IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 25, 2019, accepted January 16, 2020, date of publication January 22, 2020, date of current version January 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2968603

Near-Threshold L1 Data Cache for Yield Management Under Process Variations

JOONHO KONG[1], (Member, IEEE), AND JAE YOUNG HUR[2], (Member, IEEE)
[1]School of Electronics Engineering, Kyungpook National University, Daegu 41566, South Korea
[2]Faculty of Engineering, Vietnamese-German University, Binh Duong 75114, Vietnam

Corresponding author: Joonho Kong (joonho.kong@knu.ac.kr)

This work was supported by the Samsung Electronics.

ABSTRACT Near-threshold computing (NTC) has recently emerged and been considered as a strong candidate for future energy-efficient computing. However, adverse impacts from process variation such as delay and power fluctuations within die as well as across dies are much more severe than the super-threshold regime. In particular, static random access memory (SRAM)-based components (e.g., cache memories) are easily affected by process variation in NTC, resulting in large delay fluctuations. It incurs a huge loss in the maximum clock frequencies of processors, which eventually leads to huge yield losses. In this paper, we first analyze L1 data cache yield in NTC and reveal an inefficiency of frequency binning for yield improvement in NTC. We then introduce a variable latency L1 data cache for NTC to obtain a sufficient yield. By allowing the higher cache access cycles, we can improve cache yield with only a little performance overhead. Moreover, we propose an adaptive line migration technique which improves performance and energy efficiency of variable latency caches. The cache line which is expected to be frequently accessed in the near future is dynamically migrated to the fastest way in a cache set. According to our evaluation, our cache architecture greatly improves cache yield with only a little performance, energy, and area overhead.

INDEX TERMS Process variation, near-threshold computing, cache memory, system performance, energy efficiency.

I. INTRODUCTION

Process variations, manufacturing defects and variability in device parameters, have been a major threat in improving performance, energy and yield (the ratio of the number of usable (or sellable) chips to the number of total manufactured chips) of processors in nano-scale design era. It is known that process variation makes a non-uniform distribution of device parameters (e.g., threshold voltage and effective gate length), resulting in delay and power fluctuations within die as well as across dies. On the other hand, near-threshold computing (NTC) has emerged and considered as a strong candidate for future energy-efficient computing. The circuit operation is performed in near-threshold voltage levels enabling a great reduction in dynamic and leakage power consumption. Increased latency is compensated by exploiting parallelism in workloads. However, in near-threshold computing regime, process variations would become the most severe hurdle for enabling a mass production of near-threshold processors. As revealed in [1], only a small fluctuation in device parameters causes huge delay variations in NTC regime, which may potentially lead to severe yield losses.

Within a processor, SRAM-based structures (e.g., caches) are prone to process variations [2]. The device parameter mismatches within an SRAM cell can incur several failures: read/write failures, access time failures, and hold failures [3]. In particular, access time failure is most threatening as one uses more advanced process nodes [3]. This hypothesis is also valid for NTC regime. Kong et al. revealed that the SRAM-based components are most vulnerable to process variation in many-core processors for NTC [2]. With a certain clock frequency requirement, processor's yield would be significantly reduced as SRAM-based components (e.g., cache memories) are hard to meet the latency requirements.

Among several cache hierarchies inside of processors, L1 data cache is highly latency critical due to frequent appearance of load/store instructions. If a program is cache-intensive (i.e., a high ratio of load/store instructions rather than arithmetic or branch instructions), performance will be significantly affected depending on L1 data cache

The associate editor coordinating the review of this manuscript and approving it for publication was Yi Zhang.

access latency. Therefore, typical L1 data cache design focuses on enabling a short access cycle. However, in order to achieve a comparable yield under process variations, trying to keep the access cycle of L1 data caches (i.e., increase global clock cycle time) may incur severe clock frequency losses since there is much larger delay increase in NTC due to process variation than in super-threshold regime [2]. In this case, an efficient trade-off between the L1 data cache access cycles and processor clock frequency is the most critical factor for improving yield and performance of processors.

In this paper, we revisit a variable latency cache architecture for NTC while further improving energy and performance by employing an adaptive line migration technique. For super-threshold computing (STC), several proposals for variable latency cache architecture have been introduced [4]–[6]. However, there has been few work regarding variable latency cache for NTC. To the best of our knowledge, our work is the first work that improves yield, performance, and energy efficiency of L1 data caches operating in near-threshold voltages. In the following section, we demonstrate the yield impact of process variation in NTC, which necessitates a yield-aware variable latency cache for NTC. Furthermore, we newly propose an adaptive line migration scheme which improves energy efficiency and performance of variable latency L1 data cache for NTC. Our scheme dynamically migrates cache lines which are likely to be frequently accessed in the near future. To prevent a migration thrashing, we newly introduce a promotion threshold that enables an accurate prediction of subsequent accesses in the near future. Moreover, compared to a frequency (speed) binning technique [7] that is widely used for yield improvement, our proposed cache architecture improves not only yield but also performance and energy efficiency. In addition, as compared to disabling the faulty cache blocks (WBD) [8] (similar to Intel Pellston Technology) [9], [10] (introduced as a naïve way reduction scheme), our technique leads to much better performance with comparable yield even under severe process variations. Considering that the yield significantly affects the profitability of the chip manufacturing companies, our work can be a promising alternative to future near-threshold processor design for manufacturability. We believe that our proposal can be a one step forward for enabling a mass production of processors that operate in near-threshold voltage levels.

In summary, our contributions include:

- We provide a detailed analysis on yield in NTC particularly for L1 data cache in a processor. We demonstrate a huge reduction of yield in NTC regime and show that it is hard to be recovered by adopting the frequency binning [7] which is a commonly used technique in industries;
- We also demonstrate that the necessity of employing the variable latency cache architecture for NTC and propose an adaptive line migration technique that can be employed along with the variable latency caches for yield, performance, and energy efficiency improvements of near-threshold L1 data cache;
- We show that our cache architecture achieves much higher cache yield (5.1% $\sim$ 100%) with comparable performance and energy efficiency to the ideal case (i.e., no process variation). Compared to the frequency binning, our cache architecture shows much higher yield (improving yield by 2.4% $\sim$ 71.0%) with up to 91% performance improvement and up to 32% energy reduction;
- Our cache architecture also shows much better performance by 2.1X compared to the WBD technique under severe process variations.
- Our proposed cache architecture can be implemented with a small hardware overhead, which corresponds to less than 2% of L1 data cache area.

The rest of the paper organization is as follows. Section 2 demonstrates our motivational study and detailed yield analysis which call for yield-aware techniques for L1 data cache in NTC. Section 3 describes our proposed variable latency cache with a line migration technique for performance and energy efficiency improvement as well as yield management. Section 4 shows our evaluation results in terms of yield, performance and energy consumption. Section 5 discusses related work and describes our novelty over those works. Lastly, Section 6 concludes this paper.

II. MOTIVATIONAL STUDY: L1 DATA CACHE YIELD ANALYSIS

A. A FRAMEWORK FOR YIELD ANALYSIS

In this subsection, we present our framework for L1 data cache yield analysis. Our yield analysis is based on Monte Carlo simulation with VARIUS-NTV [1] originated from R statistical programming language [11]. Table 1 shows parameters for process variations and circuits. Most of the parameters are set identical to [12]. For architectural parameters of L1 data caches, we use 32KB 4-way set associative L1 data cache with 64Byte line size, which means there are 512 cache lines in the L1 data cache. The replacement policy of L1 data cache is a least recently used (LRU) policy. For Monte Carlo simulation, we examine V_{th} (threshold voltage) and L_{eff} (effective gate length) variations with a cache line granularity by using VARIUS-NTV process variation model [1]. Based on V_{th}, L_{eff}, and other circuit parameters for cells in each cache line, we extract a latency for each line from VARIUS-NTV [1] (for the details of the delay model, please refer to [1] which describes read and write delay models).

Table 2 summarizes the parameters used for different process variation severities in our Monte Carlo simulations. We analyze L1 data cache yield with three different process variation levels. We classify them into three degrees of parameter fluctuations: σ/μ (standard deviation / mean) of V_{th} = 0.1, 0.15, and 0.2 which are denoted as 'var10', 'var15', and 'var20', respectively. We give a variation to threshold voltage (V_{th}) and effective gate length (L_{eff}). The σ/μ of L_{eff} is set to a half of σ/μ of V_{th} for each level of process variation severity (i.e., σ/μ of L_{eff} = 0.05, 0.075, and 0.1).

TABLE 1. The basic circuit and architectural parameters.

Categories	Parameters used for simulations
Technology node	11nm
Nominal V_{dd}	0.55V [12]
Nominal V_{th}	0.33V [12]
Subthreshold slope factor	1.5
Mobility exponent	-1.5
Change in V_{th} per temperature (K) increase	$-1.5*10^{-3}$
Nominal processor clock frequency	1 GHz [12]
L1 D-cache	32KB 4-way 64-byte line size
L1 I-cache	32KB 2-way 64-byte line size
L2 cache	2MB 16-way 64-byte line size
Processor core	Close to ARM Cortex-A15 [13]
Operating temperature	353K
Correlation range (φ)	0.1

TABLE 2. σ/μ of V_{th} and σ/μ of L_{eff} across three process variation severities.

	var10	var15	var20
σ/μ of V_{th}	0.1	0.15	0.2
σ/μ of L_{eff}	0.05	0.075	0.1

Both V_{th} and L_{eff} distributions follow a normal distribution. Please note that the parameters for 'var20' is used in [1] while those for 'var10' and 'var15' are used in [2]. For comprehensive evaluations, we use three different process variation levels of which parameters are collected from different previous works.

In order to map the process parameters to each cache line, we divided L1 cache area into 512 rectangles. The L1 cache area is scaled from [1] considering the capacity of the L1 data cache. The cache dimensions used for Monte Carlo simulations are shown in Figure 1. We give the floorplan shown in Figure 1 as an input to VARIUS-NTV for Monte Carlo simulations. We generate 1000 process variation maps for each process variation level, generating total 3000 process variations maps. Though we present our analysis with a single-core L1 data cache, it can be extended to many-core architecture where there is per-core L1 data cache. We use 8T SRAM cells, as in [1] and [2], which is known to be more robust to process variations compared to the conventional 6T SRAM cells [14]. We also assume that the baseline L1 data cache access takes three clock cycles: address decoding, SRAM array access, and data out.

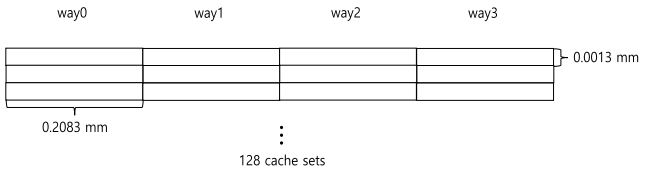

FIGURE 1. A dimension of L1 data cache used for our Monte Carlo simulations.

B. LATENCY AND YIELD ANALYSIS FOR NEAR-THRESHOLD L1 DATA CACHE

First, we present a maximum possible frequency considering the maximum latency of the L1 data cache (i.e., maximum latency among 512 lines). Figure 2 shows maximum frequency distributions from our Monte Carlo simulation across three different process variation levels. The latency extracted from our Monte Carlo simulation is SRAM array access latency (the second cycle of the L1 data cache access). Thus, we only consider the increased latency of SRAM array access cycle, which affects the maximum available clock frequency of processors. Please note that the SRAM access cycle is the most vulnerable cycle (i.e., easy to be fluctuated) among three cache access cycles due to the SRAM cell's vulnerability to process variations [2].

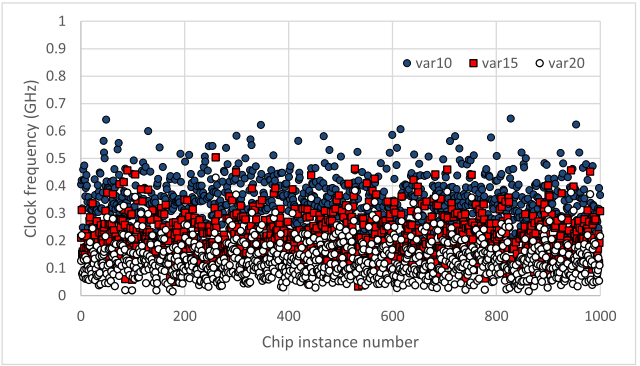

FIGURE 2. Maximum clock frequency plots from the Monte Carlo simulations.

Considering the nominal clock frequency is 1 GHz, the possible maximum clock frequency is severely limited by the L1 cache access latency. In the case of 'var10' (the lowest parameter fluctuation in our analysis), the maximum clock frequency among 1000 processors is 0.644 GHz, which is far below the nominal clock frequency of 1GHz. In the case of 'var20' (the highest parameter fluctuation in our analysis), the situation gets further worse, resulting in the maximum clock frequency of 0.429 GHz. If the clock frequency requirement is firmly set to 1GHz, the microprocessor yield will be 0%. Since we have very low yield, one can employ a frequency binning [7] for better yield while sacrificing performance of processors. Table 3 shows the (fixed) frequency binning results based on our Monte Carlo simulations. In the case of 'var10' where there is least significant process variation severity, one can get only 5 chips operating over

TABLE 3. Frequency binning based on the L1 data cache latency across three variation levels. 'x' means the clock frequency. For example, if the maximum clock frequency of the processor is 250MHz, it is binned to 0.2 < x ≤ 0.4GHz.

	var10	var15	var20
$0 < x \leq 0.2$GHz	24	517	871
$0.2 < x \leq 0.4$GHz	763	465	128
$0.4 < x \leq 0.6$GHz	208	18	1
$0.6 < x \leq 0.8$GHz	5	0	0
$0.8 < x \leq 1.0$GHz	0	0	0

0.6GHz (0.5%). A majority of the chips are operating below 0.4GHz. It means one should bear a huge performance loss compared to the chips in the nominal frequency (1GHz) in order to achieve a comparable yield. In the perspective of semiconductor company, they have no choice but to sell their manufactured processors with a very low price (due to low performance), which leads to a loss of considerable profits.

It is known that the PV-induced stability issues are much more dominant than the delay variation under extremely low V_{dd} [14]. However, based on our assumptions presented in Table 1, V_{min} (the minimum V_{dd} to guarantee a stable operation) is still lower than 0.55V even under process variation in most cases, meaning that the stability issue gives only a marginal impact on yield. It is already observed in [2] that delay variation in cache memories is the most dominant factor in NTC under process variations.

III. VARIABLE LATENCY L1 DATA CACHE FOR NEAR-THRESHOLD COMPUTING

A. VARIABLE LATENCY L1 DATA CACHE FOR NTC

In order to efficiently handle the problem stated in Section 2, we propose to use variable latency-based L1 data cache instead of fixed cycle L1 data cache. Variable latency cache architecture is not a new concept and it has been explored for process variation-aware design [4], [6], performance improvement [5] and energy reduction [15]. We extend the conventional variable latency cache architecture mainly for yield improvement in NTC.

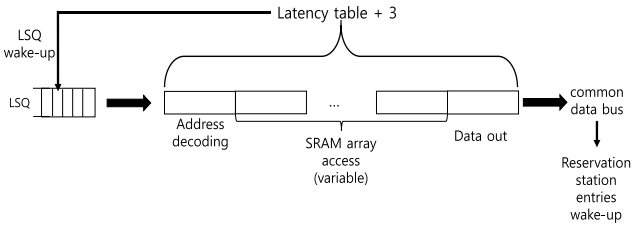

FIGURE 3. A conceptual diagram of variable latency L1 data cache.

Figure 3 depicts a conceptual description of our variable latency cache architecture. As we explained in Section 2.A, a cache access is divided into 3 cycles in the nominal case. However, as delay variations from SRAM cells are significant [2], we apply variable clock cycles to SRAM array access cycle (the second cycle of the L1 data cache access in the nominal case). Our variable latency L1 data cache can have different access cycles for each cache line. Hence, the latency information from the latency table is used for a wake-up of the corresponding LSQ entry and reservation station entries. Though we could allow any cycles equal to or higher than 3 ($3\sim\infty$) for variable latency L1 data caches, the reservation station (RS) and load/store queue (LSQ) should be notified in an asynchronous manner (i.e., they cannot know the timing of the wakeup notification), resulting in much higher design and verification complexity. Thus, we should maintain latency information in the latency table for each cache line.

Figure 4 illustrates the overall (micro-)architecture of our variable latency L1 data caches. As in the case of typical cache architecture, there are cache tag and data arrays where we store tags and data for caches, respectively. In a typical cache access, a data cache access request (address in the case of load and address and data in the case of store) is issued from load/store queue (LSQ) and fed into the L1 data cache. If the L1 data cache load hit occurs, the time when the data will be delivered back to the LSQ and reservation stations is various across the cache lines since each cache line will have different access cycles due to process variations. To make the LSQ and reservation stations accurately estimate the data arrival time in the future, the latency table is accessed along with the L1 data cache access. The latency table is to store the information of the SRAM array access cycles additionally required for each cache line due to process variations (e.g., if the latency table value is zero, it means there is no additional latency due to process variations; thus, cache access cycle is 3). To access the latency table, we first access L1 data cache tag array to locate the set and way, which will be an input to the latency table. After we obtain the latency information, we add 3 to incorporate the latency of the cache access cycles for the nominal case (i.e., 3 cycle access latency). After that, we set the LAT (latency from latency table) + 3 as the initial value of the binary down counter and set the enable signal as '1'. This will decrement the value in the binary down counter by 1 for each clock cycle while it is compared with the zero in the comparator (XNOR gates) every clock cycle. After LAT+3 cycles, the comparator signal will be changed from '0' to '1'. The comparator output is connected to LSQ and reservation stations, which wakes up the corresponding LSQ and reservation station entries. At the same time, the data from the L1 data cache will also be available in common data bus and forwarded to the relevant reservation station entries in order to issue dependent pending instructions in the following cycle. Since the L1 data caches are typically employed in a per-core or per-CPU manner, if we have multi-core or multi-CPU chip, we need to have separate latency tables for each private L1 data cache.

In the case of L1 data cache store hit, the increased latency is not sensitive as much as in the case of load instructions because the processor pipeline is not stalled for the data delivery. However, the corresponding LSQ entry for the store instruction should also consult the latency table and keep

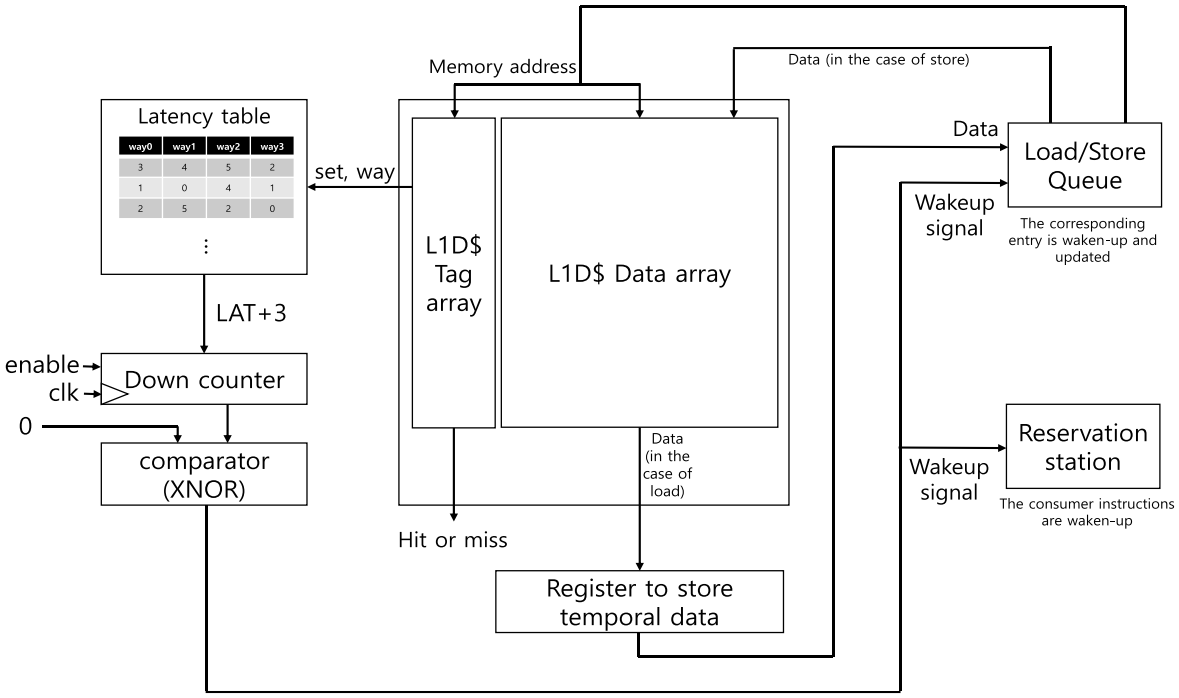

FIGURE 4. The overall (micro-) architecture of our variable latency L1 data caches.

the LSQ entry until the write operation is completed in the L1 data cache. This is because the data written to the cache can also be forwarded to awaiting load instructions that access the same memory address.

In the case of cache misses, as soon as the tag comparison is done and the processor realizes a cache miss, the SRAM array access in the L1 data cache is immediately stopped and a request is sent to the L2 cache. In variable latency L1 data cache for NTC, the cache access completion often takes a number of clock cycles. To avoid performance overhead due to long latencies of cache accesses, a data request to the L2 cache is sent (in the case of L1 data cache miss) as soon as the tag comparison is done.

For a real employment of our proposed technique, the latency table initialization is crucial. The latency table is initialized by SRAM latency testing such as March testing described in [16]. This is a common practice of SRAM array manufacturing that performs a post-manufacturing latency testing to identify delay-failed SRAM cells. It can often be accomplished by using Built-In Self-Test (BIST) logic which is generally included with the memory arrays. After the latency testing, we can figure out the cache block-level latency that corresponds to the worst-case latency of the SRAM cells in a cache block. After the latency testing, we can figure out whether the SRAM array corresponds to a yield loss or not. If there is any cache block of which latency is more than the $3+2^{B\_lat}$ (B_lat = bitwidth of the latency table entry), the cache will be regarded as unusable. If latencies of the entire cache blocks are less than $3+2^{B\_lat}$, the additional latency for SRAM array access (i.e., cache access latency - 3) is recorded in the latency table. For permanently storing the latency table information, it can be additionally stored in the non-volatile cells (secondary storage memory or small read-only memory cells) and can be loaded when the operating system (OS) initializes the system (at the system booting time) though the detailed mechanism of OS is out-of-the-scope of this work. Please note that we can utilize a similar mechanism that is used in [17], [18] where the fault bit is loaded in advance before we use the cache memory.

The latency overhead of our latency table is only 0.0573403ns (modeled by CACTI [19]) which is much less than the baseline clock cycle time (1ns). In addition, we need a 4-bit adder delay to calculate the LAT+3. Considering that modern microprocessors can sufficiently perform the 32-bit or 64-bit arithmetic or logical operations within one clock cycle, 4-bit adder delay will be negligible with carry-lookahead adders. Considering this aspect, tag access (latency = 0.105794ns), latency table access (0.0573403ns), and 4-bit addition with 0011_2 can sufficiently be performed within one clock cycle. Consequently, it can sufficiently trigger the down counter before the first (either positive or negative depending on which type of the flip-flops are used in the processor) clock edge.

B. OUR ADAPTIVE LINE MIGRATION SCHEME

As illustrated in Section 2, the latency variability across cache lines is huge. It means allocating frequently accessed data to low latency lines within a cache set would be beneficial in terms of performance. Since L1 data caches are very frequently accessed (nearly every cycle in the case of cache- and

memory-intensive workloads), an access cycle reduction of L1 data cache would translate into a noticeable performance improvement.

To accurately predict the cache lines which will be frequently accessed in the near future, we exploit a locality characteristic within a cache set. In general caches, recently accessed cache lines have a high possibility to be accessed again in the near future (a.k.a., locality). As shown in Figure 5, most of the L1 data cache accesses occur in the MRU0 (most recently used) line within a cache set. Thus, according to the location of the MRU line within a cache set, performance of variable latency L1 data cache would be hugely affected.

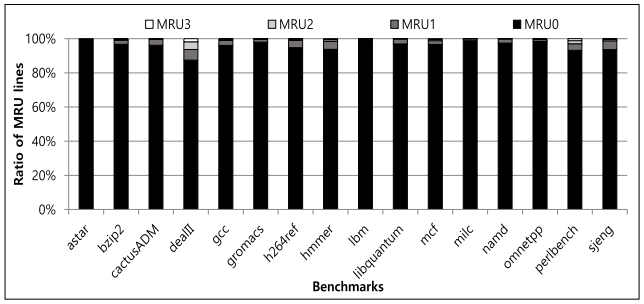

FIGURE 5. The ratio of cache hit access from MRU (most recently used) to LRU (least recently used) lines in a cache set. MRU0 is the most recently used line and MRU3 is the least recently used line in the cache set.

In this paper, we propose an adaptive line migration scheme which is geared toward our variable latency L1 data cache for NTC. Our migration scheme maintains special storages: a hit counter (2-bit: representing $0 \sim 3$ ($00_2 \sim 11_2$)) for each cache line and fastest way indicator (2-bit) for each cache set. For the quantitative measure of the importance for each cache line, we use the hit counters. According to the hit counter value, we determine whether the corresponding cache line should be migrated or not (the details will be explained in the following paragraph). When the processor is initialized, all of the hit counters are set to '0'. The fastest way is determined according to the latency testing of each cache line after chip fabrication (it can be initialized by the same process of latency table initialization). It stores the way number in a cache set which has the lowest latency among four cache lines (in the case of 4-way set associative).

Alg. 1 describes our algorithm for the adaptive line migration scheme. When there is a cache hit in the fastest way (denoted as cache line 'B' in Alg. 1), there is no additional action in our algorithm and it is totally same as a normal cache access. If there is a cache hit in a cache line which is currently not stored in the fastest way (denoted as cache line 'A' for description in Alg. 1), the hit counter is compared with the promotion threshold ($P_{threshold}$). If the hit counter is greater than or equal to the predefined $P_{threshold}$, this cache line is regarded as a very frequently accessed line. Hence, if the migration buffer (it will be explained below) is not full, we swap the data (64-byte) and corresponding metadata (tag and cache line status) between the two cache lines. It means the data of which hit counter exceeds $P_{threshold}$ (stored in cache line 'A') is migrated to the fastest way (cache line 'B'). The previously stored data in the fastest way is also migrated to the line 'A'. The hit counters of both swapped cache lines are reset to 0 after the line swap operation (lines 6 and 7 in Alg. 1). If the hit counter of the cache line is less than $P_{threshold}$ the hit counter of that cache line is increased (lines 9 and 10 in Alg. 1). In the case of cache misses (lines 12∼15 in Alg. 1), the hit counter of the replaced cache line is set to 0. The other procedure for the cache miss handling is identical to the conventional L1 data cache with LRU replacement policy.

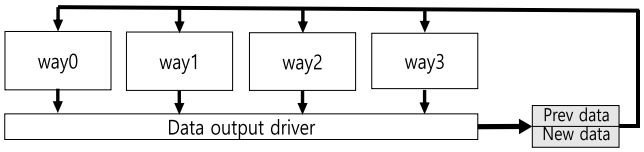

FIGURE 6. A line migration (swap) buffer for our adaptive line migration scheme.

The lower promotion threshold we have, the better performance we would obtain. However, lower promotion threshold may incur too frequent cache line swap (i.e., migration thrashing) which will translate into higher energy overhead. According to our architectural simulation, the cases of $P_{threshold} = 1$ and $P_{threshold} = 3$ show little performance difference (0.45% better in the case of $P_{threshold} = 1$ than that of $P_{threshold} = 3$) while there is a non-negligible difference on the number of line swap occurrence (33.8% less in the case of $P_{threshold} = 3$ than that of $P_{threshold} = 1$). Since more line swap operations leads to higher energy consumption, we set $P_{threshold}$ by 3 in this work. For a certain situation, even with a high $P_{threshold}$ value, there would be frequent line migrations. However, according to our simulation results, most cache accesses happen in the MRU block. It implies once we move the frequently accessed block to the fastest line in a cache set, there will be many subsequent cache hits in the fastest cache line without migration.

For a smooth line migration, we have two-entry buffer which temporarily stores two sets of 64B data and metadata (shown in Figure 6). By using this buffer, the line migration process is performed in background and does not make the processor pipeline be stalled. One entry has newly promoted data (new data in Figure 6) and the other entry has the data previously stored in the fastest way (prev data in Figure 6). The data is kept in the migration buffer until the swap operation in the L1 data cache is completed. To minimize performance overhead, if the migration buffer is full (i.e., migration is currently being performed), the migration for that cache line is skipped (line 4 in Alg. 1).

Since migration is done very sporadically (only ∼1% of the number of cache access), only 2-entry migration buffer is sufficient. It also means a negative performance impact of migration buffer overflow (that causes a migration skipping) would be negligible. To minimize negative performance

impact of migration, we could perform an actual migration when there is a long stall in the pipeline (e.g., L2 cache miss, TLB miss, etc.) though it is out-of-the-scope of this paper.

C. MICRO-ARCHITECTURAL SUPPORT AND COST ANALYSIS

Figure 7 depicts additional storages for our variable latency cache with the line migration scheme. As we already explained in Section 3.A, for our variable latency L1 data cache, we should maintain latency information for each cache line. The latency is stored with a clock cycle granularity. The value stored in the latency table corresponds to the additional clock cycle required to access that cache line. Thus, if the value is '0', the cache access cycle for the corresponding cache line is 3 cycles (i.e., nominal case). Depending on the bit-width in the latency table, cache yield is affected. For instance, if we have 2-bit latency storage for each cache line, the maximum allowable cache access latency is 6 cycles since two bits can only represent $0 \sim 3$. Thus, in this case, if there is a cache line whose access cycle exhibits more than 6 cycles, the chip is regarded as a non-usable chip (a yield loss). Consequently, the larger latency storage we have, the higher cache yield we can expect.

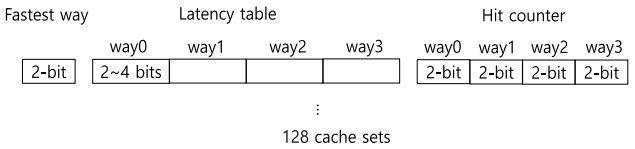

FIGURE 7. Additional storage maintained for the variable latency cache and line migration scheme.

To support our line migration scheme, for each cache set, 2-bit storage (fastest way) is maintained for rapid line migration. When the migration occurs, one may perform comparison between the latency values from the latency table instead of having the separate 2-bit fastest way storage. However, to reduce energy and performance overhead from the comparison operations in our design, we have the storage dedicated for maintaining the fastest cache line within a cache set.

Along with the fastest way storage and latency table, there is a 2-bit hit counter for each cache line. Depending on the promotion threshold in Alg. 1, one can increase or decrease the size of hit counters. Since we use promotion threshold of 3 in this work, we only need 2-bit for each hit counter.

For additional storages depicted in Figure 7, we need 26 bits per cache set, which implies 416 ($= 26*128/8$) Bytes are required for 32KB L1 data cache. For migration buffer, we need two-entry 64-byte buffer for data migration and two-entry 64-bit (conservatively estimated) storage for cache status metadata including cache tag bits. Thus, for migration buffer, one needs 144 Bytes of additional storages. Consequently, to implement our variable latency L1 data cache with the adaptive migration scheme, 560-Byte additional storage is required, which corresponds to $<2\%$ of the cache data array size (32KB). Though we present our technique based on 32KB cache size, it can also be applied to larger caches. Assuming that we have the same cache block (line) size, our technique can also be implemented in larger caches (e.g., 64KB) with larger latency table, fastest way storage and hit counter storage. Though we need larger table and storages, the delay, area, and power overhead compared to the cache will be similar because the table and storage size is proportional to the cache size.

Algorithm 1 Our Adaptive Line Migration Scheme

Assumption: the cache line 'B' is the fastest way in a cache set.
Assumption: the cache lines 'A' and 'B' are in the same cache set.

```
1   if (cache hit in the cache line 'B')
2     no further action;
3   else if (cache hit in the cache line 'A') {
4     if (hit counter of 'A' >= P_threshold &&
        !migration buffer full) {
5       Cache line swap between 'A' and 'B';
6       hit counter of 'A' = 0;
7       hit counter of 'B' = 0;
8     }
9     else if ( hit counter of 'A' < P_threshold)
10      hit counter of line 'A' ++;
11  }
12  else if (cache miss) {
13    line eviction and replacement;
14    hit counter of the replaced cache line = 0;
15  }
16  end
```

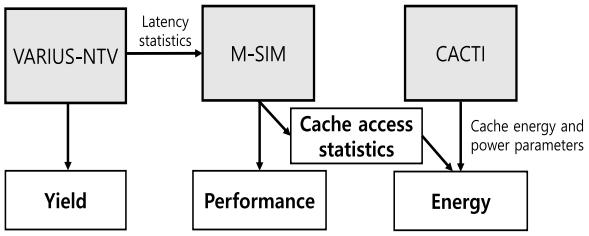

FIGURE 8. An overall usage flow of the tools used in our evaluation framework.

IV. EVALUATION

A. EVALUATION FRAMEWORK

For evaluation, we present our results in terms of yield, performance, and energy in this section. Figure 8 summarizes the usage flow of the tools used in our evaluations. For yield evaluation, we use the framework (VARIUS-NTV) explained in Section 2.A. For performance evaluation, we use M-SIM architectural simulation tool [20]. Please note that M-SIM is a derivative of SimpleScalar tool [21]. Internal micro-architectural parameters of the M-SIM simulator

closely model ARM Cortex-A15 [13]. To model variable latency cache in M-SIM simulator, we assign a random latency between 3 $\sim$ 18 cycles (i.e., in the case of variable latency cache with 4-bit latency information per cache line) to each cache line in the L1 data cache (it corresponds to the part in Figure 8 where the latency statistics delivered from VARIUS-NTV to M-SIM). Please note that the distribution of the cache access latency between 3 $\sim$ 18 cycles corresponds to the process variation levels between var15 and var20. For benchmark, we run 16 selected benchmark programs from SPEC2006. Since running the entire benchmark program takes huge time for simulations, we fast-forward 2 billion instructions and actually run 500 million instructions. Please note that the fast-forwarding instructions warm up the (micro-)architectural states in the processor (e.g., caches, many buffer structures, etc.), leading to more accurate simulation results.

For energy evaluation, L1 data cache energy and power results are derived from CACTI cache modeling tool [19] with 32nm process technology parameters. Though we use 32nm technology node instead of 11nm (used in our yield analysis), we modified circuit parameters (e.g., supply voltage, threshold voltage, etc.) in our CACTI tool to implement near-threshold operations. In addition, we consider line migration energy overhead. We assume that migration energy is same as an addition of two times of L1 cache read access dynamic energy with two times of L1 cache write access dynamic energy (L1 cache read dynamic energy $\times 2+$ L1 cache write dynamic energy $\times$ 2). This is because we need two read operations and two write operations to the caches for line migration. Though there are one additional read and write operations to the line migration buffer, due to its small size, its energy consumption is negligible as compared to the read or write operations to the caches. For the additional logic energy overheads, the non-negligible energy overheads are attributed to the latency table while the others are negligible. Thus, we estimate the latency table overhead by using CACTI (with 256B small SRAM buffer). For conservative estimation of the latency table, we assume that 4-bit latency information is maintained for each cache line in our evaluation. Table 4 summarizes energy and power parameters used in our energy evaluations. For energy consumption of the L1 data cache, with the cache access statistics (such as access counts of the L1 data caches, the number of line migrations, etc.) from the M-SIM and power and energy parameters from CACTI, we also derive the entire energy consumption of the L1 data caches.

TABLE 4. Energy parameters used for our energy evaluations. Dynamic write energy is scaled from the dynamic read energy according to the 8T SRAM array read and write energy results from [22].

Categories	L1 data cache	Additional logic (Latency table)
Leakage power (nW)	7,978,620	4,700
Dynamic read/write energy (nJ)	0.018374 / 0.021457	0.000826
Line migration energy (nJ)	0.079662	

B. YIELD

As we explained in Section 3.C, yield results are dependent on how many bits are used for storing latency information in the case of variable latency L1 data cache. We show yield results by varying the number of latency information bits. VL_2b, VL_3b, and VL_4b correspond to the cases where our variable latency cache is employed with 2-bit, 3-bit, and 4-bit latency information per cache line, respectively. In the case of using variable latency cache, the processor clock frequency is set to the nominal clock frequency (1GHz). We also compare yield when employing the frequency binning (i.e., loosely setting the timing constraint while maintaining the cache access latency as 3 cycles). 'FB $>$ x' means that we count the chip as a usable chip if the clock frequency of the chip is over 'x' GHz.

As shown in Table 5, our VL schemes show considerably higher cache yield compared to the frequency binning (FB). Cache data array access latency is hugely increased by process variation, resulting in huge yield losses even with frequency binning. When targeting the clock frequency over 0.6GHz with frequency binning, yield is nearly 0%. With much looser timing constraints (FB $>$ 0.2), one can obtain yield over 95% in the case of 'var10'. In contrast, only with 3-bit latency information per cache line (i.e., with VL_3b), 100% yield is achieved. Even though we gain considerable yield with frequency binning, we should significantly sacrifice performance to gain higher yield. In the case of applying frequency binning, the entire clock cycle time is increased, negatively affecting performance of the whole set of the instructions. In contrast, only memory-side performance (load/store instructions) is negatively affected in the case of VL_2b, VL_3b, and VL_4b.

In the super-threshold computing (STC) regime, one may be able to gain considerable yield and performance only with the frequency binning. In the NTC regime, however, only frequency binning is not sufficient for both yield and performance due to larger fluctuations on cache access latencies by process variations than in the STC regime.

C. PERFORMANCE

We compare performance among different schemes: baseline, FB_var10_max, FB_var15_max, FB_var20_max, WBD (weak block disabling), VL_base, and VL_mig. Baseline means the ideal case (it hardly exists considering yield) where L1 data cache access is performed with 3 cycles and processor clock frequency is 1GHz. FB_var10_max, FB_var15_max, and FB_var20_max mean the cases where we pick a processor which has the highest available clock frequency among 1000 processors for each of three process

TABLE 5. Yield results of various schemes across three process variation levels.

	FB>0.2	FB>0.4	FB>0.6	FB>0.8	VL_2b	VL_3b	VL_4b
Var10	97.6%	21.3%	0.5%	0.0%	86.3%	100.0%	100.0%
Var15	48.3%	1.8%	0.0%	0.0%	25.9%	86.2%	98.9%
Var20	12.9%	0.1%	0.0%	0.0%	5.1%	44.7%	83.9%

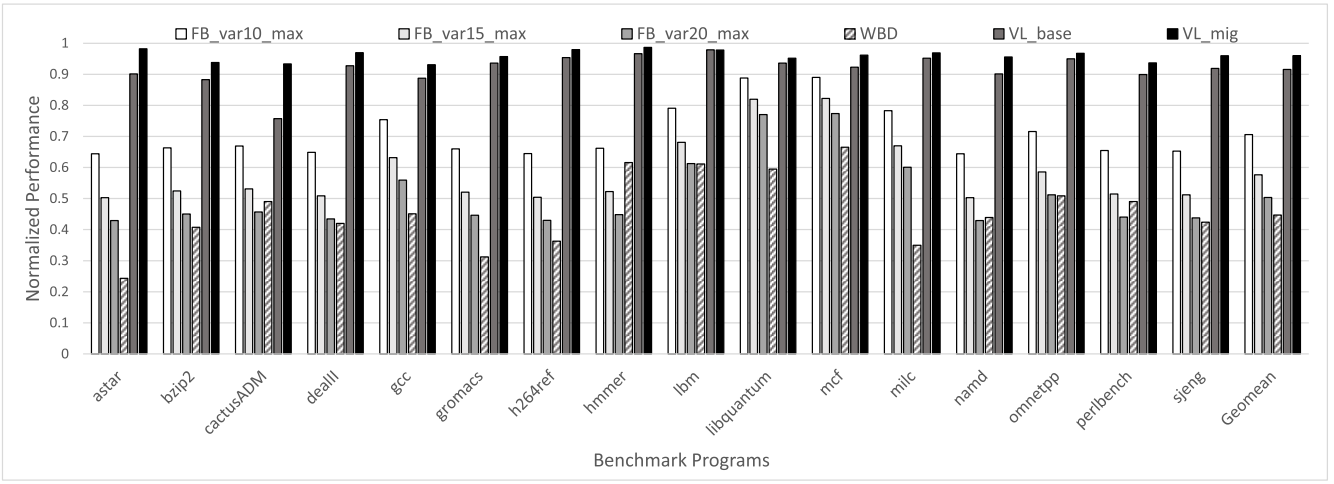

FIGURE 9. Performance results across frequency binning (FB_var10_max, FB_var15_max, FB_var20_max), weak block disabling (WBD), and our variable latency caches (VL_base, and VL_mig). The results are normalized to the baseline.

variation levels. Note that the highest available clock frequencies for var10, var15, and var20 from our Monte Carlo simulations are 644MHz, 503MHz, and 429MHz, respectively. Thus, in the cases of FB_var10_max, FB_var15_max, and FB_var20_max, we assume that the processor runs with 644MHz, 503MHz, and 429MHz, respectively, with 3-cycle L1 data cache access latency. WBD disables the faulty cache lines (blocks). Please note that this technique is widely used, simple, practical, and also introduced in [8] (similar to Intel Pellston Technology), [9], and [10] (introduced as a naïve way reduction scheme). Though this scheme could always lead to 100% cache yield by employing the adaptive cache bypassing (i.e., bypasses cache if there is no non-faulty blocks in a cache set), the effective cache capacity would be significantly reduced, eventually causing performance losses. We assume that there are 90% of the cache lines in the cache are faulty (a cache line is determined as faulty if there is at least one delay failing SRAM cell). This assumption is based on our Monte Carlo simulation. Though the average cache line-level fault rate is 97.3% in our Monte Carlo simulation, we conservatively set the cache fault rate as 90% in our simulation. VL_base and VL_mig correspond to the cases where variable latency cache is employed without and with the adaptive migration technique, respectively. Please note that we measure performance by using the inverse of the execution time.

Figure 9 depicts performance results across various schemes. Due to the increased cache access cycles, VL_base shows 8.4% worse performance compared to the baseline. In the case of VL_mig, performance degradation from the ideal case is only 4.0% which means our migration technique recovers a significant portion of the performance loss caused by VL_base. On the other hand, in the case of FB_var10_max, FB_var15_max, and FB_var20_max, performance degradation is roughly proportional to the clock frequency degradations which correspond to 29.4%, 42.4%, and 49.6% of performance losses. It means our VL_base and VL_mig are much more performance-efficient than the frequency binning. Since memory latency is not dependent on the processor clock frequency, memory-intensive workloads (e.g., mcf) shows relatively less performance losses compared to the other computation-intensive workloads in the cases of frequency binning (FB_var10_max, FB_var15_max, and FB_var20_max). In addition, our VL_mig shows better performance by 2.1X, on average, as compared to the WBD technique. The main reason is because of the reduced L1 data cache capacity in WBD, increasing the cache miss rate and lower-level cache and memory accesses.

Among the benchmark programs, CactusADM shows more huge performance degradation in the case of VL_base compared to the other benchmark programs. This is because CactusADM has a significantly higher portion of load instructions among the entire set of instructions. Load instructions make dependent instructions not issued until the data is delivered from the cache or main memory. In the case of variable latency cache, L1 cache access cycles can be increased, resulting in more stall cycles in the processor pipeline.

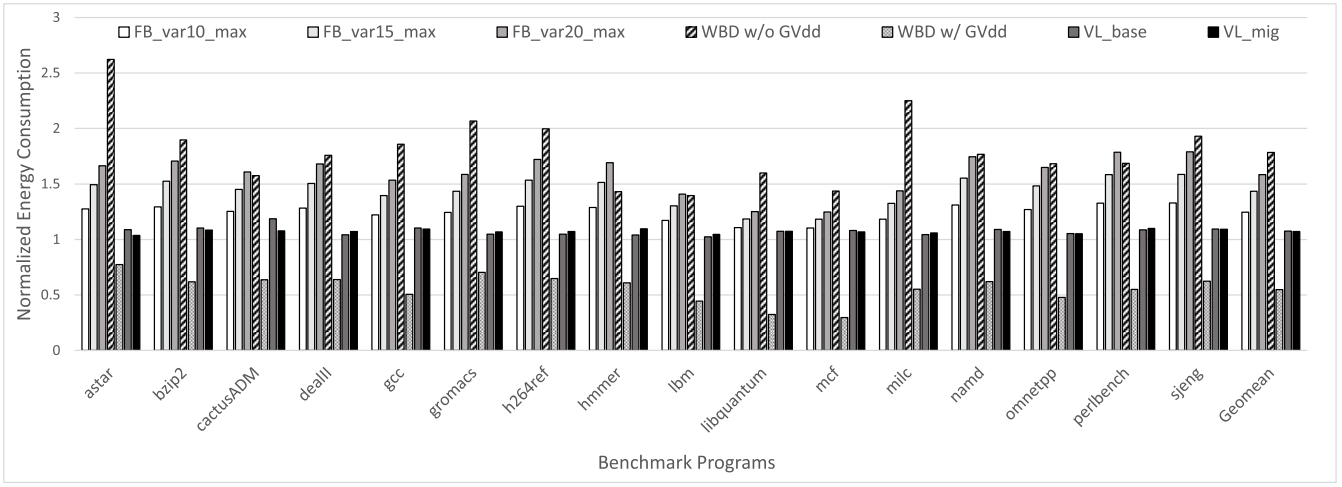

FIGURE 10. Energy consumption results across FB_var10_max, FB_var15_max, FB_var20_max, weak block disabling (WBD) without Gated-Vdd (WBD w/o GVdd), WBD with Gated-Vdd (WBD w/ GVdd), and our variable latency caches (VL_base, and VL_mig). The results are normalized to the baseline.

D. ENERGY

Figure 10 summarizes normalized energy comparison of L1 data cache across FB_var10_max, FB_var15_max, FB_var20_max, WBD without Gated-Vdd, WBD with Gated-Vdd, VL_base, and VL_mig. For energy comparison, we further classify the WBD technique into the cases where the Gated-Vdd [23] is applied (WBD w/ Gated-Vdd) and not applied (WBD w/o Gated-Vdd). In the case of WBD w/ GVdd, the disabled cache block is powered off to reduce leakage power consumption (as in [9] and [10]) while the WBD w/o Gated-Vdd (WBD w/o GVdd) does not apply the Gated-Vdd. Compared to the ideal case (i.e., baseline), VL_base and VL_mig show energy overheads of only 7.5% and 7.3%, respectively. Even with the migration energy overhead, VL_mig shows lower energy consumption compared to VL_base. This is because VL_mig shows a shorter execution time (i.e., better performance) than VL_base. It leads to less leakage energy consumption, which results in better energy efficiency of VL_mig (supported by Figure 11). In the case of FB_var10_max, energy consumption is higher than VL_base and VL_mig by 15.9% and 16.1%, respectively. It means the frequency binning is far less energy-efficient compared to the variable latency cache due to high leakage energy consumption (please refer to Figure 11). As process variation gets more severe (e.g., var20), the situation will get further worse, resulting in 47.8% more energy consumption compared to VL_mig. Our VL_mig still shows lower energy consumption by 39.9% as compared to the WBD w/o GVdd. On the other hand, the WBD w/ GVdd shows lower energy consumption by 48.9% compared to our VL_mig. Since the WBD w/ GVdd turns off the faulty cache lines (90% cache blocks are power-gated), it leads to huge leakage energy reduction. However, applying the Gated-Vdd increases the hardware complexity and our VL_mig shows much better performance (by 2.1X) as compared to the WBD techniques. Considering the trade-off between performance and energy,

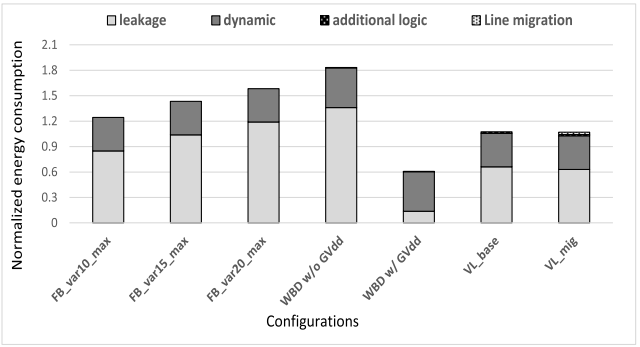

FIGURE 11. Energy breakdown results across FB_var10_max, FB_var15_max, FB_var20_max, weak block disabling (WBD) without Gated-Vdd (WBD w/o GVdd), WBD with Gated-Vdd (WBD w/ GVdd), and our variable latency caches (VL_base, and VL_mig). The results are normalized to the baseline.

our VL_mig leads to lower EDP (energy-delay product) by 8.8% as compared to the WBD w/ GVdd, meaning that our VL_mig is more energy-efficient than the WBD w/ GVdd.

V. RELATED WORK

Process variation is a huge hurdle for both super-threshold and near-threshold computing. For STC (super-threshold computing), many techniques have been proposed for variation-aware cache design. Ozdemir et al. proposed yield-aware cache architecture [4]. Their new cache architecture improves yield by employing variable latency caches and power-down techniques. In [17], to efficiently handle access time failures in L1 caches, Pan et al. proposed a selective wordline boosting technique for yield improvement. Kong et al. proposed an SRAM cell Arrays Voltage Lowering (SAVL) technique which reduces both delay- and leakage-induced yield losses [18]. Except for the studies introduced above, many other techniques [3], [24]–[26] have been proposed for variation-aware cache design since 6T

SRAM cells are known to be very vulnerable to process variations.

Variable latency cache architecture has been explored for performance improvement [5] and process variation tolerance [4], [6] in super-threshold computing regime. In [27], they group the cache lines so that the cache lines in a group have same access cycles, which eventually minimizes performance losses from the increased cache access latency due to process variations. In [15], a timing aware (TA)-LRU policy with variable latency caches were proposed. The TA-LRU shares an important insight with our proposed technique (in most cache accesses, the MRU line is likely to be accessed). However, the technique proposed in [15] is a kind of pseudo-LRU that selects a victim from several candidate lines for replacement. It may lead to sub-optimal cache replacement while our proposed technique always selects LRU line as a victim. In addition, the variable latency cache in [15] can only consider 1-cycle and 2-cycle cache latency while our proposed scheme considers a wider range of cache access cycles under process variations.

For process variation-awareness in NTC, a new L1 cache architecture that employs variation-robust SRAM cells was proposed [28]. Dreslinski et al. also proposed a dual-voltage supplied multi-core architecture in [29] for variation-awareness in NTC. Miller et al. proposed a dual supply voltage rail with Booster framework [30] and half speed units [31] for process variation robustness. EnergySmart was proposed by Karpuzcu et al. [12], which assigns the task into cores (clusters) by considering the variation-induced characteristics of cores (clusters). In [2], Kong et al. proposed CBoost and SWBoost which selectively boost supply voltages of cores in many-core processors for NTC. It significantly improves yield of the many-core processors with a negligible energy overhead. Though CBoost and SWBoost enable yield improvement with compromised clock frequencies, those techniques should still sacrifice clock frequency for comparable yield, which may lead to performance losses and energy-inefficiency.

There have also been many researches on cache architecture to support low V_{cc} (supply voltage) operations. In order to guarantee reliable operations in ultra-low voltages with the fixed yield loss target, several works try to leverage ECC (error correction code) with fine-grained management within a block [32] and data compression [33]. In [34], though ECC is not used, compression-based cache management is proposed to maximize an effective cache capacity under high fault rate in low V_{cc} operations. In [35], disabling and remapping techniques for faulty rows or blocks (or subblocks) to tolerate high bit error rates in low V_{cc} operations were proposed. Block or subblock-level disabling and remapping techniques were also proposed in [36] and [37]. A recent implementation of RISC-V processor employs a line recycling technique [38] that combines three faulty cache lines which are recycled to compose one non-faulty cache line. The above mentioned disabling-based techniques could be used when the process variation severity is low. However, under the severe process variation (as we show in our evaluation), the effective cache capacity could be significantly low, leading to severe performance losses.

For another approach for low V_{cc} operations in caches or SRAM arrays, the spare blocks [39] (i.e., redundancy-based scheme) are used to replace the faulty cache blocks to enable reliable cache operations under ultra-low V_{cc}. ZCAL cache [40] was also proposed to prevent access time failure-induced malfunction caused by ultra-low voltage operations. However, in [40], the fault detection is performed in run-time, causing performance overhead while ours is performed offline (after manufacturing or at booting time) without causing a performance overhead from the latency testing. Several works also utilized mixed-cell cache architecture to tolerate failures in low voltage operations. In [41], two types of the cells (robust and non-robust cells) are used and cache allocation and migration scheme exploits the characteristics of the robust and non-robust cells. Though the line migration scheme is also used in [41], our line migration scheme aims to reduce the cache access latency under the severe timing failures due to process variations. In contrast, the cache allocation and migration scheme in [41] only considers the type of cache misses (read or write misses), meaning that the latency-related characteristics are not considered. In [42] and [43], the cross-sensing-based SRAM arrays and cache management techniques are proposed, respectively. However, using a new type of SRAM cells and arrays restricts the applicability of the techniques while ours can be applied to any type of the SRAM cells and arrays.

Compared to the studies introduced so far, the novelty of our work is: 1) revisiting and revising the conventional variable latency cache for yield improvement in NTC; 2) performing the detailed analysis on yield by comparing with the frequency binning; 3) proposing a novel line migration technique for performance and energy efficiency which is inspired from L1 data cache locality characteristics; 4) maximizing yield with an efficient performance and energy trade-off.

VI. CONCLUSION

In this paper, we proposed a yield-aware variable latency cache architecture with the adaptive migration technique. Compared to the cases where there is no preventive technique and frequency binning, our proposed technique significantly improves yield of near-threshold L1 data cache. Moreover, compared to the frequency binning, our proposed technique significantly improves performance and energy efficiency with very low hardware overhead.

Though near-threshold computing has emerged for energy efficiency, without yield-aware techniques, it would not be possible to carry out a mass production of near-threshold processors. As shown in this paper, our proposed technique can be a good design candidate for near-threshold processors to enable a mass production of near-threshold computing by improving yield, performance, and energy efficiency. As our future work, we will 1) investigate a cache size sensitivity

of our technique, 2) apply and modify our technique to be geared towards L1 instruction caches, and 3) investigate the relationship among the cache physical layout, yield, energy, and performance impacts.

REFERENCES

[1] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas, "VARIUS-NTV: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2012, pp. 1–11.

[2] J. Kong, A. Munir, and F. Koushanfar, "Fine-grained voltage boosting for improving yield in near-threshold many-core processors," in *Proc. 25th Ed. Great Lakes Symp. VLSI (GLSVLSI)*, 2015, pp. 225–228.

[3] A. Agarwal, B. Paul, H. Mahmoodi, A. Datta, and K. Roy, "A process-tolerant cache architecture for improved yield in nanoscale technologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 1, pp. 27–38, Jan. 2005.

[4] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou, "Yield-aware cache architectures," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 15–25.

[5] S. Ozdemir, A. Mallik, J. C. Ku, G. Memik, and Y. Ismail, "Variable latency caches for nanoscale processor," in *Proc. ACM/IEEE Conf. Supercomput. (SC)*, Nov. 2007, p. 20.

[6] M. Mutyam, F. Wang, R. Krishnan, V. Narayanan, M. Kandemir, Y. Xie, and M. J. Irwin, "Process-variation-aware adaptive cache architecture and management," *IEEE Trans. Comput.*, vol. 58, no. 7, pp. 865–877, Jul. 2009.

[7] A. Datta, S. Bhunia, J. H. Choi, S. Mukhopadhyay, and K. Roy, "Profit aware circuit design under process variations considering speed binning," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 7, pp. 806–815, Jul. 2008.

[8] C. McNairy and J. Mayfield, "Montecito error protection and mitigation," in *Proc. 1st Workshop High Perform. Comput. Rel. Issues Conjunct. (HPCA)*, 2005.

[9] J. Kong and S. W. Chung, "Exploiting narrow-width values for process variation-tolerant 3-D microprocessors," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, 2012, pp. 1193–1202.

[10] J. Kong, F. Koushanfar, and S. W. Chung, "An energy-efficient last-level cache architecture for process variation-tolerant 3D microprocessors," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2460–2475, Sep. 2015.

[11] *The R Project for Statistical Computing*. [Online]. Available: http://www.r-project.org/

[12] U. R. Karpuzcu, A. Sinkar, N. Sung Kim, and J. Torrellas, "EnergySmart: Toward energy-efficient manycores for near-threshold computing," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2013, pp. 542–553.

[13] *ARM Cortex-A15*. Accessed: Jun. 2013. [Online]. Available: http://www.arm.com/products/processors/cortex-a/cortex-a15.php

[14] G. Chen, D. Sylvester, D. Blaauw, and T. Mudge, "Yield-driven near-threshold SRAM design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 11, pp. 1590–1598, Nov. 2010.

[15] P.-H. Wang, W.-C. Cheng, Y.-H. Yu, T.-C. Kao, C.-L. Tsai, P.-Y. Chang, T.-J. Lin, J.-S. Wang, and T.-F. Chen, "Variation-aware and adaptive-latency accesses for reliable low voltage caches," in *Proc. IFIP/IEEE 21st Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2013, pp. 358–363.

[16] R. Rajsuman, "Design and test of large embedded memories: An overview," *IEEE Design Test Comput.*, vol. 18, no. 3, pp. 16–27, May/Jun. 2001.

[17] Y. Pan, J. Kong, S. Ozdemir, G. Memik, and S. W. Chung, "Selective word-line voltage boosting for caches to manage yield under process variations," in *Proc. 46th Annu. Design Autom. Conf. (DAC)*, 2009, pp. 57–62.

[18] J. Kong, Y. Pan, S. Ozdemir, A. Mohan, G. Memik, and S. W. Chung, "Fine-grain voltage tuned cache architecture for yield management under process variations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 8, pp. 1532–1536, Aug. 2012.

[19] N. Muralimanohar and R. Balasubramonian, "CACTI 6.0: A tool to model large caches," HP Lab., Palo Alto, CA, USA, Tech. Rep., 2009.

[20] J. J. Sharkey, D. Ponomarev, and K. Ghose, "M-Sim: A flexible, multi-threaded architectural simulation environment," Dept. Comput. Sci., State Univ., Binghamton, NY, USA, Tech. Rep. CS-TR-05-DP01, 2005.

[21] *SimpleScalar Toolset*. [Online]. Available: http://www.simplescalar.com

[22] P. Singh and S. K. Vishvakarma, "Ultra-low power high stability 8T SRAM for application in object tracking system," *IEEE Access*, vol. 6, pp. 2279–2290, 2018.

[23] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijayku, "Gated-V_{dd}: A circuit technique to reduce leakage in deep-submicron cache memories," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2000, pp. 90–95.

[24] A. Das, S. Ozdemir, G. Memik, J. Zambreno, and A. Choudhary, "Microarchitectures for managing chip revenues under process variations," *IEEE Comput. Archit. Lett.*, vol. 6, no. 2, pp. 29–32, Feb. 2007.

[25] K. Meng and R. Joseph, "Process variation aware cache leakage management," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2006, pp. 262–267.

[26] B. F. Romanescu, M. E. Bauer, S. Ozev, and D. J. Sorin, "Reducing the impact of intra-core process variability with criticality-based resource allocation and prefetching," in *Proc. Conf. Comput. Frontiers*, 2008, pp. 129–138.

[27] M. Mutyam and N. Vijaykrishnan, "Working with process variation aware caches," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, vol. 2007, pp. 1152–1157.

[28] R. G. Dreslinski, G. K. Chen, T. Mudge, D. Blaauw, D. Sylvester, and K. Flautner, "Reconfigurable energy efficient near threshold cache architectures," in *Proc. 41st IEEE/ACM Int. Symp. Microarchitecture*, Nov. 2008, pp. 459–470.

[29] R. G. Dreslinski, B. Zhai, T. Mudge, D. Blaauw, and D. Sylvester, "An energy efficient parallel architecture using near threshold operation," in *Proc. 16th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2007, pp. 175–188.

[30] T. N. Miller, X. Pan, R. Thomas, N. Sedaghati, and R. Teodorescu, "Booster: Reactive core acceleration for mitigating the effects of process variation and application imbalance in low-voltage chips," in *Proc. IEEE 18th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2012, pp. 27–38.

[31] T. N. Miller, R. Thomas, and R. Teodorescu, "Mitigating the effects of process variation in ultra-low voltage chip multiprocessors using dual supply voltages and half-speed units," *IEEE Comput. Arch. Lett.*, vol. 11, no. 2, pp. 45–48, Jul. 2012.

[32] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S. Lu, "Improving cache lifetime reliability at ultra-low voltages," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2009, pp. 89–99.

[33] C. Yan and R. Joseph, "Cocoa: Synergistic cache compression and error correction in capacity sensitive last level caches," in *Proc. Int. Symp. Memory Syst. (MEMSYS)*, 2018, pp. 117–128.

[34] A. Ferreron, D. Suarez-Gracia, J. Alastruey-Benede, T. Monreal-Arnal, and P. Ibanez, "Concertina: Squeezing in cache content to operate at near-threshold voltage," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 755–769, Mar. 2016.

[35] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. González, "Low Vccmin fault-tolerant cache with highly predictable performance," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2009, pp. 111–121.

[36] A. Ansari, S. Feng, S. Gupta, and S. Mahlke, "Archipelago: A polymorphic cache design for enabling robust near-threshold operation," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2011, pp. 539–550.

[37] T. Mahmood, S. Kim, and S. Hong, "Macho: A failure model-oriented adaptive cache architecture to enable near-threshold voltage scaling," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit. (HPCA)*, vol. 2013, pp. 532–541.

[38] P. Chiu, C. Celio, K. Asanović, D. Patterson, and B. Nikolić, "An out-of-order RISC-V processor with resilient low-voltage operation in 28 nm CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 61–62.

[39] N. A. Siddique and A.-H. A. Badawy, "SprBlk cache: Enabling fault resilience at low voltages," in *Proc. Int. Symp. Memory Syst. (MEMSYS)*, 2017, pp. 130–140.

[40] P.-H. Wang, W.-C. Cheng, Y.-H. Yu, T.-C. Kao, C.-L. Tsai, P.-Y. Chang, T.-J. Lin, J.-S. Wang, and T.-F. Chen, "Zero-counting and adaptive-latency cache using a voltage-guardband breakthrough for energy-efficient operations," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 63, no. 10, pp. 969–973, Oct. 2016.

[41] S. M. Khan, A. R. Alameldeen, C. Wilkerson, J. Kulkarni, and D. A. Jimenez, "Improving multi-core performance using mixed-cell cache architecture," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2013, pp. 119–130.

[42] S. Shen, T. Shao, X. Shang, Y. Guo, M. Ling, J. Yang, and L. Shi, "TS cache: A fast cache with timing-speculation mechanism under low supply voltages," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 252–262, Jan. 2020.
[43] M. Ling, X. Shang, S. Shen, T. Shao, and J. Yang, "Lowering the hit latencies of low voltage caches based on the cross-sensing timing speculation SRAM," *IEEE Access*, vol. 7, pp. 111649–111661, 2019.

JOONHO KONG (Member, IEEE) received the B.S. degree in computer science and the M.S. and Ph.D. degrees in computer science and engineering from Korea University, in 2007, 2009, and 2011, respectively. He is currently an Assistant Professor with the School of Electronics Engineering, Kyungpook National University. His research interests include computer architecture design, processor cache design, hardware security, and heterogeneous computing.

JAE YOUNG HUR (Member, IEEE) received the B.S. degree in electronics engineering from Cheju National University, Cheju, South Korea, in 1995, the M.S. degrees in electronics engineering from Sogang University, Seoul, South Korea, in 1998, and the Munich University of Technology, Munich, Germany, in 2002, and the Ph.D. degree in computer engineering from the Delft University of Technology, Delft, The Netherlands, in 2011. From 1999 to 2000, he was an Engineer with the Semiconductor Division, Samsung Electronics Ltd., South Korea, where he was also a Senior Engineer, from 2008 to 2016. He is currently a Researcher with Vietnamese-German University, Binh Duong, Vietnam. His research interests include embedded system architectures, VLSI design, and reconfigurable computing.