# Block Remap with Turnoff: A Variation-Tolerant Cache Design Technique *

Mohammed Abid Hussain
Center for VLSI and Embedded System Technologies
International Institute of Information Technology - Hyderabad
Hyderabad - 500032, India
abid@research.iiit.ac.in

Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai - 600036, India
madhu@cse.iitm.ac.in

**Abstract— With reducing feature size, the effects of process variations are becoming more and more predominant. Memory components such as on-chip caches are more susceptible to such variations because of high density and small sized transistors present in them. Process variations can result in high access latency and leakage energy dissipation. This may lead to a functionally correct chip being rejected, resulting in reduced chip yield.**

**In this paper, by considering a process variation affected on-chip data cache, we first analyze performance loss due to worst-case design techniques such as accessing the entire cache with the worst-case access latency or turning off the process variation affected cache blocks, and show that the worst-case design techniques result in significant performance loss and/or high leakage energy. Then by exploiting the fact that not all applications require full associativity at set-level, we propose a variation-tolerant design technique, namely, block remap with turnoff (BRT), to minimize performance loss and leakage energy consumption. In BRT technique we selectively turnoff few blocks after rearranging them in such a way that all sets get almost equal number of process variation affected blocks. By turning off process variation affected blocks of a set, leakage energy can be minimized and the set can be accessed with low latency at the cost of reduced set associativity. We validate our technique by running SPEC2000 CPU benchmark-suite on Simplescalar simulator and show that our technique significantly reduces the performance loss and leakage energy consumption due to process variations.**

## I. Introduction

In the never ending pursuit of making a circuit faster and denser, more and more transistors are being placed on a single chip by reducing the feature size to as small as possible. With reduction in feature size, the control on the quality of the transistor being manufactured becomes difficult [7, 14, 19]. This results in variation of device parameters such as channel length, oxide thickness, threshold voltage, etc. Process variations are due to manufacturing phenomena and manifest themselves as *die-to-die* (DTD) and *with-in-die* (WID) variations. WID variation can be further divided into *random* and *systematic* variations, where random variations are small changes from transistor to transistor and systematic variations exhibit spatial correlations. These variations result in varying power and performance of circuits. For instance, process variations in 130nm technology cause a 30% variation in the maximum allowable frequency of operation and a fivefold increase in leakage power [12].

Process variations are severe in memory components as minimum sized transistors are used for density reasons [11]. In memory components, the critical path delay is mainly dictated by the memory sensing operations and the variable sense current produced by the minimum sized transistors causes a large timing variations of output data arrival [4]. Earlier the process variation affected circuits were dealt by assuming worst-case design scenario, but in the advanced process technologies as the degree of variability in the critical parameters has increased, worst-case design methodologies have become a non-viable option [6].

In this paper, by considering set-level granularity for load latency prediction, we propose an adaptive design technique, namely, *block remap with turnoff* (BRT), to minimize performance loss and leakage energy consumption due to process variations. Our technique exploits the fact that not all applications require full associativity at set-level [2]. By exploiting this fact an energy efficient cache design is proposed in [2], where an application specific associativity is determined so that some of the ways of a $n$-way set-associative cache are turned off. In our technique, we first spread the process variation affected blocks across the entire cache by applying a remap technique and then selectively turnoff the process variation affected blocks in such a way that at least one block per set is active. Note that we do not turnoff blocks of a set if all the blocks of the set are affected by process variations. We spread process variation affected blocks in such a way that all sets get almost equal number of process variation affected blocks. By turning off process variation affected blocks of a set, leakage energy can be minimized and the set can be accessed with low latency at the cost of reduced set associativity. We validate our technique by running SPEC2000 CPU benchmark-suite on Simplescalar simulator and show that, for a cache with 50% sets affected, our technique significantly reduces the performance loss to less than 1% of the base case and leakage energy consumption by about 20% of the worst case.

The rest of the paper is organized as follows. Section II presents related work. Section III describes how to work with process variation affected data caches. We present our technique in Section IV and validate it in Section V. Finally, we conclude the paper in Section VI.

*This work was supported in part by grant from Department of Science and Technology (DST), India, Project No. SR/S3/EECE/80/2006.

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

## II. Related work

As process variations significantly affect performance and leakage power consumption, several circuit-level and architectural-level techniques have been proposed in literature to mitigate the effects of process variations. We now briefly discuss some of the existing techniques.

It is shown that WID [25] and DTD [26] variations have significant impact on performance and power consumption of a chip [6, 7, 35]. SRAM cell failures under process variations are analyzed in [1, 10] and a technique to adaptively re-size the cache to eliminate faulty cells and hence to improve chip yield is proposed in [1]. In [34], adaptive body biasing is used to minimize the effects of process variations on maximum operating frequency and leakage energy dissipated. Parameter variations effect on system performance is discussed in [6] and body bias control techniques are proposed to reduce the effect of variations on circuits and improve chip yield. A gate sizing algorithm is developed to improve yield and also an algorithm for determining the optimal bin boundaries to obtain maximum benefit of using frequency binning is proposed in [12]. Limit on using forward body biasing to make the circuit more robust against variations in the threshold voltage is studied in [22]. By considering a chip with random and spatial variations, leakage energy dissipated in the chip is analyzed in [9] and a technique is proposed to predict the probability distribution function of the leakage energy dissipated in the chip.

In order to improve yield of a process variation affected cache, yield-aware cache architecture techniques are proposed in [27], which consist of turning off of cache ways, cache sets, or exploiting variable access latency by providing special buffers with functional units to deal with load instructions stored in process variation affected blocks. In [19], WID and DTD variations for on-chip caches are modeled and a technique called *way prioritization* is proposed to minimize cache leakage energy.

Several techniques have been proposed to predict load latencies, which include *value prediction* [16, 29], *transient value cache* [20], and *load reuse* [32]. All these techniques aim at minimizing the effects of unpredictable load latencies and improve system performance.

Some of the mechanisms related to our techniques are proposed in literature for different purposes. A scheme called *block permutation* [15] is proposed for power density minimization by permuting cache blocks to maximize the distance between blocks with consecutive addresses, which in turn maximizes the area of a working set, so that power density minimizes. In [18] the cache is downsized from its maximum capacity to counter the effects of process variation. To achieve this, on every cache access the effective address is boolean ANDed with the set-mask to produce the correct set index. A technique called *padded cache* [30] is proposed for providing fault-tolerance to cache memories. In padded cache technique, a special *programmable address decoder* is used to disable faulty blocks and remap their references to good blocks. In [21], using programmable address decoder, blocks of two different sets are rearranged in order to minimize the number of sets having process variation affected blocks. In [21], in a pair of sets all the process variation affected blocks are moved to one set, leaving the other set with clean (i.e., no process variation) blocks. Our technique is different from these techniques in the following ways: 1) unlike the static rearrangement of cache blocks as suggested in [15], 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 [30], the mapping scheme in our technique swaps different blocks; 3) unlike moving all process variation affected blocks to one set, of a pair of sets, as in [21], our mapping scheme spreads process variation affected blocks across all sets, in such a way that, all the sets get almost equal number of process variation affected blocks.

## III. Working with Process Variation Affected Data Caches

In this paper we consider a process variation affected L1 data cache and assume the cache access to be pipelined in two stages. For a cache without process variations, each pipeline stage takes 1 cycle latency, hence the access latency becomes 2 cycles. Due to process variations, the delay of either pipeline stage can exceed that of the nominal cycle time. As clock period for a stage is determined by the worst-case pipeline stage, hence even if the latency of one of the cache pipeline stages is increased to 2 cycles due to process variations, the access latency becomes 4 cycles for a 2-stage pipelined cache. Thus, due to process variations, different blocks of a cache can be accessed with different latencies, which in turn make the cache as a non-uniform access latency cache. In order to work with non-uniform access latency caches, one can use worst-case design techniques such as accessing the cache with the worst-case access latency (i.e., high latency) or turning off the process variation affected blocks and accessing the remaining cache blocks with the nominal latency (i.e., low latency). Accessing the entire cache with high latency incurs significant performance penalty when only few blocks of the cache are affected by process variations, whereas turning off the process variation affected cache blocks results in significant performance loss when most of the blocks of the cache are affected by process variations due to reduced size of the cache.

In order to work with non-uniform access latency data caches, one can use adaptive design methodology (as described in [21]) in place of worst-case design techniques to minimize performance loss. Under the adaptive design methodology, the access latency of a load instruction is predicted using a prediction technique [5, 13] so that based on the predicted latency all dependent instructions of the load instruction are issued. If the prediction is correct, performance improvement is achieved due to early issue of dependent instructions, whereas all the dependent instructions are replayed if the prediction is wrong to ensure correct execution. Note that the granularity of the latency prediction in non-uniform access latency cache is critical in influencing the performance benefits. One can work at set-level granularity by assuming that all blocks in a set of an associative cache take same latency (determined by the worst-case of all blocks in the set) or at way-level granularity by considering latency based on specific block corresponding to a particular way of a set in an associative cache. Though latency prediction at way-level granularity has more potential to deal with non-uniform access latency caches, without being con-

| Block Index | Mapping | | |
|---|---|---|---|
| | 001 | 010 | 100 |
| 000 | 001 | 010 | 100 |
| 001 | 000 | 011 | 101 |
| 010 | 011 | 000 | 110 |
| 011 | 010 | 001 | 111 |
| 100 | 101 | 110 | 000 |
| 101 | 100 | 111 | 001 |
| 110 | 111 | 100 | 010 |
| 111 | 110 | 101 | 011 |

TABLE I
ILLUSTRATION OF MAPPING MECHANISM USED IN BRT TECHNIQUE.

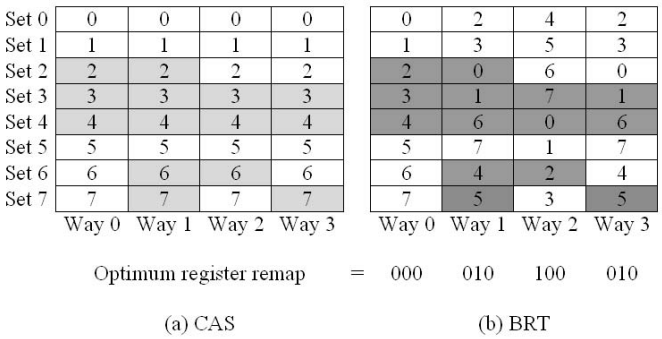


Fig. 1. Illustration of cache block organization in CAS and BRT. Shaded portions indicate the blocks which are affected by process variation. CAS=Conventional Addressing Scheme, BRT=Block Remap with Turnoff

strained by the worst-case of blocks in other ways of a cache, current way-prediction techniques for data caches are not accurate enough as in instruction caches [28]. In this paper, we work at set-level granularity.

## IV. BLOCK REMAP WITH TURNOFF TECHNIQUE

In order to minimize the impact of process variations in terms of performance and leakage power, we try to distribute the process variation affected blocks uniformly among all the sets and then turnoff the affected blocks so that the cache will have almost uniform associativity across the sets. To achieve this objective, we propose a technique called *block remap with turnoff* (BRT). We explain our technique in two stages, i.e., *Block remap* and *Block turnoff*.

### A. Block Remap

To get an optimal mapping which spreads process variation affected blocks almost uniformly across all sets, we have to consider possible $(m!)^n$ mappings for a $m$-set $n$-way set-associative cache. Though the search for optimal mapping can take place before the operational phase of a microprocessor, in order to reduce the preprocessing time, by trading accuracy for time, we consider a simple complexity-effective mapping for the BRT technique. In order to remap blocks in a way, we consider $log_2 m+1$ remap codes, which consists of $log_2 m$ one-hot codes and a null code (i.e., 00...0). When no remapping is required, we consider the null code. For each mapping, we perform bit-wise exclusive-or operation of block index with a remap code. As there are $log_2 m + 1$ remap codes and $n$ ways, we generate a maximum of $(log_2 m + 1)^n$ mappings. We select the best mapping (indicated by $n$ remap codes for $n$ ways) among all possible mappings, which yields the most uniform distribution of process variation affected blocks across the sets.

We consider a $log_2 m$-bit *remap* register for each way in a $n$-way set associative cache. After selecting the best mapping, we initialize the *remap* registers of all the ways with the corresponding remap codes. Note that the selection of a remap code and initialization of the remap register is done once in a while and that too before the operational phase of a microprocessor, so that the whole process does not affect the system performance. In order to remap a block to another block, we perform bit-wise exclusive-or operation of block index with the contents of the remap register. In other words, if $remap_k$ and $block_k^i$ are a remap register and index of an $i^{th}$ block of way $k$, $block_k^i$ can be remapped to a block $block_k^j$ such that $block_k^j = block_k^i \oplus remap_k$. The mapping mechanism used in BRT technique is illustrated by considering 3-bit remap codes as shown in Table I.

Note that, for remapping cache blocks as we consider xoring of set index with the selected remap code, we incur a slight time overhead of one xor-gate delay. This is similar to the boolean ANDing used in [18].

Note that our remap technique is different from the one given in [21], where same mapping is used for all ways of the cache and the remap code used is 001, i.e., two adjacent blocks are rearranged. Also, the objective of our remap technique is different from the that of the technique given in [21], where remapping is used to minimize the number of sets having both process variation affected and non-process variation affected blocks. Note that applying the same mapping to all cache ways can limit the possibility of turning off process variation affected blocks, which in turn minimizes performance gains and leakage power savings.

### B. Block Turnoff

After rearranging blocks using the BRT technique, if a set contains all clean (i.e., non process variation affected) blocks, then all the blocks are kept on and the set is accessed with low latency. If the set contains both clean and dirty (i.e., process variation affected) blocks, then we turnoff all the affected blocks and access the set with low latency. If all the blocks in the set are dirty, then all blocks are kept on and the set is accessed with high latency.

We illustrate BRT technique by considering a 4-way set-associative cache with 8 sets as shown in Figure 1. The light shaded blocks represent the process variation affected blocks which are kept on and the dark shaded blocks represent the turned off process variation affected blocks. Blocks in Figure 1 are arranged by considering the optimal mapping (000, 010, 100, 010). After remapping using our technique, two sets get one process variation affected block each and six sets get two process variation affected blocks each. From the figure, it is clear that our remap technique distributes process variation affected blocks almost uniformly across all sets. As no set has all four blocks affected by process variations, we can turn off all the process variation affected blocks so that the cache can

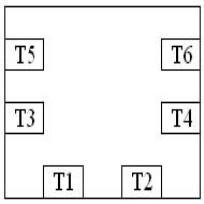


Fig. 2. 2D layout of a 6-T SRAM cell

be accessed with low latency and leakage energy can be minimized.

Note that, as shown in Figure 1 (a), in a CAS (Conventional Addressing Scheme), all the blocks of a set lie in the same row, where as shown in Figure 1 (b), in the BRT technique, blocks of the same set can lie in different rows.

## V. Experimental Validations

### A. Modeling process variation

In this work we consider WID variations. To model WID variations we first consider the 2D layout of 6-T SRAM cell [3] as shown in Figure 2. T1 and T2 are the pull down transistors, T3 and T4 are the access transistors and T5 and T6 are the pull up transistors. Process variations change the width, length, oxide thickness, etc of a transistor. All these changes can be modeled in terms of variations in the threshold voltage of the transistor. So instead of considering all the parameters seperately for each transistor, we consider only width, oxide thickness, and threshold voltage of each transistor. To model variations we consider random and systematic correlations. To model the random component, we use the statistical computing R-tool [24].

We model the systematic component as shown in Figure 3. In Figure 3, the 2D layout of four 6-T SRAM cells and the effect of transistor T5 on its neighbouring transistors is shown. The correlation component of the transistors which lie in the dark shaded area is incremented by ($L1$*random component of T5), the correlation component of the transistors which lie in the light shaded area is incremented by ($L2$*random component of T5) and the correlation component of the transistors which lie in the unshaded area remains unchanged. We can increase or decrease the number of levels where the effect of a affected transistor can be felt . We choose $L1 > L2$, because the variation effect on a transistor is inversely proportional to its distance from the affected transistor.

The final values of parameters for each transistor are obtained by adding their random and systematic variation components to their respective mean values.

Access latency of a SRAM cell is directly proportional to the threshold voltage of the transistors present in it. So the transistor with the highest threshold voltage in a SRAM cell decides its access time. Similarly the SRAM cell with the highest threshold voltage requirement decides the access time of the block.

To determine the leakage power dissipated in the cache, we consider two components, sub-threshold leakage ($I_{sub}$) and gate tunneling leakage ($I_{ox}$). These two components are calculated by using the following formulae [23].

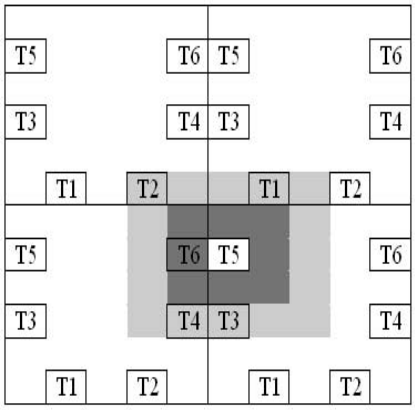


Fig. 3. 2D layout of four 6-T SRAM cells

$$I_{sub} = K_1 W e^{-V_{th}/\eta V_\theta}(1 - e^{-V/V_\theta}) \quad (1)$$

$$I_{ox} = K_2 W (V/T_{ox})^2 e^{-\alpha T_{ox}/V} \quad (2)$$

$K_1$, $\eta$, $K_2$ and $\alpha$ are experimentally derived parameters, $V$ is the supply voltage, $V_\theta$ is $25mV$ at room temperature, $W$ is width, $T_{ox}$ is oxide thickness and $V_{th}$ is threshold voltage of the transistor. Total leakage current is calculated by summing up the sub-threshold and gate tunneling components. Leakage power dissipated by a block is calculated by summing up the leakage power dissipated by all the transistors present in the block.

This whole process will give rise to four types of blocks in the cache. 1) Defect free blocks, 2) Blocks termed defective because of high access latency, 3) Blocks termed defective because of high leakage, and 4) Blocks termed defective because of high access latency and high leakage.

### B. Experimental setup

To validate our technique we run 21 SPEC2000 CPU benchmarks [33] on the SimpleScalar 3.0 [31] simulator. For each benchmark we fast-forward 1 billion instructions and then run next 500 million instructions. The configuration of the processor which we simulate is given in Table II. We assume that the blocks in L1 data cache have a latency of either $4$ cycles (if they contain transistors with threshold voltage above a cut off value), or 2 cycles (if they contain transistor with threshold voltage below the cut off value).

We compare BRT technique with *base case*, i.e., cache with no process variation so that access latency of the cache becomes 2 cycles, and *worst case*, i.e., cache with process variation affected blocks so that accessing the cache takes $4$ cycle latency. Note that in the BRT technique, we consider either 2 cycle latency (for sets having no high accesss latency blocks) or $4$ cycle latency (if a set has all four blocks having high access latency). For sensitivity analysis, we consider cache with $25\%$, $50\%$, and $75\%$ sets being affected by process variation.

| Parameter | Value |
|---|---|
| Issue width | 8 instructions/cycle (out of order) |
| RUU size | 128 instructions |
| LSQ size | 64 instructions |
| Branch prediction | Bimodal with 2K entries |
| BTB | 1K-entries, 4-way |
| Misprediction penalty | 18 cycles |
| # of ALUs | 8 integer + 8 floating point |
| # of Mul/Div units | 4 integer + 4 floating point |
| L1 D-cache | 32KB, 4-way (LRU), 64B blocks,<br>2 cycle latency (no process variation),<br>4 cycle latency (with process variation),<br>50% of the sets have 4 cycle latency<br>and the remaining have 2 cycle latency. |
| L1 I-cache | 64KB, 4-way (LRU)<br>32B blocks, 1 cycle latency |
| L2 cache | Unified, 512KB, 8-way (LRU)<br>64B blocks, 12-cycle latency |
| Memory | 160 cycles |
| ITLB | 16-entry, 4KB block, 4-way,<br>30-cycle miss penalty |
| DTLB | 32-entry, 4KB block, 4-way,<br>30-cycle miss penalty |

TABLE II
OUR DEFAULT PARAMETERS.

### C. Experimental Mechanism

We use a latency table to indicate the status of blocks in the cache. This table is initialized using the March test [8], which is performed before the operational phase of a microprocessor. Corresponding to each block in the cache, there is one bit in the latency table, which is *set* if the block is affected by process variation and *reset* otherwise. We use a 2-delta stride based address predictor [13] with 16K entries to predict the access latency of a set. In the case of misprediction, we replay all dependent instructions using the instruction-based selective replay technique [17].

### D. Experimental results

Figure 4 shows benchmark-wise IPC values for different techniques. In BRT technique we turnoff few blocks after remapping them between sets, resulting in majority of the sets having low latency and low associativity. Except for "apsi", "art", "galgel", and "lucas", our technique achieves performance close to the base case. This is due to the fact that these benchmarks does not require the full space of the cache and access only part of the cache. The performance penalty in benchmarks "apsi", "art", "galgel", and "lucas", is because of frequent accesses to the sets with low associativity, resulting in increased miss rate.

Figure 5 shows percentage of average performance degradation with respect to the base case. Accessing all the sets with $4$ cycle latency (irrespective of percentage of sets being affected by process variation) results in nearly $5.8\%$ performance penalty. When the percentage of sets affected by process variation is very low, our technique incurs negligible performance

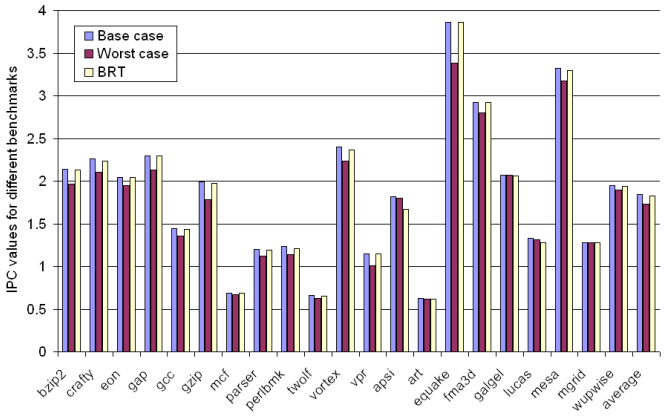


Fig. 4. Benchmark-wise IPC for different techniques w.r.t. the base case. Note that here we assumed 50% of cache sets are being affected by process variation.

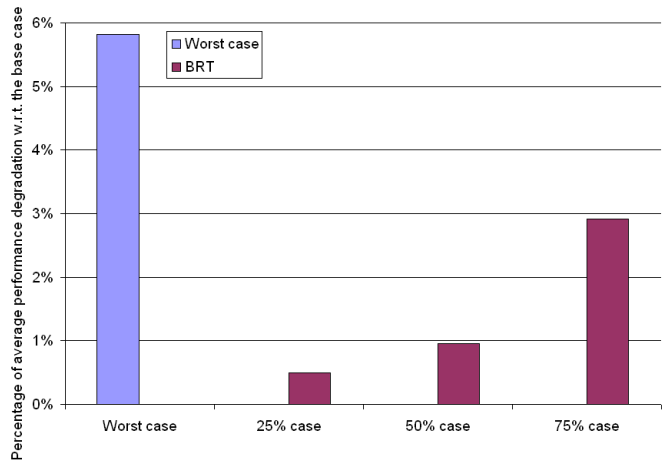


Fig. 5. IPC degradation w.r.t. the base case.

penalty. Even for the $50\%$ case, the performance penalty of our technique is less than $1\%$ of the base case. On the other hand, as the variation percentage increases beyond $50\%$, because of many blocks being turned off, not much space is available in the cache, increasing the peformance penalty of our technique. For the $75\%$ case, our technique incurs a penalty of less than $3\%$.

Figure 6 shows relative leakage energy savings due to turning off process variation affected blocks w.r.t. the worst case (where process variation affected blocks are kept on). As expected, with increase in variation percentage, the relative leakage energy savings are increased. For $50\%$ case, we achieve leakage energy savings of about $20\%$.

From the above results, it is clear that BRT technique significantly minimizes performance penalty and leakage power consumption due to process variations.

## VI. CONCLUSION

Process variation is a serious problem in deep submicron circuits, which if not taken care of, can result in a functionally correct chip getting rejected. In this paper, we proposed a technique to uniformly distribute process variation affected blocks among sets and turn them off. Though turning off process vari-

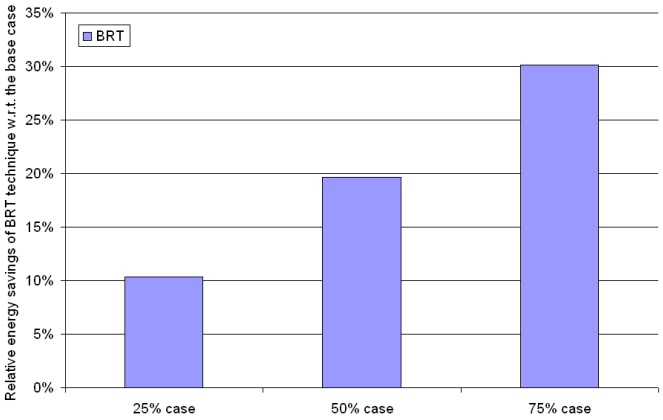


Fig. 6. Relative leakage energy savings of BRT technique w.r.t. the worst case.

ation affected blocks results in a cache with reduced associativity per set, our technique significantly minimizes performance penalty and leakage energy consumption due to process variation. Experimental results demonstrate that for a $50\%$ affected cache, our technique incurs a performance penalty of less than $1\%$ with respect to the base case and saves about $20\%$ leakage energy as compared to the worst case.

## REFERENCES

[1] A. Agarwal, *et. al.*, "Process variation in embedded memories: failure analysis and variation aware architecture", *IEEE JSSC*, Vol. 40, pp. 1804-1814, 2005.

[2] D.H. Albonesi, "Selective cache ways: on-demand cache resource allocation", *IEEE Micro*, pp. 248-259, 1999.

[3] David A. Hodges, Horace G. Jackson, Resve A. Saleh, *Analysis and design of Digital Integrated Circuits in deep sub micron technologies*, McGraw Hill, 2004.

[4] I. Arsovski and R. Wistort, "Self-referenced sense amplifier for across-chip-variation immune sensing in high-performance content-addressable memories", *IEEE CICC*, pp. 453-456, 2006.

[5] M. Bekerman, A. Yoaz, F. Gabbay, S. Jourdan, M. Kalaev, and R. Ronen, "Early load address resolution via register tracking", ISCA, pp. 306-315, 2000.

[6] S. Borkar, *et. al.*, "Parameter Variations and Impact on Circuits and Microarchitecture", *DAC*, pp. 338-342, 2003.

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

[8] M.L. Bushnell and V.D. Agarwal, *Essentials of electronic testing for digital memory, and mixed-signal VLSI circuits*, Kluwer, 2000.

[9] H. Chang and S. Sapatnekar, "Full-chip analysis of leakage power under process variations, including spatial correlations", *DAC*, pp. 523-528, 2005.

[10] Q. Chen, *et. al.*, "Modeling and testing of SRAM for new failure mechanisms due to process variations in nanoscale CMOS", *VTS*, pp. 292-297, 2005.

[11] J.A. Croon, *et. al.*, "Physical modeling and prediction of the matching properties of MOSFETs", *ESSDERC*, pp. 193-196, 2004.

[12] A. Datta, *et. al.*, "Speed binning aware design methodology to improve profit under process variations", *ASP-DAC*, pp. 712-717, 2006.

[13] R.J. Eickemeyer and S. Vassiliadis, "A load instruction unit for pipelined processors", *IBM J. of Research and Development*, Vol. 37, pp. 547-564, 1993.

[14] R. Heald and P.Wang, "Variability in sub-100nm SRAM design", *ICCAD*, pp. 347-352, 2004.

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

[16] M.H. Lipasti *et. al.*, "Value locality and load value prediction", *ASPLOS*, Vol. 30(5), pp. 138-147, 1996.

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

[18] K. Meng *et. al.*,"Physical resource matching under power asymmetry", *P=ac2 Conference, IBM TJ Watson Research Center, Yorktown, NY*, pp. 1-10, October 2006.

[19] K. Meng and R. Joseph, "Process variation aware cache leakage management", *ISLPED*, pp. 262-267, 2006.

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

[21] M. Mutyam and V. Narayanan, "Working with process variation aware caches", *DATE*, pp. 1152-1157, 2007.

[22] S. Narendra, *et. al.*, "Forward body bias for microprocessors in 130-nm technology generation and beyond", *IEEE JSSC*, 38(5), pp. 696-701, 2003.

[23] Nam Sung Kim, *et. al.*, "Leakage Current: Moore's Law Meets Static Power", *IEEE Computer Society*, pp. 68-75, 2003.

[24] The R project for Statistical computing, http://www.r-project.org./

[25] S. Nassif, "Within chip variability analysis", *IEEE IEDMC*, pp. 283-286, 1998.

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

[27] S. Ozdemir, *et. al.*, "Yield-aware cache architectures", *IEEE Micro*, pp. 15-25, 2006.

[28] M.D. Powell *et. al.*, "Reducing set-associative cache energy via way-prediction and selective direct-mapping", *MICRO*, pp. 54-65, 2001.

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

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

[31] SimpleScalar toolset, http://simplescalar.com/

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

[33] SPEC 2000 benchmarks, http://www.spec.org/cpu/

[34] J. Tschanz *et.al.*, "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage", *IEEE JSSC*, Vol. 37(11), pp. 1396-1402, 2002.

[35] P. Zuchowski *et. al.*, "Process and environmental variation impacts on ASIC timing", *ICCAD*, pp. 336-342, 2004.