The most important parameters effected by process variations include channel length (L), gate oxide thickness T_{ox}, and threshold voltage V_{th}.

In this work, the threshold voltage (V_{th}) variation has been considered as the major cause of intra-die variation, since the effect of other parameter variations can be translated as an effective variation in threshold voltage. The intra-die variations can result in *Functional failures*, *Timing failures*, and *Noise margin reduction*.

Figure 20 shows the proposed architecture for variable latency caches and the corresponding pipeline. This architecture has two major components: *Set Predictor* and *latency table*. The cache set address of the data to be accessed is predicted by the set predictor. In this work, the paper uses a 2-delta stride-based address predictor for set prediction. Stride-based address predictors predict the next address by adding the sum of the most recent address to the difference between the two most recent addresses produced by an instruction.

In the 2-delta stride-based method, two strides are maintained. One of these strides (s_1) is always updated by the difference between the two most recent values, whereas the other stride (s_2) is used for computing the predictions. When (s_1) occurs twice in a row, then it is used to update the prediction stride (s_2). Once the address of a load instruction is predicted, we compute the corresponding set address. We consider the size of the prediction table as $2K$ entries because it is providing good accuracy with the nominal power and area overhead as compared to other sizes such as $1K$, $2K$, $8K$, and $16K$ entries.

The second element of the technique suggested by the paper design is the latency table obtained by augmenting the March test that maintains the access latencies at a cache set granularity. The

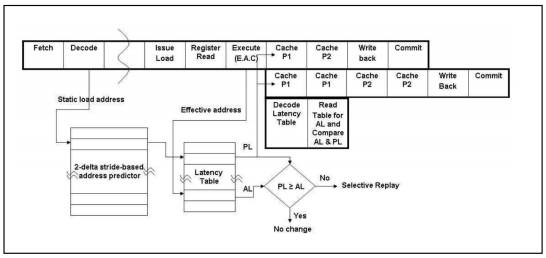

Figure 13: Proposed architecture and corresponding pipeline for exploiting variable latency data cache. PL and AL are predicted and actual latencies respectively. [29].

latency table is adopted during the functional testing of memory components.

In comparison, the experiments with the proposed scheme (with variable latencies under the original clock frequency) show an average increase of around 0.67 percent in execution time as compared to the ideal case with no process variation. The proposed architecture benefits over the baseline architecture by permitting dependent instructions to be issued based on the variable latencies captured by the latency table.

In this work, the paper [29] focused on the timing failures resulting from process variation and proposed two novel techniques to manage these failures. Our first approach deals with timing failures by allowing, where possible, some cache sets to operate with higher access latencies. The second approach tries to compensate for the effects of process variation by trading off higher energy consumption and/or accelerated aging with better performance. The proposed mechanism would result in a process variation mitigation solution that is a combination of architectural and circular based solution.

4 ARCHITECTURAL-LEVEL SOLUTIONS:

Based on the above description the architectural techniques to mitigate the process variation effects can be categorized as follow:

4.1 Architectural-Level, Disabling:

In this technique, as recited by[27] faulty cache blocks are disabled, and the programmable address decoder is re-mapping references to faulty blocks to the healthy blocks.

The Architectural-Level, Disabling technique can be further categorized as follow:

4.1.1 ***Disabling, Byte-Level:*** The disabling technique is performed at Byte-level. As recited by [42] Point-and Discard (PAD) technique instead of the entire cache line only faulty Bytes is discarded when a hard error occurs, and healthy Bytes are used in the impacted cache line.

4.1.2 ***Disabling, Word-Level:*** In this technique, the disabling technique is performed at Word-Level. As suggested by [35] a selective wordline overdriving technique is utilizing a small one step charge pump. In the approach of [35], a modified wordline driver peripheral device is used to allow selective wordline overdriving.

In the remainder of this section, I will provide a more extensive description of some selected papers associated with Architectural-level, Disabling, Word-level techniques to mitigate the effects of process variation.

Process Variation Aware SRAM/Cache for Aggressive Voltage-Frequency Scaling [35]

In CP cache [35], the peripheral circuitry of the SRAM is modified to selectively allow over-driving a word line which contains weak cell(s). This architecture allows reducing the power on the entire array; however, it selectively trades power for correctness when rows containing weak cells are accessed. Cell sizing is designed to assure successful read operations. This avoids flipping the content of the cells when the word line is over-driven.

Fig. 14 outlines the CP-Cache word-line driver architecture. It consists of two consecutive NAND gates(possibly preceded by two or more inverters to increase fan-out). The second NAND gate's pull up PMOS (P1) as shown in Fig. 14 is connected to the supply voltage. Pull up PMOS (P2) is connected to the charge pump output. Using this driver P1 first drives the word line to Vdd and conditioned upon activation of P2 the charge pump and word-line start charge sharing. Being at a higher voltage the charge pump's capacitor charge migrates to the word-line, effectively raising its voltage as shown in Fig. 15.

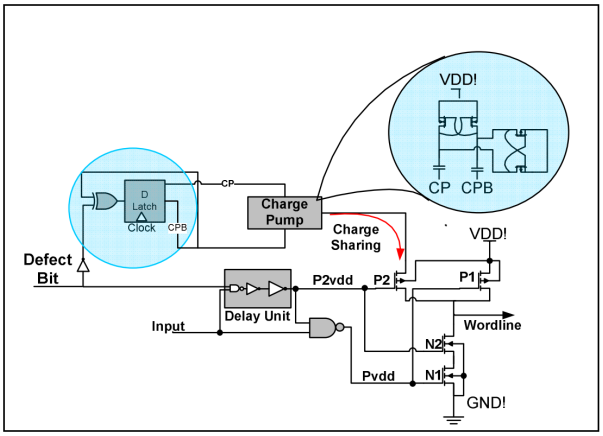

Figure 14: Proposed wordline driver [35]

The selective behavior of this circuit is controlled by the input to the Nand-gate in the delay unit of the word-line driver. If the external-input to this Nand-gate is high, the word-line overdrive will be inactive whereas a low input initiates the overdriving behavior. Typically, the input to the Nand-gate will be from a defect map that stores the results of a BIST run that is performed for each new set of voltage, frequency, and temperature. This approach assures support for a large number of operation modes. Alternatively, if the system is known to have a small number of operating modes, the configuration information can be stored inside fuses that are configured for each mode.

Overdriving the word-line driver may cause cell stability problems. The access and pull-down transistors in an SRAM cell during

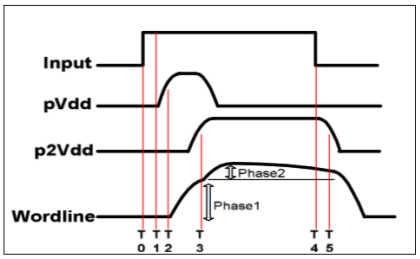

Figure 15: Signals timing order in the proposed charge pumped wordline driver[35]

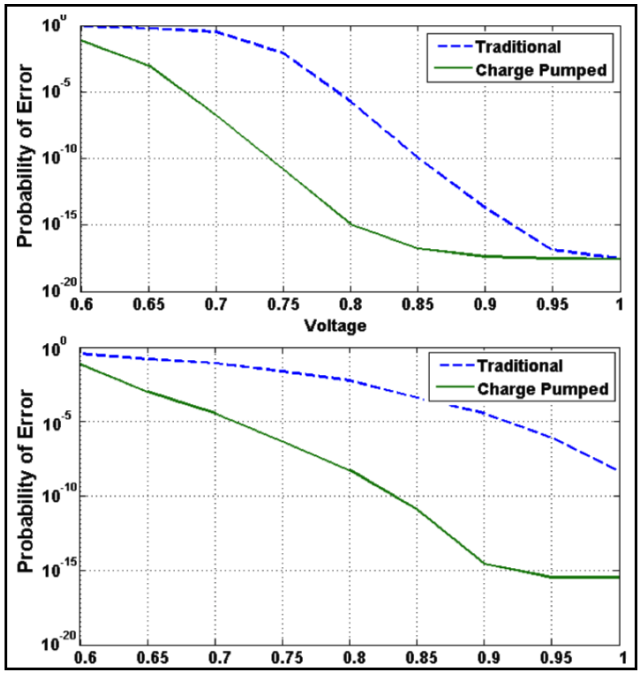

Figure 16: Total probability of failure (Weite Failure +Access Failure +Read Failure) for FFVS policy (top) and FSVS policy (buttom)[35]

a read form a voltage divider. The size of the pull-down transistor is chosen large enough to assure that the rise in the intermediate node of the voltage divider smaller than the threshold voltage of devices used in the cell. Increasing the voltage of the access transistor lowers the resistance of the access transistor effectively increasing the voltage at this intermediate node. This, in turn, increases the likelihood of a bit flip during a read operation. To prevent this effect, one could trade cell area increases the size of the pull-down transistor to counter the voltage overdrive impact on cell stability. Increasing the size of the pull-down device also impacts the Static Noise Margin (SNM) and the leakage of the SRAM cell.

Overdriving the word-line using a charge pump improves both mean and standard deviation of the access/write time distribution. As illustrated in Fig. 16, this results in a significant reduction in read access and write failure at scaled voltages.

4.1.3 ***Disabling, Row-Level:*** In this technique, the disabling technique is performed at Row-Level. For example, the technique as suggested by [21, 40] the memory design is simply altered by modifying and equipping the decoder with the added capability of using the redundant row(s) instead of defective memory row in case of such faults (that could be detected using memory march and testing solutions).

4.2 Architectural-Level, Masking:

4.2.1 ***Architectural-Level, Masking, Re-locating:*** In this type of technique, the memory cache is modified to combat the effects of process variation. For example, the cache blocks can be rearranged [28], to minimize the number of sets having both high and normal latency. This is achieved by mapping and grouping together Process Variation affected blocks in a few sets.

Furthermore, reviewing the techniques explored by [16, 34], the paper could also be categorized into Architectural-level, Masking, relocating technique since the technique introduces a smart defect relocation technique, wherein the technique decomposes the data destined to be transferred to a defective cache way and relocates one or a few words of data to a new location. As a result, it avoids a write to the defective bits. The detailed analysis of [16, 34] has been presented in Circuit-Level, Memory Bank sub-section.

In the remainder of this section, I will provide a more extensive description of some selected papers associated with Architectural-level, Masking, re-locating techniques to mitigate the effects of process variation.

Dynamic Cache Pooling for Improving Energy Efficiency in 3-D Stacked Multi-core dynamic processor [48]:

This paper proposes a run time policy to dynamically appropriate jobs to cores on the three-dimensional stacked system based on applications cache needs and cache uses. The applications with different cache needs are paired up with each other. The technique increases energy efficiency by providing flexible heterogeneity of cache resources. The proposed technique is of circular and architectural nature. The dynamic policy is allowing the cache to be re-sized while job scheduling is based on pairing applications with contrasting needs.

The analysis shows that the suggested dynamic run time policy is increasing the three-dimensional processors' Energy Delay Product (EDP) by up to 39.2% in comparison to three-dimensional processors with static cache sizes. As per Energy Delay-area Product (EDAP) the dynamic policy proposed by the paper increases by 57.2% in comparison to its counterpart having static cache sizes.

The technique disclosed by the paper [24] enables vertical cache resource pooling in three-dimensional architecture. The disclosed architecture as displayed by Fig. 17, has a four-layer 3-D system with one core on each layer wherein a private L2 caches of the cores are capable of increasing their size by exploiting the cache resources from other layers.

Fig. 17 (a) and (b) are the baseline 3D systems with 1MB and 2MB static private L2 caches, respectively. Furthermore, in Fig. 17 (c), each core has a 1MB private L2 cache and the vertically adjacent

caches are connected using Trough Silicon-Vias (TSVs) for enabling cache resource pooling.

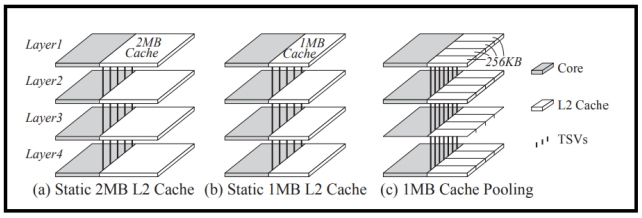

Figure 17: Proposed 3D system with cache resource pooling versus 3D systems with static 1MB and 2 MB caches. In (c), cores are able to access caches on the adjacent layers through the TSVs. [48].

The re-sizeable cache allows the cores to increase their respective L2 cache sizes by the expense of cache resources from other layers with negligible access latency penalty. The design achieves two objectives of increasing the performance and power efficiency by respectively increasing the cache size and turning off unused cache partitions.

The performance improvement resulted from increasing the cache size is based on the modification of the cache associativity. The cache associativity is calculated based on block size, the number of sets and the level of associativity. While the power efficiency is obtained by selectively turning off unneeded cache ways. Each cache way is called a cache partition. Each partition is independently poolable to one of its neighboring layers.

The paper introduces a Local Cache Status Register (LCSR) for each cores private L2 cache partition (e.g., there are four partitions in a 1MB cache in our design) to store the status of the cache partitions. The paper [24] also introduces Remote Cache Status Registers (RCSR) for the L1 cache so that the L1 cache is aware of its remote cache partitions. RCSR and LCSR logics are illustrated in Fig. 18 (a), (b), and (c).

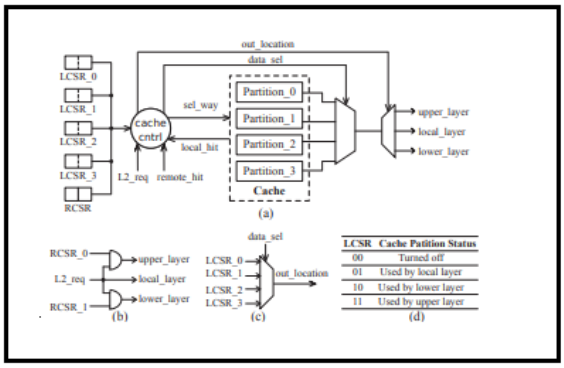

Figure 18: Cache resource pooling implementation: (a) Logic for cache resource pooling (b) L2 request generation logic (c) Output location generation logic (d) Local cache status registers. [48].

There are four different scenarios for each of the local cache partitions. (1) used by the local layer, (2) used by the upper layer, (3) used by the lower layer and (4) being turned off. The core's LCSR has two bits to specify the current status of the corresponding local cache partition as indicated by the table in Fig. 18 In addition, two 1-bit RCSR is allocated in L1 caches for each core to be aware of its remote cache partitions. Upon receiving the requests and addresses, the tag extracted from the requested address will be cross-referenced at each block with the tag array. Meantime, the entries of each way will be selected based on the index.

Furthermore, the paper presents the flow of our run-time job allocation and cache resource pooling policy. The policy contains two stages (1) Job allocation which decides on which core each job should run on, and (2) cache resource pooling, which distributes a pool of cache partitions among a pair of applications.

This paper has introduced a novel design for three-dimensional cache resource pooling that necessitates minimal circuit and architectural modification. In addition, an application-aware job allocation and cache pooling policy improves the energy efficiency of three-dimensional systems.

Working with Process Variation Aware Caches [27]

In this paper, a block-rearrangement technique has been proposed to minimize the performance penalty resulted from a process variation aware cache which works at set-level granularity. The paper rearranges physical locations of cache blocks by distributing blocks of a cache set over multiple sets to minimize the number of sets being impacted by process variation. Furthermore, the impact of access latency variations on performance is analyzed. Based on the proposed design and different tested benchmark the worst-case performance penalty for the proposed technique would be 7.76%. As per area overhead the paper notes that area overhead is similar to [39]. It is worth noting that the reported area overhead for [39] is 10%. The proposed block-rearrangement technique is an example of one of the many architectural techniques developed to combat the effects of process variations.

The concentration of the paper experiment is on on-chip data caches. To decrease performance loss resulted from the process variation the technique is using a process variation aware cache which utilizes access latency fluctuations.

The paper is analyzing two rearrangement techniques, wherein the cache blocks are rearranged between a pair of cache sets and between all cache sets. The two rearrangement techniques are paired block rearrangement technique (PairedBRT) and perfect block rearrangement technique (PerfectBRT), respectively. In PairedBRT arrangement two adjacent sets are considered as a group and block rearrangement is performed inside the group. In PerfectBRT, the block can be rearranged to any location within the set it belongs to in the associated cache way. In both techniques, blocks in a cache way are rearranged by moving the high latency blocks to the bottom of a group or an entire cache way. Each of the suggested techniques have their own pros and cons it is noted that PairedBRT is simple to implement, whereas PerfectBRT is very effective in performance.

This paper [27] is using a 2-way set-associative cache which has 8 sets (numbered from 0 to 7). Blocks of a set i are represented as block i in both way 0 and way 1. In Conventional Addressing Scheme (CAS), all blocks of a set are placed in a single row so that even if one block is affected by process variation, the corresponding

set takes high access latency. From CAS part of Figure 19, we know that two sets, i.e., sets 5-6, take low latency and all other sets take high access latency as they have at least one high latency block.

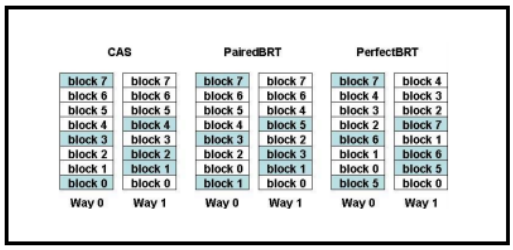

Figure 19: An illustration for Cache block organization in CAS, PairedBRT, and PerfectBRT. Shaded portions indicate the blocks which are affected by process variation. [27].

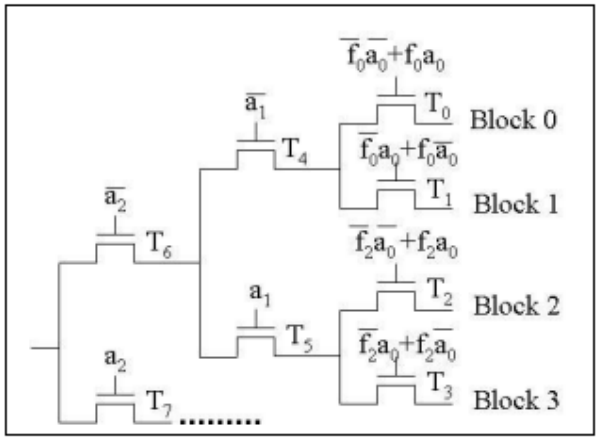

Figure 20: Decoder configuration in Paired-BRT. Here f_i indicated whether or not the i^{th} block is affected by process varaition. [27].

In PairedBRT section of Fig. 19, blocks which belong to sets 2i and 2i + 1, 0 i 3, are rearranged, it is noted that four sets, i.e., sets 0, 2, 4, and 6, take low latency and the other sets take high latency. Note that after applying PairedBRT, we have set 5 with one high latency block in way 1 and set 7 with one high latency block in way 0. As these two sets are differently grouped, it is not possible to rearrange their blocks. We can overcome this problem by using PerfectBRT, where any block can be placed anywhere in its cache way. PerfectBRT part of Fig. 19 shows that five cache sets take low latency.

To rearrange cache blocks, the paper [27] considers a programmable address decoder to disable faulty cache blocks. Wherein the programmable address decoder is re-mapping any references to the faulty blocks to the healthy blocks. To program the address decoder, the paper suggests use of the March test which can differentiate the low and high latency cache blocks.

This paper examines a 4-way set-associative cache where each way has one decode. The modified decoder which is used in Paired-BRT is shown in Fig. 20 PairedBRT rearranges blocks of adjacent sets. The block characterization attributes f_i, which is resulted from the March test, are used in combination with the attributes a_0 and a_0 to update the control inputs of the pass transistors. For example, if we consider block 0 and block 1 with $f_0 = 1$, then block 0 is mapped to block 1 and block 1 is mapped to block 0. In PerfectBRT, we need to program at all levels.

By considering a process variation aware data cache layout that operates at set-level granularity, the paper proposed a block rearrangement technique to minimize the performance penalty due to access latency variations in data caches. By validating the proposed technique using SPEC2000 CPU benchmarks, showed that the proposed technique can drastically decrease the performance penalty happened by a conventional addressing scheme.

4.2.2 ***Architectural-Level, Masking, Temporary Fix:*** In this type of technique, the processor is modified to combat the effects of process variation. For example, the processor pipeline is modified [41] to improve performance. In this technique, the pipeline clock is a period equal to the average stage delay instead of the longest one. In another word, the spare time of the faster stage is added to the slow ones by skewing clock arrival times to latching elements after fabrication.

Another example of processor modification to improve the effects of process variation would be the employment of an additional cache to mitigate the impact of Translation Look-aside Buffer (TLB) latency variation as recite by [18]

Variation Trained Drowsy Cache (VTD) [32, 33]

The ideas proposed in VTD [32, 33] is to manage the voltage scaling at a very fine granularity, where each cache way can be sourced at a different voltage. The selection of voltage levels depends on both the vulnerability of the memory cells in that cache way to process variation and the likelihood of access to that cache location. After a short training period, the proposed architecture will micro-tune the cache, allowing significant power reduction with a negligible increase in the number of misses. Also, the proposed architecture actively monitors the access pattern and re-configures the supply voltage setting to adapt to the execution pattern of the program. The novel and modular architecture of the VTD-Cache and its associated controller makes it easy to be implemented in memory compilers with a small area and power overhead.

The VTD-Cache conceptually operates by allowing a fine-grain control over the voltage of each cache way. Among available voltage levels, the supplied voltage is chosen by a simple voltage selector that is implemented at each cache way. The voltage selector dynamically changes its state as the processor explores new segments of the running program and shifts and/or resizes its window of execution (WoE). In VTD-Cache each cache way can be supplied from one of three possible voltages. The lowest voltage level supplies Data Retention Voltage (DRV) for cold lines which are determined by cache access pattern and are managed via the proposed architecture. Cache ways that are supplied with this voltage are referred to as “Cold Ways”. The remaining two voltage levels are used in cache ways located within the Cache Window of Execution (CWoE).

V_{dd}^{Low} is supplied if the cache way could operate correctly in that voltage, otherwise, the cache way is supplied with V_{dd}^{High}. In VTD cache ways are referred to as **Warm** and **Hot** Ways if they are supplied from V_{dd}^{Low} or V_{dd}^{High} accordingly. The decision to use which supply voltage is made based on a defect map that is generated using the memory BIST. Section III-F further elaborates on how the Warm and Hot cache ways are redefined based on the change in the operational temperature.

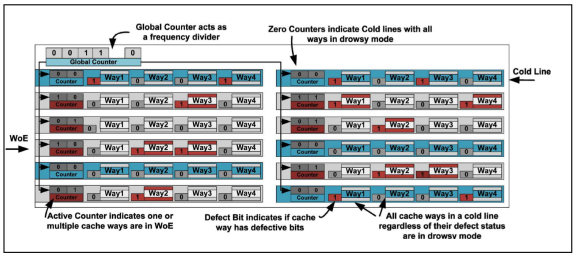

Figure 21: Top level view of the VTD-Cache, sets with non-zero set counters take part in WoE for which ways with defect bit set to 1 are on high voltage and the rest on low voltage [32, 33].

Fig. 21 illustrates the general idea of this architecture. In this Figure, each line represents a set that consists of four ways. Each set has its own Set Voltage Selector (SVS) which contains a dedicated bit counter (with $N = 1$ being the simplest realization and equal to what was proposed in the drowsy cache suggested in [6]). Upon access to a cache way in that set, the set is identified as being in the CWoE. This is being done with setting the countdown counter to a nonzero value. Each cache way has its own simple Way Voltage Selector (WVS), which is linked to a defect bit that indicates whether that specific way contains defective bits or not. If the SVS counter reaches zero, all WVSs that are associated with that SVS automatically shift the state of their cache way to data retention (Drowsy) mode. Otherwise based on their internal defect map bit, they supply their associated cache way from either V_{dd}^{Low} or V_{dd}^{High}. As suggested by Fig. 21, all cache ways within WoE (which are not cold/drowsy lines) and have their defect map bit set to one are sourced from a higher voltage and those with defect map bit of zero are supplied from V_{dd}^{Low}. A Set enters the CwoE by access to the set and exits the CWoE when its associated counter reaches 0. The set counter counts down when a count down signal (CDS) is sent from the global counter. The global counter acts as a cache access frequency divider and is shared by all sets. It is a cyclic counter that counts down and upon reaching 0 while being reset to its high value generates the CDS-signal that is fed to all SVSs.

Failure Analysis and Variation Aware Architecture [2]

In this paper, the authors are proposing an architectural solution to mitigate the effects of process variation. The solution is to analyze the failure and provides a variation aware architecture. The paper is analyzing the SRAM cell failures under process variation. The proposed architecture for high-performance applications is avoiding faulty cells by adaptively re-sizing the cache. The proposed solution has a negligible, 0.5%, area overhead. The energy overhead is at 1.8%, the obtained yield is at 93% while the CPU performance loss is an average 5.7%.

The paper is focusing on achieving better yield by increasing the number of blocks in a row. In conventional techniques such as [15] and [45] multiple cache blocks are placed in a single row to achieve lower delay and to decrease the area overhead and routing complexity. The optimum number of blocks being placed in one row is determined based on a cache size to minimize the cache access time. It is worth noting, that the increase in the number of blocks placed in a row would request in total energy increase. However, the main concern in designing the L1 cache is the cache access time and the reasonable increase in energy is acceptable if the access time is increased.

The paper [2] focuses on increasing the yield at the cost of a slight increase in energy overhead and cache access time. The proposed design assumes a 32K cache designed with four blocks in a row. The design increases the cache access time by only 2% while the energy overhead is increased by 7% compared to a design having two blocks in a row. In return, the yield is improved by 14%. The paper goes on to conclude that even in a larger cache four or more blocks in a row will be the optimum case and will not result in a drastic increase in the cache access time.

As disclosed by Fig. 22 the proposed cache architecture focuses on direct map L1 cache, however, the design is applicable to both direct and set-associative caches as well as different levels of cache hierarchy such as the L2 or L3 caches. By adaptively resizing the cache the architecture detects and replaces faulty cells under process variation. The architecture assumes that the cache is equipped with a Built-In Self Test (BIST) circuitry, which tests the entire cache and detects faulty cells based on the failure mechanisms. Since the number of faulty cells and their location changes depending on operating conditions (e.g., supply voltage, frequency), such tests are conducted whenever there is a change in operating conditions.

The proposed solution adaptively resizes the cache to circumvent faults. This solution is transparent to the processor and has negligible energy and area overhead. The experimental results on a 32 K L1 cache show redundant rows/columns and ECC results in only 52% and 77% yield, respectively. Hence, with the increase in process variations, the proposed scheme can be useful in future cache design to achieve high yield.

4.3 Architectural-Level, Error Correction:

In the remainder of this section, I will provide a more extensive description of some selected papers associated with Architectural-level, Error Correction technique to mitigate the effects of process variation.

Exploring Variation-Aware Fault-Tolerant Cache under Near-Threshold Computing [43]

The paper is purposing an innovative fault-tolerant cache architecture suitable for high error rates memories. The technique is evolved around a variation-aware skewed associative cache which redirects the faulty blocks to the error-free blocks. The approach is an architectural approach to overcome the effects of processor variations. The advantage of the suggested technique is that it has the least cache capacity waste by using all error-free blocks without the

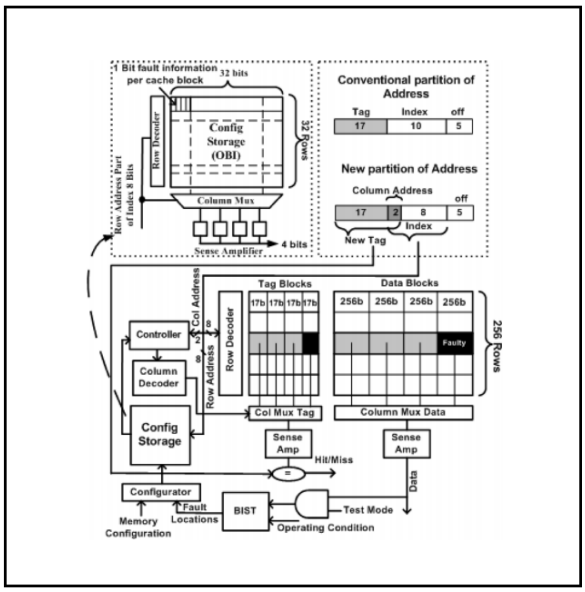

Figure 22: Architecture of 32 K process-tolerant cache [2].

need to disable any fault-free blocks to form a complete functional set. Furthermore, the skewed cache design minimizes the hardware overhead by avoiding the complex remapping from faulty blocks to the error-free blocks. The design has a high error rate tolerance. In the near-threshold region, the performance is improved while the cache miss rate is reduced. The paper evaluates it is cache design using Zsim, wherein the system has Out-of-Order cores, with each core having L1 data and L1 instruction caches. All the system cores are sharing one Last-Level Cache (LLC). The simulation has been performed using SPEC CPU2006 benchmarks. The performance effectiveness is analyzed in different environments starting at Super Threshold-voltage Computing (STC) moving to Low-voltage Computing (LVC) to Near-threshold Voltage Computer (NTC) where the supply voltage is aggressively scaled down to the near-threshold environment. A lower the supply voltage goes higher is the error rate and lower is the cache capacity. Furthermore, the paper is analyzing the bit-cell failure rate and block failure rate in different environments.

The area overhead in the proposed design is the fault-map which based on calculation presented by the paper is 1.3% across the board. The other factor affecting the total overhead is a hash unit to index the way, which causes around 0.31% area over the head. Considering all the aforementioned factors the worst-case scenario for the total overhead would be less than 6%. For this paper, the worst-case scenario has been used in the classification table.

As per access latency and delay penalty the design includes the delay on the hash function and the access delay to the fault map based on the index output from the hash function. Because the design does not need tag comparison the hit rate is similar to conventional set-associative design.

Fig. 23 shows the proposed fault-tolerant cache design. Upon the voltage scaling, Cache first generates the fault map by re-structures the position of blocks that contain faulty cells using the fault detection method. Next, the skewed-associative cache design takes the aggregated effect of processes variation and aging impact into the hash function for the address index upon receiving the memory access request. Third, the faulty blocks are circumvented based on the re-programmed fault map obtained in step one. Afterward, we could access the storage array directly if the cache hits, or cache replacement policy generates a candidate tree if the cache misses. The generated candidate tree finds the best replacement block.

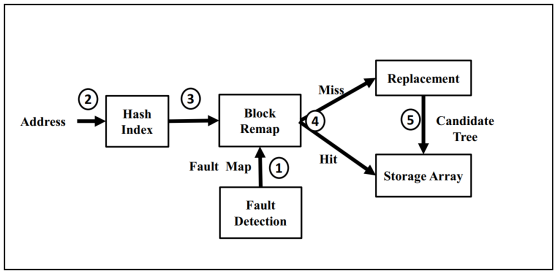

Figure 23: Architecture of the proposed fault-tolerant cache. [43].

In conclusion, Near-Threshold Voltage (NTV) Technology as purposed by the paper [43] provides a reliable and efficient SRAM cache structure by considering a compound effect of process variation and aging impact. The design provides more replacing candidates to maximize the utilization of error-free cache lines and avoids the complicated cache remapping procedure to minimize the hardware overheads.

5 SOFTWARE AND SYSTEM LEVEL:

In this section, we are reviewing papers that are proposing System or Software-Level solutions to mitigate the impact of memory process variation.

In this type of technique, the operation of the main memory is effected and modified. For example, to mitigate the effects of process variation in Phase Change Memory (PCM) main memory, the PCM is partitioned into a plurality of domains and voltage level is adjusted for different domains [47]. In other techniques, the memory pages are categorized into two different types of hot and cold based on the frequency of page updates. afterward, the hot-modified pages are correlated to regions that are positively impacted by the process variation, and the cold modified pages are correlated to regions that are negatively impacted by the process variation [47].

In the remainder of this section, I will provide a more extensive description of some selected papers associated with System or Software-level technique to mitigate the effects of process variation.

Adaptive Proactive Reconfiguration: A Technique for Process-Variability and Aging Aware SRAM Cache Design [30]

The paper [30] provides a new method to compensate for the effects of process variation for SRAM design by extending the cache lifetime using an adaptive strategy. The paper proposes an on-chip

monitoring technique to monitor the effects of aging in the SRAM cells. The results provided by the paper show the technique as a practical method to extend the cache lifetime up to 5X with around a 12% area overhead and negligible drop of performance.

The paper provides an architectural as well as a circuit-based solution to remedy the effects of process-variation and aging in SRAM cache designs. The paper [30] addresses the background of how the process variation and aging caused by Bias Temperature Instability (BTI) are two key reliability concerns in modern technologies. The effect and importance of BTI has been drastically increased when operating and integrating at the deep nanoscale. The SRAM is one of the main sections in integrated circuits prone to such type of deviations due to its utmost sensitivity to process variations.

Cache memories are normally designed based on reactive configuration principles. Where the memories are designed with a plurality of spare columns/rows to substitute the failing ones for yield improvement purposes. In this design, upon detection of a failure, the non-operational spare spaces are becoming operative. In another design, called proactive reconfiguration all the spare units are used in the normal operation of the memory system until the time of failure, as a result, the aging phenomena are shared among all the units, resulting in a longer lifetime of the memory.

The paper [30] recites an improved proactive reconfiguration technique in which results in a balanced aging distribution and larger lifetime extensions throughout the memory columns.

The paper's suggested technique considers the process variation and BTI wear-out of SRAM cells based on time and it is designed based on a non-homogeneous round-robin sequence between all the memory columns. The utilization of spare units provides a possibility of making a test in the memories to determine the status of memory columns.

It also allows defining distinct recovery times, which can be dynamically adapted to the respective V_T values. It starts with a test that measures the V_T value of each SRAM cell. Then, each of the tracked columns will be characterized by its highest V_T SRAM cell (the weakest cell in the column). These calculated values, determine the needed recovery time length (D_i) for each of the tracked memory columns.

The recovery duration calculation technique is based on the value determined by subtraction between the weakest and strongest SRAM cells V_T values [min is the value of the minimum (best) V_T column, and max is the value of the maximum (worst) V_T column] in the memory columns. First, we consider several V_T ranges in which we want to classify the memory columns among them. Then, we calculate the V_T, which is the difference value between the best and worst column V_TT values. Finally, the specific ranges are determined by the mentioned values. The columns are divided between these ranges such that the columns with higher V_T values will have longer recovery times.

This paper's technique has only the additional overhead of the monitoring circuits in comparison to the technique discussed by IBM. The implemented technique in the SRAM memory has a slight effect on memory cache performance. The frequency of reconfiguration process between columns is very low, which results in small performance loss at a switching time of a column to another, wherein the monitoring process of the recovery column can be a dc measurement.

This paper [30] discloses an adaptive proactive re-configuration technique for SRAM-based memory systems. It concludes that the adaptive proactive approach extends the system lifetime larger than the former (IBM) proactive approach. Also, the paper has proposed a specific monitoring circuit that tracks the time zero process variation and BTI aging of SRAM cells during operation.

Process Variation Aware Cache Leakage Management [25]

In this paper, the effects of within-die and die-to-die leakage variation for on-chip caches are analyzed. It purposes *way prioritization*, a manufacturing variation aware technique that minimizes cache leakage energy.

The paper employs a Six-Transistor (6-T) as the basic construction block for cache design as the basis for static power analysis and focuses on sub-threshold leakages.

The proposed technique as suggested by the paper [25] uses a two-phase approach as disclosed by Fig. 24. In the first phase, a statistical model for a cache is constructed based on organizational parameters such as capacity and block size as well as physical parameters such as geometric position on-chip and overall die size. In the second phase, by generating multiple random samples that have the statistical properties for computing total leakage the model Monte Carlo analysis is conducted.

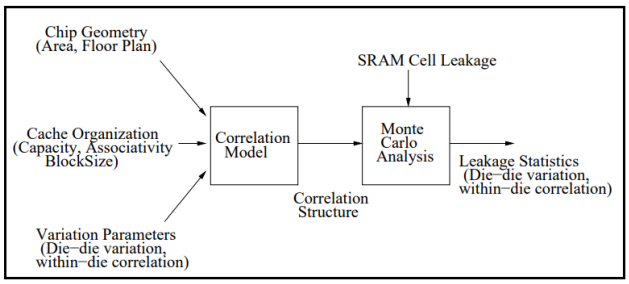

Figure 24: Leakage Variation Modeling approach used in [25].

Since conventional cache sizing strategies do not differentiate cache ways, although some portions of the cache will be leakier than others. In cases where one of the enabled ways happens to have a high overall leakage current, the conventional cache sizing strategies would not achieve good energy savings. To overcome the issue arising from conventional cache sizing strategies the paper is purposing *way prioritization* as a technique to enable the appropriate number of cache ways and select which subset of the cache array to make active. with circuit-level leakage and variation aware design techniques such as adaptive body-biasing and multiple v_t assignments to maximize leakage savings. The key hardware difference between a standard selective cache ways implementation and a way prioritized one is a set of hardware registers that identify the leaky cache ways and make cache sizing effective. Fig. 25 depicts the hardware differentiation and highlights the PRIORITY and DEGREE registers.

The DEGREE register supplements this information by tracking the absolute leakage of the corresponding physical way. When the

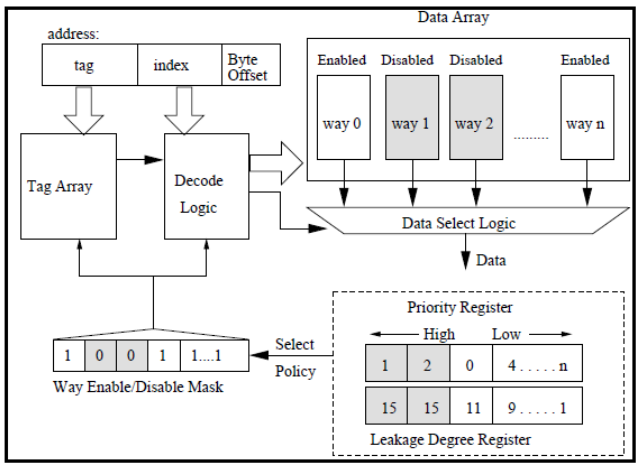

Figure 25: Hardware Organization for prioritized cache ways used in [25].

cache is being resized for a particular workload, these registers can be queried to determine how many ways should be enabled and which specific ways should be enabled. The measurements needed to populate the leakage registers can be collected off-line during the manufacturing test phase. Individual cache ways can be independently enabled as part of a BIST sequence while the rest of the processor is left idle. The leakage current for each way can be calculated from ammeter readings of total chip current draw. Collected data can be quantified, physical ways can be sorted by their leakage power, and the resultant information can be kept in non-volatile near-chip storage. At boot time, the PRIORITY and DEGREE registers can be configured based on the previously determined values.

The *Way Prioritization* allows the cache to be sized and configured on a per workload and per chip basis. Given knowledge of how application performance varies with increasing total cache size, we can either choose a sizing that minimizes power for a fixed performance level or we can target a more flexible power/performance optimization metric. For optimizations which allow a variable amount of performance degradation, we need to know the incremental energy cost of enabling each additional way. Different physical chips may have different total leakage or different ratios of leakage between ways. The DEGREE register tracks how much leakage energy each additional way contributes. When re-configuring the cache to minimize energy-delay, for example, the optimal value can be found by iterating through the PRIORITY and DEGREE registers.

The paper uses the Monte Carlo method to evaluate spatial leakage variation under several different cache configurations. In all cases, there is an assumption of a normal distribution on gate length intra-die variation and the 3σ value of the distribution was set to 9.4% of the nominal value. We see that there is a dramatic difference in cache leakage for regions chosen from different locations in the data array. The leakage ratio decreases rapidly as the region size increases. This is due to the fact that when the regions are small, there are many distant sections to choose from, increasing the chance that the regions do not have similar parameter sizes. As the regions grow, both the maximum and minimum leakage regions tend towards mean values, and the distance between the regions decreases. The second trend is that increasing cache sizes boost the max/min ratio. This is due to the fact that larger caches have a larger population of 6-T cells and hence longer "tails" on the distribution.

McPAT-PVT: Delay and Power Modeling Framework for FinFET Processor Architectures Under PVT Variations [20]

In this paper, the authors are addressing the problem posed by using FinFETs. The FinFETs have surfaced as a replacement for conventional CMOS due to their exceptional control of Short Channel effects (SCEs) and process scalability. However, FinFETs still have challenges such as process, supply voltage, and temperature (PVT) variations, which, in turn, results in a large increase in delay and leakage. The paper introduces both Circuit-level and an architectural-level solution based on FinFET processor to overcome the challenges of process, supply voltage, and temperature variations.

To address the ever-increasing demand for high-performance, high frequency the processor architectures and designs are moving toward increasing the processor tiles count. This has resulted in a significant boost in Chip Multiprocessor (CMP) power consumption. As the Transistors scale to deep sub-micrometer technology nodes, leakage is becoming an even more important part of power consumption. In CMOS implemented circuits the leakage in the active-mode has been estimated to be as high as 40% of the total power consumption even at the 90-nm technology node. To counteract the pointed out deficiency the FinFETs have been replacing bulk CMOS at the 22-nm node and beyond as they provide better scalability. The paper further has analyzed the characteristics of FinFETs as being non-planar double-gate devices. They provide a higher degree of control of short-channel effects compare to CMOS transistors and have smaller sub-threshold leakage. It is also possible to independently control the two-transistor gates of FinFETs. These design features and capabilities have resulted in the design of creative circuit modules, and dynamic power and thermal management schemes. Despite all these significant benefits of FinFETs, Process, Supply Voltage, and Temperature (PVT) variations still strike a big challenge to the designers. Process variations in FinFETs are mainly caused by lithographic constraints and difficulties in gate-work function engineering.

The paper [20] presents a Multicore Power, Area, and Timing (McPAT)-PVT, an integrated framework for the simulation of power, delay, as well as PVT variations of FinFET-based processors. McPAT-PVT uses a FinFET design library, consisting of logic and memory cells, to model circuit-level characteristics as well as their PVT variation trends. The integrated framework is based on macro-models, determined from highly precise TCAD device simulations that describe different functional units in a processor under PVT variations, making yield analysis for timing and power for processor components possible. McPAT-PVT can model both Shorted-Gate (SG) and Asymmetric-work-function Shorted-Gate (ASG), FinFET-based processors. Combining these macro-models with a FinFET-based CACTI-PVT cache model and an ORION-PVT on-chip network model, McPAT-PVT can simulate a delay and power consumption of all processor components under PVT variations.

The paper [20] analyzes the different time delays imposed based on different scenarios. For example, the paper looks into the variation in the time delay between SG-Mode and ASG-mode. Both modes have a good timing yield, all above 90%. It is worth saying that for an alpha-like processor and multi-core simulations based on Princeton Application Repository for Shared-Memory Computers benchmarks. The Result of simulations shows that the ASG FinFET-based processor implementation has 73× lower leakage power and 2.6× lower total power relative to the SG FinFET-based processor implementation for the same performance, with <1% area penalty.

In conclusion, this paper[20] is using McPAT-PVT built on top of a FinFET design performed processor-level simulations under PVT variations as well as real-traffic PARSEC simulations for varying number of cores. Results show that ASG-mode implementations can meet the same performance.

6 BENCHMARKING AND ANALYSIS CRITERIA

In this survey paper, the cache and memory design solutions are discussed and compared according to the following design and solution characteristics:

- **re-sizable:**Does the solutions allow the cache or the memory size to changes, when the number of faults increases. For example, does the solution support operation fo the cache in two voltages (high and low) which require fault tolerance at only one scaled voltage, or at multiple voltages, supporting various defect maps.
- **technology dependence:** is the solution processing technology dependent, or technology solutions in dependent. For example is the solutions applicable for specific technology (planer vs Finfet).
- **DVFS tolerance limit:** the extent of the tolerance of the cache/memory to change in voltage or frequency. Some solutions allow limited range of change in voltage, while others can support very wide range of change in voltage. The solutions are divided between small (less than 5% DVFS change), moderate (less than 15%), high (less than 25%), and aggressive (more than 25%).
- **Area overhead:** the area overhead for building the the fault tolerance solutions
- **Power overhead:** the power overhead for building the the fault tolerance solutions
- **Delay overhead:** The timing impact of the tolerance solutions at full voltage, when compared to a conventional memory/cache operated at the same Process, Voltage, and Temperature (PVT).
- **Fault Map:** does the solution require access to the fault map of the defective memories, and if it does, what is the defect map granularity (bank, row, word or byte level). Note that the smaller the granularity of defect map, the larger the area overhead of the design.
- **Adeptness**: is the fault map generated once at fabrication time, or there exist a mechanisim (such as MBIST) for the regeneration of the defect map. The memory solutions that support regeneration of the defect map and dynamically change the solutions to protect against such defects deemed more reliable and could also protect against aging induced defects that are not observed during the manufacturing test.

7 INNOVATIVE SOLUTIONS FOR PROCESS VARIATION MANAGEMENT

In this section, we are addressing the innovative solutions employed by the papers surveyed to mitigate the process variations effects.

7.1 Augmenting Cache with Additional Resources

The differences in features, parameters, and characteristics would be alleviated by assigning additional resources such as augmenting the cache with two auxiliary blocks.

The augmentation is assigned for example as suggested by [36]. In this approach, The Bit Lock Block (BLB) is a defect map for the cache and the Inquisitive Defect Cache (IDC) which is a small auxiliary cache operating as a dynamic redundancy to tolerate defects within the most recently visited locations is added to the cache.

7.2 Over-driving the Memory Array

In this approach, the entire memory array voltage is over-driven, which would result in increased power consumption.

7.2.1 Selective Over-driving. - To minimize the power consumption and effect on the entire memory array the paper [35] suggests selective wordline overdriving utilizing a small one step charge pump. In the approach of [35], a modified wordline driver peripheral device is used to allow selective wordline overdriving.

7.2.2 Body Biasing Techniques. - Body biasing is a circuit-level technique used to reduce the effect of process variation and the performance loss. In body biasing the V_{th} is modified at post-fabrication or during run-time to mitigate the process variation effects.

7.3 Decreasing the Workload of Process Variation affected segments

7.3.1 Decrease the workload. - In this approach to decrease the workload on the memory portions affected by process variation is to substitute the defected portions. As suggested by the paper [16, 34], the data that is destined to be written at a defective cache way is decomposed and relocated to a new location. Therefore, avoiding a write to a defective bit. In this approach, a Re-sizeable Data Composer Cache is utilized.

7.3.2 disabling the defected area. - In this approach as suggested by [2] the cache is adaptively resized to avoid the defected area and as a result, the yield is increased.

7.4 Providing Process Variation Aware Configuration Techniques

In this approach, the architecture is informed of process variation affected areas and manages the cache based on the knowledge of defected areas.

Title	Arch	Circuit	Overhead	Delay Pen	Cell-type	Re-sizable
Inquisitive Defect Cache (IDC) [36]	✓	✓	%7.3 1	1%	SRAM	
PV Aware SRAM/Cache for Aggressive... [35]	✓		4%	not reported	SRAM	
Re-sizable Data Composer Cache (RDC) [34]	✓		6.72%	0.91%	SRAM	✓
Variation Trained Drowsy Cache (VTD) [32]	✓		4%	1%	SRAM	✓
History and Variation Trained Cache (HVT-Cache) [33]	✓		4%	1%	SRAM	✓
Process Variation Aware Cache Leakage Management [25]		✓	25%	not reported	SRAM	✓
Failure Analysis and Variation Aware Architecture [2]	✓		0.5%	5.7%	SRAM	✓
Process Variation-Aware Adaptive Cache...[29]	✓	✓	not reported	not reported	SRAM	
Exploring Variation-Aware Fault Tolerant... [43]	✓		6%	negligible	SRAM	
Dynamic Cache Pooling [48]	✓	✓	1%	negligible	SRAM	✓
Working with Process Variation Aware Caches [27]		✓	10%	7.76%	SRAM	
Fine-Grained Fault Tolerance for PVA Cache [22]	✓		3%	10%	SRAM	
PV and Aging-Aware SRAM Cache Design [30]	✓	✓	12%	negligible	SRAM	
McPAT-PVT [20]		✓	1%	very small	FinFET	

Table 1: Classification of references analyzed in this survey

One way to obviate the effect of process variation as suggested by [32, 33], the architecture is to only applying a higher voltage to the cells, and other areas are operating at a lower voltage level. As a result, this approach is providing a low cost distributed supply voltage management. The dependent instructions are issued based on the forecasted latency of the associated load instruction.

Another way as suggested by [25] is to use way prioritization, that re-sizes caches to reduce the average and worst-case leakage power without compromising performance. As a result, it provides a leakage reduction strategy.

7.5 Bypassing Process Variation Using Instruction Scheduling Techniques

In this approach, the effect of process variation on manufactured cache sets is bypassed using different instruction scheduling techniques.

For example, the technique cited by [29] uses a set predictor and latency table, wherein the latency table for the cache sets can be determined by the March set performed during the functional testing of the memory [29].

8 CONCLUSIONS

In this short servery paper, I reviewed several of the fault-tolerant cache organizations proposed for tolerating a large number of process variation induced defects when the supplied voltage to the cache (that is fabricated at advanced geometry nodes) is reduced. The proposed solutions covered a range of techniques from circuit level to architectural and organization level solutions. Some of the covered techniques (such as Charge Pumping Word-line driver solution) apply to both memory and Cache organizations, while some others rely on the temporal and locality of access to the data stored in the cache to mitigate and hide the impact of defects while slowly downsizing the cache size.

REFERENCES

[1] Amit Agarwal, Bipul Chandra Paul, Hamid Mahmoodi, Animesh Datta, and Kaushik Roy. 2005. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13, 1 (2005), 27–38.

[2] A. Agarwal, B. C. Paul, S. Mukhopadhyay, and K. Roy. 2005. Process variation in embedded memories: failure analysis and variation aware architecture. *IEEE Journal of Solid-State Circuits* 40, 9 (Sep. 2005), 1804–1814. https://doi.org/10.1109/JSSC.2005.852159

[3] M. Anis. 2003. Subthreshold leakage current: challenges and solutions. In *Proceedings of the 12th IEEE International Conference on Fuzzy Systems (Cat. No.03CH37442)*. 77–80. https://doi.org/10.1109/ICM.2003.238359

[4] A. J. Bhavnagarwala, , and J. D. Meindl. 2001. The impact of intrinsic device fluctuations on CMOS SRAM cell stability. *IEEE Journal of Solid-State Circuits* 36, 4 (April 2001), 658–665. https://doi.org/10.1109/4.913744

[5] B. Cheng, S. Roy, and A. Asenov. 2007. The scalability of 8T-SRAM cells under the influence of intrinsic parameter fluctuations. In *ESSDERC 2007 - 37th European Solid State Device Research Conference*. 93–96. https://doi.org/10.1109/ESSDERC.2007.4430887

[6] K. Flautner, , S. Martin, D. Blaauw, and T. Mudge. 2002. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings 29th Annual International Symposium on Computer Architecture*. 148–157. https://doi.org/10.1109/ISCA.2002.1003572

[7] Siddharth Garg and Diana Marculescu. 2011. System-level leakage variability mitigation for mpsoc platforms using body-bias islands. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20, 12 (2011), 2289–2301.

[8] P. Ghafari, M. Anis, and M. Elmasry. 2007. Impact of technology scaling on leakage reduction techniques. In *2007 IEEE Northeast Workshop on Circuits and Systems*. 1405–1408. https://doi.org/10.1109/NEWCAS.2007.4488021

[9] Houman Homayoun, Mohammad Makhzan, and Alex Veidenbaum. 2008. Multiple Sleep Mode Leakage Control for Cache Peripheral Circuits in Embedded Processors. In *Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES '08)*. ACM, New York, NY, USA, 197–206. https://doi.org/10.1145/1450095.1450125

[10] H. Homayoun, M. Makhzan, and A. Veidenbaum. 2008. ZZ-HVS: Zig-zag horizontal and vertical sleep transistor sharing to reduce leakage power in on-chip SRAM peripheral circuits. In *2008 IEEE International Conference on Computer Design*. 699–706. https://doi.org/10.1109/ICCD.2008.4751937

[11] Houman Homayoun, Sudeep Pasricha, Mohammad Makhzan, and Alex Veidenbaum. 2008. Dynamic Register File Resizing and Frequency Scaling to Improve Embedded Processor Performance and Energy-delay Efficiency. In *Proceedings of the 45th Annual Design Automation Conference (DAC '08)*. ACM, New York, NY, USA, 68–71. https://doi.org/10.1145/1391469.1391488

[12] Houman Homayoun, Sudeep Pasricha, Mohammad Makhzan, and Alex Veidenbaum. 2008. Improving Performance and Reducing Energy-delay with Adaptive Resource Resizing for Out-of-order Embedded Processors. In *Proceedings of the 2008 ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '08)*. ACM, New York, NY, USA, 71–78. https://doi.org/10.1145/1375657.1375668

[13] Houman Homayoun, Sudeep Pasricha, Avesta Sasan (MA Makhzan), and Alex Veidenbaum. 2008. Improving performance and reducing energy-delay with adaptive resource resizing for out-of-order embedded processors. *ACM Sigplan Notices* 43, 7 (2008), 71–78.

[14] Houman Homayoun, Avesta Sasan, Jean-Luc Gaudiot, and Alex Veidenbaum. 2011. Reducing power in all major CAM and SRAM-based processor units via centralized, dynamic resource size management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19, 11 (2011), 2081–2094.

[15] Masashi Horiguchi. 1997. Redundancy techniques for high-density DRAMS. In *1997 Proceedings Second Annual IEEE International Conference on Innovative Systems in Silicon*. IEEE, 22–29.

[16] Michael Hübner and Cristina Silvano. 2015. *Near Threshold Computing: Technology, Methods and Applications*. Springer.

[17] Lei Jiang, Youtao Zhang, and Jun Yang. 2011. Enhancing phase change memory lifetime through fine-grained current regulation and voltage upscaling. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*. IEEE Press, 127–132.

[18] Ismail Kadayif, Mahir Turkcan, Seher Kiziltepe, and Ozcan Ozturk. 2013. Hardware/software approaches for reducing the process variation impact on instruction fetches. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 18, 4 (2013), 54.

[19] J. P. Kulkarni, , S. P. Park, and K. Roy. 2008. Process variation tolerant SRAM array for ultra low voltage applications. In *2008 45th ACM/IEEE Design Automation Conference*. 108–113. https://doi.org/10.1145/1391469.1391498

[20] C. Lee and N. K. Jha. 2014. FinCANON: A PVT-Aware Integrated Delay and Power Modeling Framework for FinFET-Based Caches and On-Chip Networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 5 (May 2014), 1150–1163. https://doi.org/10.1109/TVLSI.2013.2260569

[21] M. A. Lucente, C. H. Harris, and R. M. Muir. 1990. Memory system reliability improvement through associative cache redundancy. In *IEEE Proceedings of the Custom Integrated Circuits Conference*. 19.6/1–19.6/4. https://doi.org/10.1109/CICC.1990.124781

[22] T. Mahmood and S. Kim. 2010. Fine-Grained Fault Tolerance for Process Variation-Aware Caches. In *2010 IEEE Computer Society Annual Symposium on VLSI*. 46–51. https://doi.org/10.1109/ISVLSI.2010.57

[23] M. A. Makhzan, A. Khajeh, A. Eltawil, and F. Kurdahi. 2007. Limits on voltage scaling for caches utilizing fault tolerant techniques. In *2007 25th International Conference on Computer Design*. 488–495. https://doi.org/10.1109/ICCD.2007.4601943

[24] Jie Meng, Tiansheng Zhang, and Ayse K Coskun. 2013. Dynamic cache pooling for improving energy efficiency in 3D stacked multicore processors. In *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 210–215.

[25] K. Meng and R. Joseph. 2006. Process Variation Aware Cache Leakage Management. In *ISLPED'06 Proceedings of the 2006 International Symposium on Low Power Electronics and Design*. 262–267. https://doi.org/10.1145/1165573.1165636

[26] Sparsh Mittal. 2016. A survey of architectural techniques for managing process variation. *ACM Computing Surveys (CSUR)* 48, 4 (2016), 54.

[27] M. Mutyam and V. Narayanan. 2007. Working with Process Variation Aware Caches. In *2007 Design, Automation Test in Europe Conference Exhibition*. 1–6. https://doi.org/10.1109/DATE.2007.364450

[28] Madhu Mutyam and Vijaykrishnan Narayanan. 2007. Working with process variation aware caches. In *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 1–6.

[29] M. Mutyam, F. Wang, R. Krishnan, V. Narayanan, M. Kandemir, Y. Xie, and M. J. Irwin. 2009. Process-Variation-Aware Adaptive Cache Architecture and Management. *IEEE Trans. Comput.* 58, 7 (July 2009), 865–877. https://doi.org/10.1109/TC.2009.30

[30] P. Pouyan, E. Amat, and A. Rubio. 2015. Adaptive Proactive Reconfiguration: A Technique for Process-Variability- and Aging-Aware SRAM Cache Design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23, 9 (Sep. 2015), 1951–1955. https://doi.org/10.1109/TVLSI.2014.2355873

[31] Avesta Sasan. 2010. *Low power and process variation aware SRAM and Cache design fault tolerance in SRAM circuit, architecture and organization.* University of California, Irvine.

[32] Avesta Sasan, Kiarash Amiri, Houman Homayoun, Ahmed M Eltawil, and Fadi J Kurdahi. 2012. Variation trained drowsy cache (VTD-cache): A history trained variation aware drowsy cache for fine grain voltage scaling. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20, 4 (2012), 630–642.

[33] A. Sasan, H. Homayoun, K. Amiri, A. Eltawil, and F. Kudahi. 2012. History amp; Variation Trained Cache (HVT-Cache): A process variation aware and fine grain voltage scalable cache with active access history monitoring. In *Thirteenth International Symposium on Quality Electronic Design (ISQED)*. 498–505. https://doi.org/10.1109/ISQED.2012.6187540

[34] Avesta Sasan, Houman Homayoun, Ahmed Eltawil, and Fadi Kurdahi. 2009. 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*. ACM, 251–260.

[35] A. Sasan, H. Homayoun, A. Eltawil, and F. Kurdahi. 2009. Process Variation Aware SRAM/Cache for aggressive voltage-frequency scaling. In *2009 Design, Automation Test in Europe Conference Exhibition*. 911–916. https://doi.org/10.1109/DATE.2009.5090795

[36] A. Sasan, H. Homayoun, A. M. Eltawil, and F. Kurdahi. 2011. Inquisitive Defect Cache: A Means of Combating Manufacturing Induced Process Variation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19, 9 (Sep. 2011), 1597–1609. https://doi.org/10.1109/TVLSI.2010.2055589

[37] Avesta Sasan (MA Makhzan), Amin Khajeh, Ahmed Eltawil, and Fadi Kurdahi. 2009. A low power JPEG2000 encoder with iterative and fault tolerant error concealment. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17, 6 (2009), 827–837.

[38] Avesta Sasan (MA Mazkhan). 2006. *JPEG2000 Error Detection and Concealment.* Ph.D. Dissertation. University of California, Irvine.

[39] Philip P Shirvani and Edward J McCluskey. 1999. PADded cache: a new fault-tolerance technique for cache memories. In *Proceedings 17th IEEE VLSI Test Symposium (Cat. No. PR00146)*. IEEE, 440–445.

[40] G. S. Sohi. 1989. Cache memory organization to enhance the yield of high performance VLSI processors. *IEEE Trans. Comput.* 38, 4 (April 1989), 484–492. https://doi.org/10.1109/12.21141

[41] Abhishek Tiwari, Smruti R Sarangi, and Josep Torrellas. 2007. ReCycle:: pipeline adaptation to tolerate process variation. *ACM SIGARCH Computer Architecture News* 35, 2 (2007), 323–334.

[42] Jue Wang, Xiangyu Dong, and Yuan Xie. 2012. Point and discard: a hard-error-tolerant architecture for non-volatile last level caches. In *Proceedings of the 49th Annual Design Automation Conference*. 253–258.

[43] J. Wang, Y. Liu, W. Zhang, K. Lu, K. Qiu, X. Fu, and T. Li. 2016. Exploring Variation-Aware Fault-Tolerant Cache under Near-Threshold Computing. In *2016 45th International Conference on Parallel Processing (ICPP)*. 149–158. https://doi.org/10.1109/ICPP.2016.24

[44] V. Wang, K. Agarwal, S. Nassif, K. Nowka, and D. Markovic. 2008. A Design Model for Random Process Variability. In *9th International Symposium on Quality Electronic Design (isqed 2008)*. 734–737. https://doi.org/10.1109/ISQED.2008.4479829

[45] Don Weiss, John J Wuu, and Victor Chin. 2002. The on-chip 3-mb subarray-based third-level cache on an itanium microprocessor. *IEEE Journal of Solid-State Circuits* 37, 11 (2002), 1523–1529.

[46] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S. Lu. 2008. Trading off Cache Capacity for Reliability to Enable Low Voltage Operation. In *2008 International Symposium on Computer Architecture*. 203–214. https://doi.org/10.1109/ISCA.2008.22

[47] Wangyuan Zhang and Tao Li. 2009. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2–13.

[48] B. Zhao, Y. Du, J. Yang, and Y. Zhang. 2013. Process Variation-Aware Nonuniform Cache Management in a 3D Die-Stacked Multicore Processor. *IEEE Trans. Comput.* 62, 11 (Nov 2013), 2252–2265. https://doi.org/10.1109/TC.2012.129