eBaRe: An Efficient Backup and Restore Techniques in Hybrid L-1 Cache for Energy Harvesting Devices

SatyaJaswanth Badri\textsuperscript{1*}, Mukesh Saini\textsuperscript{1} and Neeraj Goel\textsuperscript{1}

\textsuperscript{1}Computer Science and Engineering, IIT Ropar, Rupnagar, 140001, Punjab, India.

*Corresponding author(s). E-mail(s): 2018csz0002@iitrpr.ac.in; mukesh@iitrpr.ac.in; neeraj@iitrpr.ac.in;

Abstract

Battery operated devices are rapidly increasing due to the bulk usage of IoT enabled nodes in various fields. The alternative and promising solution to replace battery-operated devices is energy harvesters, which helps to power up the embedded devices. The energy harvester systematically stores sufficient energy in a capacitor to power up the embedded device for computing the task. This type of computation is defined as intermittent computing. Energy harvesters cannot ensure a continuous power supply for embedded devices. All registers and cache are volatile in conventional processors. We require a Non-Volatile Memory (NVM) based Non-Volatile Processor (NVP), which store the registers and cache contents during power failure. Introducing NVM at the cache level degrades the system performance and consumes more energy than SRAM based caches. In this paper, an Efficient Backup and Restore (eBaRe) hybrid cache architecture is proposed by integrating SRAM and STT-RAM at a first-level cache. The eBaRe architecture proposes cache block placement and migration policies to reduce the number of writes to STT-RAM. During a power failure, the backup strategy finds the important blocks to migrate from SRAM to STT-RAM. In comparison to baseline architecture, eBaRe architecture reduces STT-RAM writes from 63.35\% to 35.93\%, resulting in a 32.85\% performance gain and 23.42\% reduction in energy consumption. Our backup strategy decreases backup time by 34.46\% compared with baseline.
1 Introduction

Internet of things (IoT) has created several fascinating applications that consist of intelligent sensors and systems. IoT will consist of billions of sensors and systems by the end of 2050 [1]. This prediction may be promising and exciting, but the main challenge is how to power these IoT devices. Most of the IoT devices are battery-operated. It is difficult and expensive to replace batteries after installation in some areas like deep mines, space and industrial environments.

Moreover, the battery has a specific issue of its lifetime [2]. Therefore, the alternative and promising solution is to use energy-harvesters, which harvest ambient energy to power up an IoT device. Energy-harvesting devices extract energy from resources available in their environment, e.g., light, vibration, radio, and many others.

The unpredictable nature of energy harvesters causes voltage fluctuation or power failure. A common approach to solve the voltage fluctuation issue is to use a voltage stabilizer or a capacitor. However, the power failures lead to data loss in a conventional processor. The data is lost because registers, caches, and main memories are implemented using SRAM/DRAM, which is volatile in nature [3], [4]. Data lost includes the program state and progress of the application. Program state includes the content of registers, cache, and main memory. Therefore, power failure causes re-executing some parts of the application, making the application’s execution progress incremental.

The solution is to store the application’s program state at a precise restart point before a power failure. The question is, where to store the program state? Due to their persistent nature, Non-volatile memories (NVM) may be useful to save the application’s program state in the presence of power failures. There are several emerging NVM’s such as spin-transfer torque RAM (STT-RAM) [5], phase-change memory (PCM) [6], ferroelectric RAM (FRAM) [7].

Researchers have explored NVM based Non-volatile processor (NVP) [8], which helps in performing the application’s execution under unpredictable energy supply. Using NVM cache improves the application’s execution progress during the intermittent power supply. NVM’s have higher read latency and longer write latency than the SRAM based caches. Replacement of SRAM based cache with pure NVM based cache is not a good idea. The hybrid solution has been considered in the past [9] [10]. Xie et al. [9] propose a hybrid cache architecture to achieve energy efficiency for intermittent power supply processors.

This paper builds on previous work [9] to improve the following aspects for a hybrid cache (a) performance, (b) utilize energy efficiently (c) reducing writes to NVM. In conventional processors, main memory is implemented
using DRAM, but in emerging micro-controllers, like MSP430FR5969 [7] main memory is non-volatile. Therefore, we have chosen STT-RAM as an NV-cache and PCM as an NV-main memory throughout this paper.

The proposed system of an efficient Backup and Restore (eBaRe) hybrid cache architecture comprises SRAM and STT-RAM at the L1 cache. This architecture is proposed for energy-harvesting IoT applications. During a power failure, we assume that a capacitor’s energy can backup the processor state [11]. Since SRAM cache and SRAM based registers are volatile, their values need to be stored in NVM. eRaBe identifies the blocks that need to be written to the main memory and STT-RAM at the L1 cache to reduce backup time.

Further, the exception mechanism of the pipelined processor is used to arrive at a precise wake-up point. When power comes back, our proposed architecture works like a regular architecture, i.e., every memory access is first looked up in the L1 cache. Since STT-RAM is present at the L1 cache, stores frequently accessed blocks, and the proposed architecture has an efficient restore.

We also proposed a prediction table, which helps during block placement and migration. The proposed architecture results in a 32.85% performance gain and reduces energy consumption by 23.42% compared with the baseline architecture. Our proposed backup strategy decreases backup time by 27.91% compared with baseline. The storage overhead of the proposed architecture is only 2.34%.

This paper is organized as follows: Section 2 discusses the related works. Section 3 explains the motivation behind the proposed system architecture and gives an overview of the problem formulation. Section 4 explains about proposed hybrid cache architecture. The experimental setup and results are described in section 5. We conclude this paper in section 6.

2 Related works

This section reviews the related work in Hybrid cache Architectures (HCA) and NVM for Last-level caches (LLC), HCA for L1 cache, and architectures for intermittent power systems.

2.1 NVM for LLC:

Most of the literature cites STT-RAM as an emerging candidate among all NVM in LLC [12], [13], [14]. STT-RAM promises higher density and less leakage power than existing SRAM caches. There are two different choices for using STT-RAM at LLC. First, by replacing the whole SRAM cache with STT-RAM at LLC. Wu et al. modelled a 3-level cache architecture, which uses STT-RAM at L3 cache, SRAM at L1 and L2. This architecture achieved instructions per cycle (IPC) improvement of around 4%, 63% reduction in power consumption when compared with a 3-level SRAM cache design [15]. Second, using
HCA (SRAM+STT-RAM) at LLC, which take advantage of both SRAM and STT-RAM.

Normally for hybrid caches, block placement and movement is a challenge. Classification of the blocks based on write frequencies [8], and write access behaviour [16] [17] [18] [19] helps in block placement. Many architectures with the HCA uses prediction table-based techniques to place the block and migrate from one cache to another [18]. Challenges at L1 HCA is different from LLC. At LLC, input traffic is due to misses of L1/L2, while read/write requests at L1 is due to Load/store instructions.

2.2 HCA for L1 caches:

The high access latency of STT-RAM is the primary limiting factor of its use at the L1 cache. For this concern, there are two choices. First, relaxing the non-volatility of STT-RAM to reduce the access latency [13], [14]. Non-volatility relaxation is achievable by reducing the MTJ planar area and MTJ switching current [20]. Second, reducing the STT-RAM’s write energy consumption by reducing the number of writes to STT-RAM. Usually, the number of read and write operations in the L1 cache is more than in the LLC.

Xie et al. [9] introduces an HCA that consists of STT-RAM in the L1 cache. They backup the program state from the SRAM cache to the STT-RAM cache during power failures. Authors use an access pattern-based predictor that predicts the block behaviour. Based on the prediction, block placement and migration takes place from one cache region to another. Whenever power comes back, Xie et al. restore the cache contents from STT-RAM cache to SRAM cache.

In these hybrid caches, if we wrongly place a block in any of cache region causes migration overhead. Migration overhead increases the number of writes to NVM and write energy. By introducing NVM in the L1 cache shows an impact on performance and energy consumption. Therefore, we proposed an HCA to efficiently address the above issues using SRAM and STT-RAM to reduce overheads as much as possible. We proposed a placement policy, which uses a prediction table to reduce overheads mentioned above. Better prediction and migration policies help us to reduce migration overhead and write to NVM.

2.3 Architectures for Intermittent power devices:

NVM based NVP [21], [22], [23] are proposed by storing register, volatile on-chip data into non-volatile register, non-volatile memories respectively. Whenever power comes back, the system uses NVM data to continue and complete the application execution.

Another approach for designing intermittent power devices is checkpoint based HCA. This approach checkpoints volatile data to NVM to preserve the application program state [24]. Mementos [25] was one of the initial checkpointing schemes. It used periodic voltage checks to decide when to back up the program state. Hibernus [26] extended the work in [25] by introducing NVM.
These checkpointing schemes consider the timely execution of the applications. TICS [27] overcomes this problem by introducing timely execution, branching and efficient automatic checkpoints.

Checkpoints are placed by using either software procedures or hardware components. In related works, checkpointing approaches like [25], [28], [29] were proposed to backup and restore consistent program state. Software-based checkpoints were mainly placed by the compiler or software procedures. When a checkpoint is identified, the system triggers the backup procedure by saving the program state to NVM. In [29], checkpoints are placed based on the expiration of a timer. Hardware-based checkpoints were mainly associated with external devices. In [28], hardware-based checkpoints are placed/identified using a voltage detector to trigger a backup mechanism for an NVP.

The main issue in all the above architectures is the unnecessary computation due to multiple checkpoints. Another issue with the checkpointing during a power failure is data inconsistency. Inconsistency of data leads to a corrupted output. One more issue with the checkpointing approach is that we have to restore the contents from non-volatile main memory to cache whenever power comes back. We have to introduce a restoration procedure during a power failure case, restoring the saved checkpoint from NVM. Restoration of the program state introduces overhead. Instead of saving the program state at the desired checkpoints, we proposed an HCA with a backup policy that triggers during a power failure. Our proposed HCA uses an automatic restoration process instead of restoring the program state after a power failure.

3 Motivation and Problem Formulation

In this section, we discuss observations that motivate us to propose a new solution. We performed a set of experiments on a system configuration that consists of equal SRAM and STT-RAM at the L1 cache. In section 5, we have given more details of the experimental setup, and table 1 has architectural parameters.

3.1 Motivation

Introducing STT-RAM as a cache can deteriorate the system performance due to its long access time. STT-RAM write latency is ten times longer than its read latency [30]. We modelled two cache architectures in gem5 [31], pure SRAM cache (only SRAM at L-1) and pure STT-RAM cache (only STT-RAM at L-1), to compare their performances and energy consumptions.

We performed experiments for these two architectures. In the figure 1, the performance and energy consumption of the cache architectures are normalized based on the pure SRAM cache architecture. Figure 1 shows that STT-RAM cache architecture takes 45.93% more execution time in comparison to pure SRAM cache architecture. Our first motivation is to reduce the overhead caused because of pure STT-RAM cache.
Migration-based policies are introduced in the literature for hybrid caches [32], [33], [34], [35]. Migration needs extra cycles and energy for data movement because of their additional read and write operations. These additional overheads deteriorate the NVM based cache system performance and energy consumption. These overheads degrade system performance and consume more energy. Our second observation is to minimize these overheads as much as possible, which are due to migration.

These overheads motivated us to propose the prediction based HCA for the block placement. If we place a block in the correct cache region, we automatically reduce migrations between the caches. All these overheads motivate us to introduce an intelligent migration policy along with the placement prediction for NVM based HCA.

3.2 Problem definition

We propose a cache architecture based on the observations in the above section 3.1 as shown in fig 2. By introducing NVM at the L1 cache, we observed additional overheads, which were discussed in section 3.1. These extra overheads show an impact on both system performance and energy consumption, as explained in the section 3.1. So we reduce these overheads by introducing intelligent placement and migration policy to use NVM and SRAM efficiently. We formulate our three main objectives below, which reduce overheads mentioned in the above section.

- Minimize backup energy.
- Maximize backup efficiency.

Fig. 1: Comparisons between SRAM, and STT-RAM Architectures in-terms of Execution time and Dynamic Energy Consumption
• Maximize energy efficiency.

Our design performs a backup during a power failure for the hybrid architecture model, and whenever power comes back, performs a memory restore operation. Thus, for this system, we can represent the cost associated with energy as below.

\[ E_{\text{overall}} = E_{\text{exec}} + E_{\text{backup}} + E_{\text{restore}} \]  

Where \( E_{\text{overall}} \) is the energy required to execute the overall application, \( E_{\text{exec}} \) is the energy required to execute the program. We save the system state by copying all register contents and SRAM cache blocks to NVM. The energy consumed by this process, \( E_{\text{backup}} \), depends on the number of bytes to be backed up to NVM, can be written as follows:

\[ E_{\text{backup}} = N_{\text{w,L1}} \times e_{\text{w,DRAM}} + N_{\text{w,main}} \times e_{\text{w,PCM}} \]  

Where \( N_{\text{w,L1}} \) is the number of writes into STT-RAM, \( N_{\text{w,main}} \) is the number of writes to main memory, \( e_{\text{w,DRAM}} \) is the energy per write for the STT-RAM and \( e_{\text{w,PCM}} \) is the energy per write for PCM RAM. We achieve our first objective by reducing the number of writes, \( N_{\text{writes}} = N_{\text{w,L1}} + N_{\text{w,main}} \), during intermittent power supply.

Our second objective is to maximize backup efficiency (\( \eta \)), which is defined as follows:

\[ \eta = \frac{N_{\text{w,L1}}}{N_{\text{w,L1}} + N_{\text{w,main}}} \]  

If we achieve less \( N_{\text{writes}} \), our \( \eta \) increase. Thus, we achieve our second objective by reducing \( N_{\text{writes}} \). Lastly, we define energy efficiency as the ratio of energy used during normal execution without any power failures to the energy used with an intermittent power supply. Let \( \theta \) be the energy efficiency.

\[ \theta = \frac{E_{\text{normal}}}{E_{\text{overall}}} \]  

\( E_{\text{normal}} \) is the energy required for normal execution without any power interruptions.

4 Proposed eBaRe Architecture

This section explains the proposed eBaRe architecture that uses the proposed placement, migration, and backup policies.

4.1 Hybrid Cache Architecture

The proposed eBaRe architecture is shown in figure 3.

In the proposed eBaRe architecture, there is a combination of SRAM and STT-RAM cache blocks in every cache set. In each cache block, along with
valid bit (V), dirty bit (D), tag and data, we added three more entries i) RIC, ii) WIC, and iii) Confidence bits (CONF). These three entries help in cache placement and migration policies. We differentiated blocks into two categories, read-intensive and write-intensive blocks. Read-Intensive (RI) blocks are the blocks that have more read accesses than a predefined threshold at a particular point of time, and Write-Intensive (WI) blocks are the blocks, which have more write accesses than a predefined threshold at a particular point of time.

We maintain two counters for each block, Read-Intensive Counter (RIC) and Write-Intensive Counter (WIC). In addition, we introduced a 2-bit CONF field that tracks important blocks; important block information is helpful during power failure. We have also incorporated a prediction table. Each entry in the prediction table has a previous region (PR) bit. This PR bit is updated during replacement/eviction. PR bit stores the block’s last cache region.

4.2 Placement and Migration Policies

In this section, we describe the proposed block placement and migration policies. Since STT-RAM has more write latency and write energy, the placement policy’s objective is to reduce the number of writes to NVM. So, we would like to place write-intensive blocks in the SRAM cache and read-intensive blocks in the STT-RAM cache. We look into the prediction table to see if the block is read-intensive or write-intensive using the PR bit and insert the block in the corresponding cache.

Algorithm 1 demonstrates placement policy in case of a cache miss. Line 1 checks the prediction table on a read/write miss using a tag. We access the PR bit associated with the tag entry. We maintain PR bit to know the previous block placement for that tag entry. If PR=0, line 3-5 in algorithm 1 checks whether the corresponding STT-RAM cache set is full or not. If it is full, we replace the block with the lowest RIC value, otherwise place the block.
in the STT-RAM cache. If PR !\neq 0, line 10-12 in algorithm 1 checks whether the SRAM cache set is full or not. We replace the block with the lowest WIC value; else, we place the block in the SRAM cache.

Algorithm 2 describes placement and migration policies whenever there is a read hit. Line 1 checks the block’s RIC value with the empirically determined threshold. We fixed the threshold limit empirically. If the block’s RIC is equal to the threshold, we call that block an RI block. The proposed placement policy suggests that all RI blocks should place in STT-RAM. If the block is present in the SRAM cache, we migrate from SRAM to STT-RAM and re-initialize RIC, WIC, and CONF to zero. If the block is not in the SRAM cache, we place the block in the STT-RAM cache and increment CONF by 1. If the threshold is not equal to the block’s RIC value, we increment RIC by 1. The block which is chosen for replacement has to update its PR bit in the prediction table. If RIC reaches to threshold and CONF reaches 11 state, then we don’t increment RIC.

Algorithm 3 describes placement and migration policies whenever there is a write hit. Line 1 checks the block’s WIC value with the threshold. If block’s WIC is equal to the threshold, we call that block a WI block. The proposed
Algorithm 1 Placement algorithm in case of cache miss

1: Check Prediction Table.
2: if $PR == 0$ then
3:    if STT-RAM set is full then
4:        Replace block with lowest $b.RIC$
5:        Update the replaced block’s PR bit in Prediction Table.
6:    else
7:        Place in the STT-RAM cache.
8:        Re-Initialize $b.RIC$, $b.WIC$ to zero.
9:    end if
10: else
11:    if SRAM set is full then
12:        Replace block with lowest $b.WIC$.
13:        Update the replaced block’s PR bit in Prediction Table.
14:    else
15:        Place in the SRAM cache.
16:        Re-Initialize $b.RIC$, $b.WIC$ to zero.
17: end if
18: end if

Algorithm 2 Placement and Migration algorithm in case of read hit

1: if $b.RIC ==$ threshold then
2:    if Block is in SRAM then
3:        if STT-RAM set is full then
4:            Replace block with lowest $b.RIC$.
5:            Update the replaced block’s PR bit in Prediction Table.
6:            Migrate to STT-RAM.
7:        else
8:            Migrate to STT-RAM.
9:        end if
10:    end if
11:    $b.CONF = b.CONF + 1$
12:    Re-Initialize $b.RIC$, $b.WIC$, $b.CONF$ to zero.
13: end if
14: else
15:    $b.RIC = b.RIC + 1$
16: end if

placement policy suggests that all WI blocks should place in SRAM. If the block is already present in the STT-RAM cache, we migrate from STT-RAM to SRAM and re-initialize RIC, WIC, and CONF to zero. This case reduces the number of writes to the STT-RAM cache to minimize write energy/latency. If the threshold is not equal to the block’s WIC value, we increment WIC by 1. The block which is chosen for replacement has to update its PR bit in the prediction table. If WIC reaches to threshold and CONF reaches 11 state, then we don’t increment WIC.
Algorithm 3  Placement and Migration algorithm in case of write hit

1: if b.WIC == threshold then
2:   if Block is in STT-RAM then
3:     if SRAM set is full then
4:       Replace block with lowest b.WIC.
5:       Update the replaced block’s PR bit in Prediction Table.
6:       Migrate to SRAM.
7:     else
8:       Migrate to SRAM.
9:   end if
10: Re-Intialize b.RIC, b.WIC, b.CONF to zero.
11: end if
12: b.CONF = b.CONF + 1
13: Re-Intialize b.WIC to zero.
14: else
15:   b.WIC = b.WIC + 1;
16: end if

4.3 Prediction Table Design
The importance of the prediction table in eBaRe architecture is to store the previous region for the respective tag entry. The prediction table has L-entries, where L denotes the number of entries in the prediction table. This table acts as a direct-mapped buffer, indexed using (Address/block_size) % L. Each entry in the prediction table has a PR (Previous Region) field. The prediction table does not store the tag bits to save area, and its size is L bits. All bits of the prediction table is initialized to 1.

Whenever there is a replacement in the cache due to SRAM/STT-RAM set is full, we update its PR field. If PR is 1, then the block is a WIC because the block’s WIC is higher than RIC during the replacement. So place the WIC block in the SRAM cache. If PR is 0, then the block is a RIC because the block’s RIC is higher than WIC during the replacement.

4.4 Support for Intermittent Power Supply

eBaRe architecture works efficiently during intermittent power supply. We define important blocks are the blocks that have high CONF value. We update the CONF field using RIC/WIC values. In a conventional architecture, whenever power comes back, we start execution by accessing blocks from the main memory and copying them to cache. We save important blocks in STT-RAM that helps to start execution without restoring blocks from the main memory to SRAM.

We propose a state model, which helps to identify the important blocks. We know the necessary blocks that should be there in STT-RAM during a power failure using the CONF field. Initially, CONF is in 00 state and supports four states, i.e., 00, 01, 10, 11 states, as shown in figure 4. We require a 2-bit CONF field to represent the proposed state model. We have already explained the update process of the CONF field in algorithm 2 and 3. In summary, if
RIC/WIC crosses the threshold, CONF will be incremented by one and reaches to next state. Whenever CONF is in 11 state and crosses the threshold, then CONF stays in the same 11 state. If any migration happens from SRAM/STT-RAM to STT-RAM/SRAM cache, then CONF resets to 00 state along with the RIC, WIC values.

During a power failure, the proposed backup policy triggers to save important blocks from SRAM to STT-RAM. The blocks with the CONF field 11 are treated as the most important blocks in the proposed backup policy. Therefore, we prioritise blocks with the CONF field in the order of 11 > 10 > 01 > 00. If any SRAM block has a CONF field as 11, we replace that SRAM block with the least priority block in STT-RAM. If there is no block with 11 state in the SRAM, we decrement our priority order by 1.

Now our priority becomes 10. If there are blocks with 10 state in the SRAM cache line, we replace the blocks with the least priority block in STT-RAM. If there is no block with 10 state in the SRAM line, we decrement our priority order by one. Similarly, we check blocks with 01 and 00 states. Priority with 00 is the case where we copy the SRAM contents to STT-RAM and STT-RAM contents to PCM. Whenever power comes back, STT-RAM contents are accessed automatically without copying to SRAM. Our migration policy automatically migrates from STT-RAM to SRAM if needed and vice-versa.

\[
\begin{align*}
00 & \quad \text{Migrate()} \quad \text{invoked} \\
01 & \\
10 & \\
11 & \quad \text{Migrate()} \quad \text{invoked} \\
\end{align*}
\]

\textbf{Fig. 4: State diagram to update CONF}

\section*{4.5 Detailed Example}

Figure 5 illustrates the detailed working of eBaRe architecture. In figure 5, Initially SRAM cache has (a,c) blocks and STT-RAM cache has (b,d) blocks. We defined all counters and CONF as a tuple \([\text{RIC, WC, CONF}]\) and initialized it to \([0, 0, 00]\). A prediction table has a PR field. We take a sequence of access requests; read requests are labelled as \(rd_i\) (i.e., read block i) and write requests
are labelled as \( wr_i \) (i.e., write block \( i \)). We labelled different timing points as A, B, C, .., K. In this section, we discuss how eBaRe architecture works after every timing point.

In Fig A of figure 5, we update the RIC of ‘a’ to 2 because of two consecutive reads. In Fig B, the WIC of ‘b’ has become 2. In Fig C, \([RIC, WIC]\) of ‘a’ updates to [3, 1]. In Fig D, the WIC of ‘b’ becomes 7, which equals the threshold and becomes a write-intensive block. Our placement policy suggests write-intensive blocks should place in SRAM. SRAM set is full; to replace the block, we find the block having the lowest WIC. Block ‘c’ has a low WIC value; we replace ‘c’ with ‘b’ and reset all ‘b’ counters to [0, 0, 00]. In Fig E, the RIC of ‘a’ becomes 7, which equals the threshold and becomes a read-intensive block. Our placement policy suggests read-intensive blocks should place in STT-RAM. STT-RAM set was having one empty slot; we migrate ‘a’ from SRAM to STT-RAM. Reset all ‘a’ counters to [0, 0, 00] and update the WIC of ‘b’ to 2.

In Fig F, RIC of ‘a’ updates to 4. In between F, G timing points, a new block request ‘c’ occurred. Block request ‘c’ is not present in both caches. Check the prediction table for index 2, associated with \( tag_c \) to find the c’s PR field. We found an entry in the prediction table of index 2 with PR = 1. If the PR value is 1, the block is placed in the SRAM cache during the last eviction. We place ‘c’ in the SRAM cache. In Fig G, the WIC of ‘c’ updates to 7, which equals the threshold and becomes a write-intensive block and update RIC of ‘a’ to 4. Our placement policy suggests that write-intensive blocks should be placed in the SRAM; ‘c’ is already in SRAM itself. We update the CONF of ‘c’ to 01 and resets the counter values. After H, WIC of ‘c’ updates to 3.

In between H, I timing points, a new block request ‘e’ occurred. Block request ‘e’ is not present in both caches. Check the prediction table for index 3, associated with \( tag_e \) to find the e’s PR field. We found an entry in the prediction table of index 3 with PR = 0. If the PR value is 0, the block is placed in the STT-RAM cache during the last eviction. We place ‘e’ in the STT-RAM cache. STT-RAM set is full; to replace the block, we find the lowest RIC. Block ‘d’ has a low RIC value; we replace ‘e’ with ‘d’. Update all ‘e’ counters to [1, 0, 00].

Power failure (PF) occurred; our backup policy saves important blocks using the CONF field. Where the CONF of ‘c’ has 01 and ‘a’ has 00, our priority order suggests that 01 has the highest priority than 00. We place ‘a’ to the main memory and backup ‘c’ to STT-RAM. We prefer write-intensive blocks compared to read-intensive during a power failure. So ‘b’ replaces ‘e’. In Fig J, ‘c’ and ‘b’ are saved to STT-RAM. Whenever power comes back (PB), we don’t require any restoration process. In Fig K, the RIC of ‘c’ and ‘b’ updates to 1.

4.6 Storage Overhead

We analyze the storage overhead because we added extra bits, a prediction table, and backup logic. We evaluate area overhead for the proposed eBaRe
architecture for the same system configuration shown in table 1. We showed the area overhead as an example. There are two aspects of the proposed eBaRe architecture that cause storage overhead.
• eBaRe architecture has two counters of 3-bit each and two confidence bits per block. Data-cache has 256 blocks; each block has 8 bits, so 256*8=2048 bits are required for data cache.
• The proposed prediction table has 4K byte entries, involving 1-bit per entry, so that total storage overhead will be 1024*4 = 4096 bits.

The overall storage overhead of eBaRe architecture will be 2048 + 4096 = 6144 bits=0.75KB. The total percentage of area overhead is about 0.75KB/32KB=2.34%.

5 Experimental Setup and Results

5.1 Experimental Setup

We evaluate the proposed eBaRe architecture using the gem5 [31] simulator and 18 benchmarks from the MiBench suite [36]. Overall micro-architectural parameters used for implementation are shown in table 1.

| Component          | Description                                                                 |
|--------------------|-----------------------------------------------------------------------------|
| CPU core           | 1-core x86, 2GHZ                                                            |
| L1 Cache           | Block size - 64-byte, 4-way associative (2-way SRAM, 2-way STT-RAM); Private cache (16KB hybrid D-cache, and 16KB I-cache), write-back, 2 cycles access time (write to STT-RAM: 20 cycles) |
| Size Parameters    | VB-1bit, WIC and RIC-3bits, CONF-2bits, L- 4K bytes, threshold-7, and PR-1bit |
| Main memory        | 128MB PCRAM                                                                 |

Table 2: Nvsim parameters of SRAM, MRAM Caches, PCRAM memory (350K, 22nm)

| Parameter          | 16KB SRAM       | 16KB MRAM      | 128MB PCRAM    |
|--------------------|-----------------|----------------|----------------|
| Read Latency       | 1.230 ns        | 1.956 ns       | 204.584 ns     |
| Read Energy        | 0.006 nJ        | 0.124 nJ       | 1.553 nJ       |
| Write Latency      | 1.210 ns        | 10.500 ns      | RESET - 104.954 ns SET - 214.954 ns |
| Write Energy       | 0.002 nJ        | 0.515 nJ       | RESET - 6.946 nJ SET - 6.927 nJ |
| Leakage Power      | 18.972 mW       | 3.014 mW       | -              |

The threshold which we used in our eBaRe architecture is set to 7 in all experiments. We experimented with threshold values 1, 3, 7, 15. We observe better energy consumption at threshold value 7 compared to other threshold values. This threshold value helps us to fix the counter bits.

Table 2 shows the dynamic energy and latency for a single read and write operation to SRAM and STT-RAM, taken using Nvsim [37].
5.2 Baseline Architecture

We modelled a baseline architecture to compare with the proposed eBaRe architecture. First, we compare the performance and dynamic energy consumption of pure SRAM, pure STT-RAM and hybrid (SRAM and STT-RAM) cache architectures.

- **Pure SRAM cache:** We don’t require any placement or migration policies in pure SRAM cache because we have only SRAM at L1.
- **Pure STT-RAM cache:** We don’t require any placement or migration policies in pure STT-RAM cache because we have only STT-RAM at L1.
- **Hybrid cache:** Hybrid cache architecture consists of both SRAM, STT-RAM at L1. We have taken the size of L1 as 32KB in all three architectures. In this HCA, we use a random placement policy. In the random placement policy, we place the block in either SRAM or STT-RAM randomly. We use a migration policy, which migrates blocks from one cache to another based on the counters. We empirically determined the threshold as 7 and the size of counters as 3-bit. Suppose the WIC crosses the threshold value and is present in the STT-RAM cache region. In that case, we migrate that particular block into the SRAM cache region. Suppose the RIC crosses the threshold value and is present in the SRAM cache region. In that case, we migrate that particular block into the STT-RAM cache region.

In above all three cache architectures, we were not using any prediction mechanisms. In this figure 6, the performance and energy consumption of the cache architectures are normalized based on the pure SRAM cache architecture. As shown in figure 6, hybrid-based architecture is performing in between pure SRAM, pure STT-RAM cache architectures, i.e., hybrid-based architecture, is better than pure STT-RAM cache. We selected this HCA as our baseline architecture throughout this paper with the above modelling details.

We experimented with the baseline architecture to fix the threshold value. Whenever the respective counter crosses its threshold, we migrated the block from one cache to another to check energy values. The size of the counter depends on a chosen threshold value. For instance, if the threshold value is 3, the counter size is log 4 (counts from 0 to 3). We experimented with the baseline architecture to fix the threshold value. Whenever the respective counter crosses its threshold, we migrated the block from one cache to another to check energy values. The size of the counter depends on a chosen threshold value. For instance, if the threshold value is 3, the counter size is log 4 (counts from 0 to 3). We fixed the threshold value as 7 because if we fixed the threshold value as 7 because if the threshold is ¡7, then the number of migrations between cache regions increases. If the threshold is ¿7, we observed more writes to NVM, which increases the energy consumption of HCA. We observed better energy consumption values for the threshold value 7 than other threshold values, as shown in figure 7.
Fig. 6: Comparisons between Pure SRAM, Pure STT-RAM, and Hybrid cache architectures.

5.3 Results

This section evaluates the proposed architecture under stable power and during intermittent power supply. We also evaluate proposed architecture efficiency w.r.t traditional checkpointing approach under stable power and frequent power failures. Lastly, we evaluate the proposed architecture for $\eta, \theta$ w.r.t baseline and existing architectures.

5.3.1 Under Stable Power:

We compare our proposed architecture with baseline and architecture proposed by Xie et al. [9]. We implemented Xie et al. [9] work to analyze both stable power and intermittent power systems. For a fair comparison, all these
architectures use the same system configuration shown in table 1, and energy/delay from table 2. One of the main objectives of the eBaRe architecture is to reduce the number of writes to the STT-RAM cache. To achieve this, we place the write-intensive blocks in the SRAM cache. We have shown the ratio of the write operations to STT-RAM with total write accesses in figure 8. A lower number of writes to STT-RAM shows the effectiveness of the proposed eBaRe architecture. The percentage of writes to the STT-RAM cache is normalized with the baseline architecture shown in figure 8. Overall, eBaRe architecture helps in reducing STT-RAM write operations from 63.35% to 35.93% compared to the baseline architecture.

Reducing the STT-RAM writes also guarantees better endurance and lifetime of IoT nodes. In figure 9, the performance and energy consumption values are normalized with the baseline architecture. Figure 9 (a) and figure 9 (b) shows that better execution time and dynamic energy consumption than the baseline and existing architectures. When we compare the proposed architecture with Xie et al. [9] architecture, we achieve better values because of accurate prediction. Xie et al. [9] work uses a pattern sampler for prediction, but we maintained PR bit for every block in our proposed architecture. PR bit helps us for block placement. If our prediction accuracy increases, the number of migrations decreases. If the number of migrations decreases, we observe a fewer number of writes to NVM. eBaRe architecture decreases the number of writes to STT-RAM.

Further, the proposed prediction table helps to decrease the number of migrations and accesses. Therefore, our architecture results in 32.85% better execution time and saves 23.42% of dynamic energy consumption than baseline architecture.
We did experiments for comparing the traditional checkpointing approach with the proposed architecture during stable power. We implemented a traditional checkpointing approach by creating a safe point for every 5-million instructions. At every 5-million instructions, we save the program state to main-memory. Our proposed architecture performs better than traditional checkpointing. In traditional checkpointing, backup happens for every safe point, but we only backup during a power failure in the proposed architecture. We normalized the performance and energy consumption values with the traditional checkpointing approach. Proposed HCA reduces performance overhead and energy consumption by 21.03% and 22.95%, as shown in figure 10.

5.3.2 Under Frequent Power failures:

We assume frequent power failures happen for every 2, 4 million instructions. We perform all experiments for one billion instructions in the gem5 simulator. We modelled three power failure cases. In case-1, power failures occur for every 2-million instructions. In case-2, power failures occur for every 4-million instructions. In case-3, power failures occur randomly in between 2 to 5-million instructions. In the figures 11, 12, 13, and 14, we refer to proposed 2M with case-1, proposed 4M with case-2, and proposed random with case-3. We calculated the average backup time ($B_t$), i.e. time required to backup all the SRAM contents to NVM. We also evaluate a random intermittent power system, where power failure occurs very often and randomly to check $B_t$ and the efficiency of the proposed eBaRe architecture. The performance and energy consumption values are normalized based on the baseline architecture.
Fig. 9: Comparisons between Proposed, Baseline, and Existing Architectures under stable power.

We compare the average $B_t$ w.r.t the baseline, as shown in figure 11. We also compared SRAM+PCM based architecture to show how much improvement in performance during intermittent power supply. In SRAM+PCM architecture, SRAM as L1 cache and PCM as main memory. We introduced a power failure randomly and a safe point for every 5-million instructions. When a power failure occurs, we back up all SRAM contents to PCM. Whenever power comes backs, we start the application’s execution from the nearest safe point. When we compared SRAM+PCM architecture with the proposed eBaRe architecture, eBaRe gives better because eBaRe saves data at the L1 cache itself (by using STT-RAM). eBaRe architecture saves the re-execution time of application and reduces the number of writes to PCM during a power failure. The performance and energy consumption values are normalized based on the baseline architecture. We compare the execution time and energy consumption.
with the baseline during these frequent power failures, as shown in figure 12 (a), (b).

We also compared eBaRe architecture with Xie et al. work. We achieved better execution time and energy values because they checkpoint only selective dirty blocks from SRAM to STT-RAM during power failures. This type of checkpointing increases writes to PCM, which increases dynamic energy consumption in Xie et al. work. Whenever power comes back, the proposed architecture uses blocks from STT-RAM directly. In Xie et al. work, STT-RAM has fewer blocks than the proposed architecture, increasing execution time for Xie et al. work.

As we discussed SRAM+PCM architecture, there is a safe point for every 5 million instructions. Whenever power failure occurs, we save the state in PCM. This type of backup policy increase writes to PCM. Whenever power comes back, the restore procedure increases the number of accesses from PCM to SRAM cache. In a hybrid cache, STT-RAM saves some blocks so that PCM observes fewer writes, and the restore takes lesser accesses from PCM. We evaluated both 32KB SRAM cache and hybrid cache (16KB SRAM+16KB STT-RAM) to check static power. We have seen the proposed architecture has a 17.02% improvement in static power compared to 32KB SRAM+PCM architecture.

We compare the traditional checkpointing approach with the proposed architecture during power failures. As earlier said, we implemented a traditional checkpointing approach by creating a safe point for every 5-million instructions. We save the program state for every 5-million instructions. We
retrieve the program state from the main memory at every safe point to continue with the remaining execution of the application. For instance, if a random power failure occurs at 9th million instruction. We re-execute the application from 5th million instruction because the nearest safe point is at 5th million instruction. The performance and energy consumption values are normalized based on the traditional checkpointing approach. We compared the proposed architecture with the traditional checkpointing approach, which reduces performance overhead and energy consumption by 36.10% and 31.03%, as shown in figure 13.

Lastly, as shown in figure 14, we calculate the backup efficiency ($\eta$) and energy efficiency ($\theta$) using equations 3, 4 for both proposed and existing architectures. The $\eta$ and $\theta$ values are normalized based on the baseline architecture. Our proposed architecture improves $\eta$ by 32.52% and $\theta$ by 43.41% because of the proposed backup strategy compared with the baseline. The other reason for improvement in both $\eta$ and $\theta$ is reduction in both $E_{backup}$ and $B_t$.

6 Conclusion

eBaRe architecture is a promising HCA for embedded systems in IoT. eBaRe architecture is mainly helpful for IoT applications, where power failures are unpredictable and very often. In hybrid caches, NVM introduces overhead because of its high write latency and energy. We proposed an efficient prediction based placement policy and an intelligent migration policy, which uses SRAM and STT-RAM efficiently. Using the proposed prediction table effectively, we reduce the number of writes to STT-RAM. eBaRe architecture
decreases STT-RAM writes from 63.35% to 35.93% w.r.t baseline architecture. As a result, we achieve better energy consumption and execution time.

We evaluated eBaRe architecture against state of the art, and baseline architectures. eBaRe achieves better energy and backup efficiency. To guarantee the efficient backup of the program state, we proposed a backup strategy. The proposed backup strategy helps to recognise important blocks and migrate them to the STT-RAM cache during a power failure. eBaRe takes less backup time when compared with baseline and state of the art architectures. Whenever power comes back, we use STT-RAM contents directly without any restore procedure.

Fig. 12: Comparisons between Proposed, Baseline, and Existing Architectures under frequent power failures.
Fig. 13: Comparison in-terms of Performance Overhead and Energy Consumption during power failure

References

[1] Golpîra, H., Khan, S.A.R., Safaeipour, S.: A review of logistics internet-of-things: Current trends and scope for future research. Journal of Industrial Information Integration, 100194 (2021)

[2] Hu, X., Xu, L., Lin, X., Pecht, M.: Battery lifetime prognostics. Joule 4(2), 310–346 (2020)

[3] Liu, Y., Li, H., Li, X., Xue, J.C., Xie, Y., Yang, H.: Self-powered wearable sensor node: Challenges and opportunities. In: International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp. 189–189 (2015). IEEE

[4] Martinez, B., Monton, M., Vilajosana, I., Prades, J.D.: The power of models: Modeling power consumption for iot devices. IEEE Sensors Journal 15(10), 5777–5789 (2015)

[5] Jog, A., Mishra, A.K., Xu, C., Xie, Y., Narayanan, V., Iyer, R., Das, C.R.: Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps. In: Design Automation Conference, pp. 243–252 (2012). IEEE

[6] Pan, C., Xie, M., Hu, J., Chen, Y., Yang, C.: 3m-pcm: Exploiting multiple write modes mlc phase change main memory in embedded systems. In: Proceedings of the International Conference on Hardware/Software
Fig. 14: (a) Comparison of Backup Efficiency ($\eta$) and (b) Energy Efficiency ($\theta$) during Power failures.

[7] Instruments, T.: MSP430FR5969 launchpad development kit (2018)

[8] Wang, Z., Jiménez, D.A., Xu, C., Sun, G., Xie, Y.: Adaptive placement and migration policy for an stt-ram-based hybrid cache. In: IEEE 20th International Symposium on High Performance Computer Architecture, pp. 13–24. IEEE, ??? (2014). IEEE

[9] Xie, M., Pan, C., Zhang, Y., Hu, J., Liu, Y., Xue, C.J.: A novel stt-ram-based hybrid cache for intermittently powered processors in iot devices. IEEE Micro 39(1), 24–32 (2018)

[10] Ma, K., Zheng, Y., Li, S., Swaminathan, K., Li, X., Liu, Y., Sampson,
J., Xie, Y., Narayanan, V.: Architecture exploration for ambient energy
harvesting nonvolatile processors. In: IEEE 21st International Symposium
on High Performance Computer Architecture, pp. 526–537. IEEE, ???
(2015). IEEE

[11] Li, H., Liu, Y., Zhao, Q., Gu, Y., Sheng, X., Sun, G., Zhang, C., Chang,
M.-F., Luo, R., Yang, H.: An energy efficient backup scheme with low
inrush current for nonvolatile sram in energy harvesting sensor nodes. In:
Design, Automation & Test in Europe Conference & Exhibition, pp. 7–12.
IEEE, ???. (2015). IEEE

[12] Mao, M., Li, H., Jones, A.K., Chen, Y.: Coordinating prefetching and
stt-ram based last-level cache management for multicore systems. In:
Proceedings of the 23rd ACM International Conference on Great Lakes
Symposium on VLSI, pp. 55–60. ACM New York, NY, USA, ???. (2013)

[13] Sun, Z., Bi, X., Li, H., Wong, W.-F., Ong, Z.-L., Zhu, X., Wu, W.: Multi
retention level stt-ram cache designs with a dynamic refresh scheme. In:
Proceedings of the 44th Annual IEEE/ACM International Symposium on
Microarchitecture, pp. 329–338. ACM New York, NY, USA, ???. (2011)

[14] Smullen, C.W., Mohan, V., Nigam, A., Gurumurthi, S., Stan, M.R.:
Relaxing non-volatility for fast and energy-efficient stt-ram caches. In:
IEEE 17th International Symposium on High Performance Computer
Architecture, pp. 50–61. IEEE, ???. (2011). IEEE

[15] Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., Xie, Y.: Design
exploration of hybrid caches with disparate memory technologies. ACM
Transactions on Architecture and Code Optimization 7(3), 1–34 (2010)

[16] Kim, N., Ahn, J., Choi, K., Sanchez, D., Yoo, D., Ryu, S.: Benzene: An
energy-efficient distributed hybrid cache architecture for manycore sys-
tems. ACM Transactions on Architecture and Code Optimization 15(1),
1–23 (2018)

[17] Zhao, J., Xu, C., Zhang, T., Xie, Y.: Bach: A bandwidth-aware hybrid
cache hierarchy design with nonvolatile memories. Journal of Computer
Science and Technology 31(1), 20–35 (2016)

[18] Ahn, J., Yoo, S., Choi, K.: Prediction hybrid cache: An energy-efficient stt-
ram cache architecture. IEEE Transactions on Computers 65(3), 940–951
(2015)

[19] Gao, L., Wang, R., Xu, Y., Yang, H., Luan, Z., Qian, D., Zhang, H., Cai,
J.: Sram-and stt-ram-based hybrid, shared last-level cache for on-chip
cpu–gpu heterogeneous architectures. The Journal of Supercomputing
74(7), 3388–3414 (2018)
[20] Yao, J., Ma, J., Chen, T., Hu, T.: An energy-efficient scheme for stt-ram l1 cache. In: IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, pp. 1345–1350. IEEE, ??? (2013). IEEE

[21] Xie, M., Zhao, M., Pan, C., Hu, J., Liu, Y., Xue, C.J.: Fixing the broken time machine: Consistency-aware checkpointing for energy harvesting powered non-volatile processor. In: Proceedings of the 52nd Annual Design Automation Conference, pp. 1–6. ACM New York, NY, USA, ??? (2015)

[22] Liu, Y., Suy, F., Wangy, Z., Yang, H.: Design exploration of inrush current aware controller for nonvolatile processor. In: 2015 IEEE Non-Volatile Memory System and Applications Symposium, pp. 1–6. IEEE, ??? (2015). IEEE

[23] Zhou, Y., Zhao, M., Ju, L., Xue, C.J., Li, X., Jia, Z.: Energy-aware morphable cache management for self-powered non-volatile processors. In: IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 1–7. IEEE, ??? (2017). IEEE

[24] Xie, M., Zhao, M., Pan, C., Li, H., Liu, Y., Zhang, Y., Xue, C.J., Hu, J.: Checkpoint aware hybrid cache architecture for nv processor in energy harvesting powered systems. In: International Conference on Hardware/Software Codesign and System Synthesis, pp. 1–10. IEEE, ??? (2016). IEEE

[25] Ransford, B., Sorber, J., Fu, K.: Mementos: System support for long-running computation on rfid-scale devices. In: Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 159–170 (2011)

[26] Balsamo, D., Weddell, A.S., Das, A., Arreola, A.R., Brunelli, D., Al-Hashimi, B.M., Merrett, G.V., Benini, L.: Hibernus++: a self-calibrating and adaptive system for transientsly-powered embedded devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35(12), 1968–1980 (2016)

[27] Kortbeek, V., Yildirim, K.S., Bakar, A., Sorber, J., Hester, J., Pawelczak, P.: Time-sensitive intermittent computing meets legacy software. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 85–99 (2020)

[28] Su, F., Liu, Y., Wang, Y., Yang, H.: A ferroelectric nonvolatile processor with 46 \(\mu s\) system-level wake-up time and 14\(\mu s\) sleep time for energy
harvesting applications. IEEE Transactions on Circuits and Systems I: Regular Papers 64(3), 596–607 (2016)

[29] Choi, J., Joe, H., Kim, Y., Jung, C.: Achieving stagnation-free intermittent computation with boundary-free adaptive execution. In: IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 331–344 (2019). IEEE

[30] Jog, A., Mishra, A.K., Xu, C., Xie, Y., Narayanan, V., Iyer, R., Das, C.R.: Cache revive: Architecting volatile sst-ram caches for enhanced performance in cmps. In: Design Automation Conference, pp. 243–252 (2012). IEEE

[31] Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., et al.: The gem5 simulator. ACM SIGARCH computer architecture news 39(2), 1–7 (2011)

[32] Sun, G., Dong, X., Xie, Y., Li, J., Chen, Y.: A novel architecture of the 3d stacked mram l2 cache for cmps. In: IEEE 15th International Symposium on High Performance Computer Architecture, pp. 239–249 (2009). IEEE

[33] Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., Xie, Y.: Hybrid cache architecture with disparate memory technologies. ACM SIGARCH computer architecture news 37(3), 34–45 (2009)

[34] Li, J., Xue, C.J., Xu, Y.: Sst-ram based energy-efficiency hybrid cache for cmps. In: IEEE/IFIP 19th International Conference on VLSI and System-on-Chip, pp. 31–36 (2011). IEEE

[35] Jadidi, A., Arjomand, M., Sarbazi-Azad, H.: High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement. In: IEEE/ACM International Symposium on Low Power Electronics and Design, pp. 79–84 (2011). IEEE

[36] Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: Mibench: A free, commercially representative embedded benchmark suite. In: Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization, pp. 3–14 (2001). IEEE

[37] Dong, X., Xu, C., Xie, Y., Jouppi, N.P.: Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 31(7), 994–1007 (2012)