Working with Process Variation Aware Caches*

Madhu Mutyam
International Institute of Information Technology, Hyderabad
Gachibowli, Hyderabad - 500032, India
mutyam@iiit.ac.in

Vijaykrishnan Narayanan
Pennsylvania State University
University Park, PA 16802, USA
vijay@cse.psu.edu

Abstract

Deep-submicron designs have to take care of process variation effects as variations in critical process parameters result in large variations in access latencies of hardware components. This is severe in the case of memory components as minimum sized transistors are used in their design.

In this work, by considering on-chip data caches, we study the effect of access latency variations on performance. We discuss performance losses due to the worst-case design, wherein the entire cache operates with the worst-case process variation delay, followed by process variation aware cache designs which work at set-level granularity. We then propose a technique called **block rearrangement** *to minimize performance loss incurred by a process variation aware cache which works at set-level granularity. Using block rearrangement technique, we rearrange the physical locations of cache blocks such that a cache set can have its "n" blocks (assuming a n-way set-associative cache) in multiple rows instead of a single row as in the case of a cache with conventional addressing scheme. By distributing blocks of a cache set over multiple sets, we minimize the number of sets being affected by process variation. We evaluate our technique using SPEC2000 CPU benchmarks and show that our technique achieves significant performance benefits over caches with conventional addressing scheme.*

1 Introduction

With continued reduction in technology feature sizes, the device parameters such as channel length, oxide thickness, random placement of dopants in channel, etc., are expected to exhibit significant variations. Variations in critical process parameters can result in large variations in access latencies and/or leakage energy of hardware components. With CMOS process technology moving into nanometer regime, the degree of variability encountered in critical parameters makes the worst-case design methodologies a non-viable option for future designs [5, 6, 26]. Process variation effects are very severe in memory circuits as memories are typically designed using minimum sized transistors for density reasons [21]. In order to reduce performance loss due to process variation, one can work with an adaptive design methodology which exploits the variability in memory access latency.

As our main focus is on on-chip data caches, we now discuss adaptive design methodologies for data caches. In order to reduce performance loss due to process variation, we can design a process variation aware cache which exploits access latency variations. Under such a variation aware design scenario, we predict access latency of a load instruction using a prediction technique [4, 10] and then issue all dependent instructions of the load instruction based on the predicted latency. If the prediction is correct, we obtain performance improvements due to early issue of dependent instructions. On the other hand, if the prediction is wrong, we replay all the dependent instructions to ensure correct execution. The granularity of the latency prediction in presence of variation is critical in influencing the performance benefits. For example, all blocks in a set of an associative cache can assumed to have same latency (determined by worst-case of all blocks in the set) or based on specific block corresponding to a particular way of a set in an associative cache. Latency prediction at way-level granularity has more potential to deal with variability without being constrained by the worst-case of blocks in other ways of a cache. However, current way-prediction techniques for data caches are not accurate enough as in instruction caches [22].

In this paper, by considering process variation aware data caches which work at set-level granularity, we propose *block rearrangement* technique to minimize performance loss due to access latency variations. Using our technique, we rearrange the physical locations of cache blocks to minimize the number of sets being affected by process variation.

The rest of this paper is organized as follows. The next section presents our technique. We validate our technique in

*This work was supported in part by grants from DST (India) and NSF (Career No. 0093085).

978-3-9810801-2-4/DATE07 © 2007 EDAA

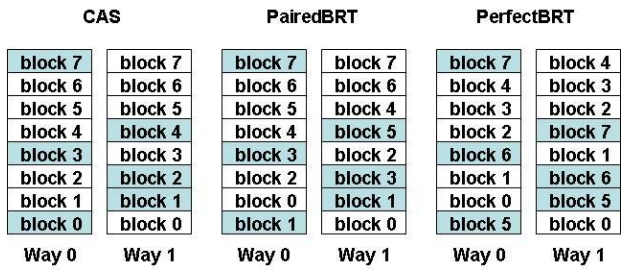

Figure 1. An illustration for cache block organization in CAS, PairedBRT, and PerfectBRT. Shaded portions indicate the blocks which are affected by process variation.

Section 3. Section 4 presents related work in the literature and Section 5 concludes the paper.

2 Block Rearrangement Technique

The idea in block rearrangement technique is to rearrange the physical locations of blocks to minimize the number of sets having both low and high latency blocks. By considering rearrangement of cache blocks between a pair of cache sets and among all cache sets, we propose two techniques, namely, *paired block rearrangement technique* (PairedBRT) and *perfect block rearrangement technique* (PerfectBRT), respectively. In PairedBRT, we consider two adjacent sets as a group and perform block rearrangement within the group. In PerfectBRT, block of a set can be rearranged to any position in the corresponding cache way. For a n-way set-associative cache, the block rearrangement *range* in PairedBRT and PerfectBRT is 2 and n, respectively. In both techniques, blocks in a cache way are rearranged in such a way that the high latency blocks are moved to the bottom of a group or an entire cache way. Note that PairedBRT is simple to implement, whereas PerfectBRT is very effective in performance.

We illustrate our techniques using a 2-way set associative cache which has 8 sets (numbered from 0 to 7) as shown in Figure 1. Blocks of a set i are represented as block i in both way 0 and way 1. Shaded blocks in Figure 1 represent the process variation affected blocks, and hence they take high access latency. 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 1, 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. From PairedBRT part of Figure 1, as blocks which belong to sets $2i$ and $2i+1$, $0 \leq i \leq 3$, are rearranged, it is clear that *four* sets, i.e., sets 0, 2, 4, and 6, take low latency and the remaining sets take high latency.

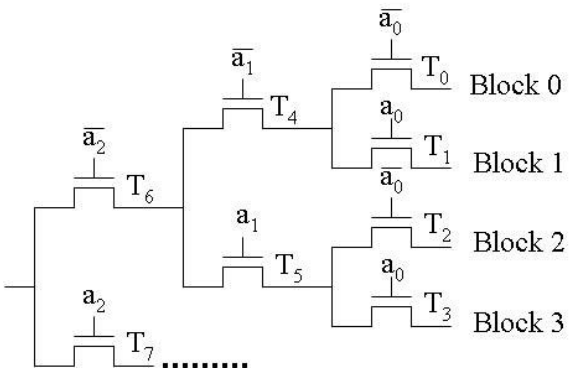

Figure 2. Decoder configuration in CAS.

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 Figure 1 shows that *five* cache sets take low latency.

In general, using PerfectBRT, the minimum number of sets which take high access latency in a n-way set associative cache becomes k, where

$$k = max\{k_i \mid k_i \text{ is the number of high access latency blocks in way } i, 0 \leq i \leq n-1\}.$$

In order to rearrange cache blocks, we consider a *programmable address decoder* which is similar to the one proposed in [24]. Address decoder in [24] is programmed in a way to disable faulty cache blocks and re-map any references to the faulty blocks to the good blocks. In our techniques, we consider an address decoder which is programmed to rearrange block positions. In order to program the address decoder, we make use of the March test [7] which can distinguish the low and high latency cache blocks. The March test is employed during the functional testing of memory components. It involves applying a specific sequence of operations that read and write values of 0s and 1s to different locations in a memory. The March test can also identify the effects of process variation such as destructive readout failures [8]. So, cache blocks can be characterized as either low latency blocks or high latency blocks through such a test that captures access time failures before the operational phase of a microprocessor and rearrange the cache blocks using our techniques. If cache block i is characterized as a low latency block, we consider f_i=0, otherwise, f_i=1. We pass this block-wise information to the address decoder to perform required rearrangement.

In this paper, we consider a 4-way set associative cache with one decoder for each way. Figure 2 shows a part of the last three stages of a decoder used in CAS. Though this type

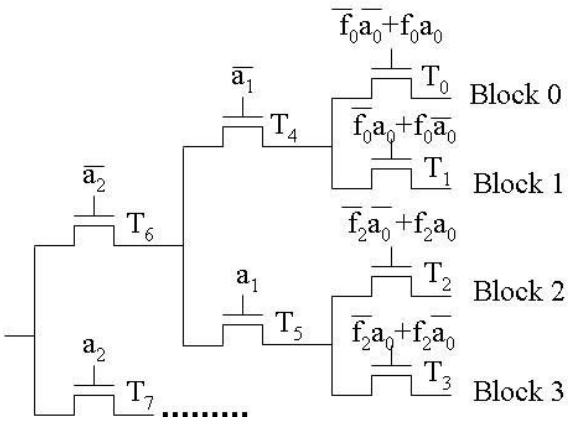

Figure 3. Decoder configuration in Paired-BRT. Here f_i indicates whether or not the ith block is affected by process variation.

of decoder may not be used in performance-critical circuits, we consider it here because it makes illustration of our technique easier. Note that the same procedure can be applied to gates instead of pass transistors as shown in [24]. Based on the values of a_0, a_1, and a_2, one of the eight blocks can be selected. The modified decoder which is used in PairedBRT is shown in Figure 3. PairedBRT rearranges blocks of adjacent sets. The block characterization values f_i, which are obtained from the March test, are used in combination with the values a_0 and $\overline{a_0}$ to modify 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. Though we have not calculated the area overhead due to programmable address decoder, it is expected to be small as demonstrated for a similar design in [24].

3 Experimental Validation

3.1 Experimental Setup

We validate our technique by simulating 19 SPEC2000 CPU benchmarks [2] using the Simplescalar 3.0 simulator [1]. For each benchmark, we fast forward 100 million instructions and then simulate next 300 million instructions. The baseline processor configuration is given in Table 1. We assume two-stage pipelined data cache. In the ideal scenario where there is no process variation, the data cache access takes two cycle latency. On the other hand, if process variation influences the pipeline stages of cache access, the delay of either pipeline stage can exceed that of the nominal cycle time. Note that clock period for a stage is determined by the worst-case pipeline stage hence even if process variation increases the latency of one of the cache pipeline stages to

Issue width	8 instructions/cycle (out-of-order)
RUU size	128 instructions
LSQ size	64 instructions
L1 data cache	64KB, 4-way (LRU), 64B blocks, 2 cycle latency (no process variation), 4 cycle latency (with process variation), 25% of cache sets have 4 cycle latency, and the remaining have 2 cycle latency
L1 Inst. cache	64KB, 4-way (LRU) 32B blocks, 1 cycle latency
L2 cache	Unified, 512KB, 8-way (LRU) 128B blocks, 12-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

Table 1. Base processor configuration.

two cycles, this results in four cycle latency for a two stage pipeline. In the baseline configuration we assume 25% of cache sets are affected by process variation and hence they take 4 cycle latency and the remaining cache sets take two cycle latency. As part of sensitivity analysis, we conduct experiments by assuming different percentages of cache sets are being affected by process variation. In our experiments we also consider the worst-case design scenario where the cycle time of the entire pipeline is doubled.

3.2 Experimental Mechanism

In order to exploit access latency variations, we predict the latency of a given load operation ahead of time by using a prediction technique. In this paper, as we work at set-level granularity, we consider a *2-delta stride-based address predictor* [10] for set prediction. In our experiments, we use a predictor with $16K$ entries. We also consider a *latency table* which maintains the access latencies at a cache set granularity (assuming that each cache set has a single latency which is the largest latency among all its ways). The latency table can be initialized through the March test [7] that captures access time failures before the operational phase of a microprocessor. The memory locations where access failures can be removed through operating at a slower frequency are provided a bit value of 1 in the latency table, whereas those that can complete within a single cycle with no access failures are initialized to 0. Note that the March test also marks other bits to indicate permanent failures. This includes timing failures that cannot be compensated by even doubling access latency. It also needs to be mentioned that the latency table itself may be subject to process variations. However, due to the reduced wordline capacitance offered by a single cell used in our look-up table, even with process variation,

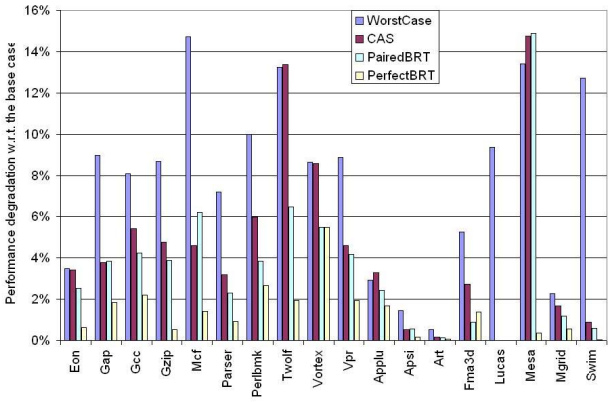

Figure 4. Benchmark-wise IPC degradation for different techniques w.r.t. the base case. Note that here we assumed 25% of cache sets are being affected by process variation.

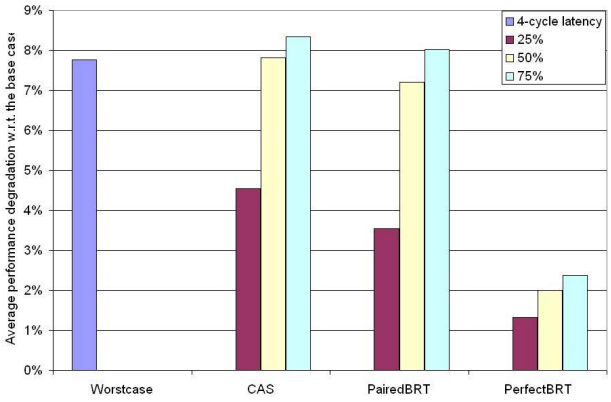

Figure 5. Average IPC degradation for different techniques w.r.t. the base case.

the cycle time never exceeds the sum of the decoder and word line delay of the cache.

Given the static address of a load instruction, we predict the cache set address of a data to be accessed by using the prediction technique. The latency table is indexed using the predicted set address to obtain the predicted cache access latency so that the dependent instructions can be issued accordingly. It is important to note that, as there are multiple pipeline stages between decode and dispatch, our set prediction and latency table access for the predicted latency do not fall into the critical path. If the set prediction is correct, we obtain performance gains. On the other hand, even if the set prediction is wrong, we may have correct latency prediction as two different sets can have same access latency. So, only when both the set and latency predictions are wrong we incur performance penalty as all dependent instructions are replayed. We replay instructions using the instruction-based selective replay technique [15].

3.3 Experimental Results

We consider five scenarios which include a cache without process variation so that each cache access takes 2 cycle latency (base case), a pessimistic case where each cache access takes 4 cycle latency (Worst-case), prediction with conventional addressing scheme (CAS), prediction with paired block rearrangement (PairedBRT), and prediction with perfect block rearrangement (PerfectBRT).

Figure 4 shows benchmark-wise IPC degradation for different techniques w.r.t. the base case. Note that here we assume 25% of cache sets take 4 cycle latency for data cache access and the remaining sets take 2 cycle latency. Performance degradation of the worst-case design w.r.t. the base case is very significant and it ranges from 0.52% ("Art") to 14.71% ("Mcf"). Even though CAS technique achieves better performance across different benchmarks (except "Applu", "Twolf", and "Mesa") as compared to the worst-case design, its performance degradation w.r.t. the base case ranges from 0.17% ("Art") to 14.76% ("Mesa"). In case of "Applu", "Twolf", and "Mesa", the prediction accuracy is very low so that we incur large number of instruction replays and hence performance loss. Generally, when we apply prediction based techniques on a data cache with access latency variations, performance of these techniques depends on factors such as *which set* (either a low latency set or a high latency set) a load instruction is accessing and *how many times* it is accessing the set. Further more, our block rearrangement techniques rearranges blocks so that a low access latency set in CAS (for example, set 6 in CAS part of Figure 1) may have high access latency after applying PairedBRT (for example, set 6 in PairedBRT part of Figure 1) which may result in better performance in CAS as compared to the block rearrangement techniques. This fact is evident from the results of benchmarks "Gap", "Mcf", and "Mesa". Except for these three benchmarks, PairedBRT technique achieves significant performance benefits across different benchmarks as compared to CAS. As PerfectBRT can rearrange blocks any where, it reduces performance penalty significantly. In the case of "Mesa", PerfectBRT incurs performance penalty of 0.37%, whereas both CAS and PairedBRT incur more than 14% performance penalty.

Figure 5 shows average IPC degradation across all benchmarks w.r.t. the base case for different percentages of cache sets being affected by process variation. The worst-case design can incur a performance penalty of 7.76%. As long as the percentage of cache sets being affected by process variation is low, CAS can perform better than the worst-case design. There is a huge difference in performance degradation values of CAS when the percentage of

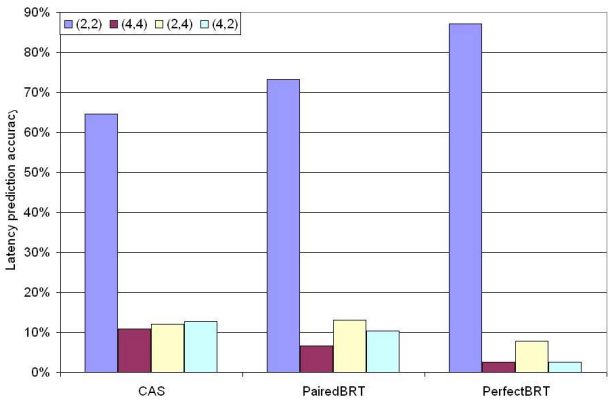

Figure 6. Latency prediction accuracy in different cache organizations.

cache sets being affected by process variation is changed from 25% to 50%, but from 50% to 75% variation, the difference is very small. The reasons are two fold: prediction accuracy is high for both 25% and 75% variation cases as either low latency group or high latency group dominates, whereas both low and high latency groups have equal probability in the 50% variation case so that the prediction accuracy is low; as variation percentage increases, the number of low latency cache sets decreases. If the prediction accuracy is low, mis-prediction rate can increase which in turn increases the number of replay instances and hence we incur significant performance loss. From the figure we see that CAS is not effective w.r.t. the worst-case as variation percentage is 50% or more. On the other hand, PairedBRT can produce better results as compared to CAS. Finally, PerfectBRT brings the performance penalty close to 2%.

We know that in case of set misprediction, the latency prediction can still be correct for the cache set accesses. The probability of this correctness is influenced by the number of bits and locations that vary in the set prediction and the rearrangement technique. In CAS, both low and high latency cache sets can be distributed across the entire cache, whereas in PerfectBRT, all low latency blocks are moved to one end of a cache way and all high latency blocks are moved to the other end of the same cache way. So, in CAS, even if the actual set address differs in one bit position w.r.t. the predicted set address, we may incur a latency misprediction, whereas in PerfectBRT, as long as the predicted set address is within a range of the actual set address, we get a latency hit. Figure 6 shows the average latency prediction accuracy across all benchmarks in CAS, PairedBRT, and PerfectBRT, respectively. In the figure, notation (a, p) indicates that a is the actual latency and p is the predicted latency. We consider latency prediction as *hit* if the predicted latency is greater than or equal to the actual latency. With respect to the worst-case design, performance improves in case of $(2, 2)$, whereas there is no change in performance in case of $(4, 4)$ and $(2, 4)$. In case of $(4, 2)$, we incur performance penalty as dependent instructions are replayed. From the figure, it is clear that PerfectBRT improves the latency prediction accuracy. As block rearrangement in PairedBRT is limited to adjacent sets, its prediction accuracy is less than that of PerfectBRT but it is more than that of CAS.

Note that the results presented in the paper are based on the prediction accuracy of *2-delta stride based predictor*. Any high accuracy set-predictor can improve the latency prediction accuracy which inturn helps in further reducing the performance penalty due to process variation.

4 Related Work

It has been shown that both inter-die [20] and intra-die [19] process variations have significant impact on both performance and power consumption of a chip [5, 6, 26]. The impact of process variation on memory has been analyzed by previous works [3, 8, 18]. Analysis of the emerging SRAM failure mechanisms due to process variations is made in [8]. In [3] SRAM cell failures under process variation is analyzed and a variation-aware cache architecture suitable for high performance applications is proposed. The proposed architecture adaptively resizes the cache to avoid faulty cells, thereby improves yield. For a small size embedded SRAM, as the decoder and wordline drivers are responsible for the majority of the delay and energy variation, techniques for variable tapered buffer design for embedded SRAM decoder are proposed in [21], so that the decoder configuration can be changed at run-time. There have also been recent efforts [11, 12] that deal with process variation in other components of a processor.

As load latencies are unpredictable, several techniques have been proposed to minimize the effects of unpredictable load latencies. Prediction techniques predicts the load addresses in the front end of the pipeline and accesses data cache speculatively. Value prediction [14, 23] predicts the values that would be brought by load operations early in the pipeline. Memory dependence prediction [9, 16] predicts dependence between a load and prior stores to reduce load issue delay and improves performance. Transient value cache [17] captures recently-used data and employs memory dependence prediction to steer memory accesses. Load reuse [25] compares the operand values of a previous dynamic load instruction stored in its reuse table with the values read from the register file of the current dynamic load.

Earlier works indicate that mechanisms related to our techniques are proposed for different purposes. A scheme called *block permutation* is proposed [13] in the context of power density minimization. It permutes cache blocks to maximize the distance between blocks with consecutive

addresses. As distance between consecutively addressed blocks increases, the area of a working set increases and hence power density minimizes. A technique called *padded cache* is proposed [24] for providing fault-toleranance to cache memories. In order to provide fault-tolerance, they proposed a special *programmable address decoder* which can disable faulty blocks to re-map their references to good blocks. Our technique is different from these two techniques in the following ways: 1) unlike the static rearrangement of cache blocks as suggested in [13], our technique rearranges cache blocks based on whether or not the blocks are affected by process variation; 2) unlike disabling cache blocks as suggested in [24], the programmable address decoder in our technique swaps different blocks.

5 Conclusion

In this paper, by considering a process variation aware data cache design which works at set-level granularity, we proposed block rearrangement technique to minimize performance penalty due to access latency variations in data caches. By validating our technique using SPEC2000 CPU benchmarks, we showed that our technique can significantly reduce performance penalty incurred by a conventional addressing scheme. As future work, we are planning to exploit block rearrangement concept to reduce leakage energy overhead due to process variation. We are also interested to see the effect of block rearrangement technique on conventional load address prediction techniques.

References

[1] SimpleScalar toolset. http://www.simplescalar.com.

[2] SPEC 2000 Benchmark. http://www.spec.org.

[3] A. Agarwal *et. al.*, "Process variation in embedded memories: failure analysis and variation aware architecture", *IEEE J. of Solid-State Circuits*, 40(9), 2005, pp.1804-1814.

[4] M. Bekerman *et. al.*, "Early load address resolution via register tracking", *ISCA*, 2000, pp. 306-315.

[5] S. Borkar *et. al.*, "Parameter variations and impact on circuits and microarchitecture", *DAC*, 2003.

[6] K. Bowman *et. al.*, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration", *IEEE J. of Solid-State Circuits*, 2002, 37(2), pp.183-190.

[7] M. L. Bushnell and V. D. Agarwal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*, Kluwer, 2000.

[8] Q. Chen *et. al.*, "Modeling and testing of SRAM for new failure mechanisms due to process variations in nanoscale CMOS", *VLSI Testing Symposium*, 2005, pp.53-59.

[9] G. Chrysos and J. Emer, "Memory dependence prediction using store sets", *ISCA*, 1998, pp.142-153.

[10] R.J. Eickemeyer and S. Vassiliadis, "A load instruction unit for pipelined processors", *IBM J. of Research and Development*, 1993, 3, pp. 547-564.

[11] D. Ernst *et. al.*, "Razor: circuit-level correction of timing errors for low-power operation", *IEEE Micro*, 2004, 24(6), pp.10-20.

[12] N.S.Kim *et. al.*, "Total power-optimal pipelining and parallel processing under process variations in nanometer technology", *ICCAD*, 2005, pp. 535-540.

[13] J.C. Ku *et. al.*, "Thermal management of on-chip caches through power density minimization", *MICRO*, 2005.

[14] M.H. Lipasti *et. al.*, "Value locality and load value prediction", *ASPLOS*, 1996, pp.54-61.

[15] G. Memik *et. al.*, "Precise instruction scheduling", *J. of Instruction-Level Parallelism*, 2005, pp. 1-29.

[16] A. Moshovos *et. al.*, "Dynamic speculation and synchronization of data dependences", *ISCA*, 1997, pp.181-193.

[17] A. Moshovos and G.S. Sohi, "Streamlining inter-operation memory communication via data dependence prediction", *MICRO*, 1997, pp.235-245.

[18] S. Mukhopadhyay *et. al.*, "Modeling and estimation of failure probability due to parameter variations in nanoscale SRAMs for yield enhancement", *Symposium on VLSI Circuits*, 2004, pp.789-796.

[19] S. Nassif, "Within Chip variability analysis", *IEEE IEDM conference*, 1998, pp.283-286.

[20] S. Nassif, "Modeling and analysis of manufacturing variations", *CICC*, 2001, pp.223-228.

[21] A. Papanikolaou *et. al.*, "A system-level methodology for fully compensating process variability impact of memory organizations in periodic applications", *CODES+ISSS*, 2005, pp.117-122.

[22] M.D. Powell *et. al.*, "Reducing set-associative cache energy via way-prediction and selective direct-mapping", *MICRO*, 2001.

[23] Y. Sazeides and J.E. Smith, "The predictability of data values", *ISCA*, 1997, pp.248-258.

[24] P.P. Shirvani and E.J. McClusky. "PADded cache: a new fault-tolerant technique for cache memories", *VTS*, 1999, pp. 440-445.

[25] A. Sodani and G. Sohi, "Dynamic instruction reuse", *ISCA*, July 1997, pp.194-205.

[26] P. Zuchowski *et. al.*, "Process and environmental variation impacts on ASIC timing", *DAC*, 2005.