Hybrid Cache Design Under Varying Power Supply Stability – A Comparative Study

Nils Wilbert, Stefan Wildermann, Jürgen Teich
{nils.wilbert,stefan.wildermann,juergen.teich}@fau.de
FAU Erlangen-Nürnberg
Erlangen, Germany

ABSTRACT

As emerging Non-Volatile Memory (NVM) technologies are ever maturing, the usage of such novel memory technologies in embedded systems is becoming increasingly viable. Especially when introduced at the cache level, these technologies can promise great advantages over conventional SRAM technology as they can serve read requests at lower energy and, in applications of intermittent computing, reduce the backup overhead during a power outage. Nevertheless, write accesses to NVMs consume more energy compared to writing to a conventional SRAM. As a remedy, hybrid caches, introducing both NVM and SRAM to the cache hierarchy, can realize trade-offs by minimizing the expected NVM write overhead while still exploiting the NVM read efficiency and non-volatility property. However, these hybrid designs open up a number of additional decisions in the design space, which are examined in this paper. Most notably, the choice of the degree of non-volatility introduced to the cache hierarchy in addition to the choice of a suitable cache replacement policy has not been analyzed and solved systematically yet. By employing simulation-based experiments, we evaluate embedded applications featuring various memory access patterns under both continuous and intermittent power supplies for different types of hybrid cache designs. Our results show how the combination of application and architecture characteristics, in addition to the expected power supply stability, leads to a vast range of hard-to-predict cache effects affecting both performance and energy efficiency. Both the potential and a necessity for automated design tools, including the exploration of hybrid memory hierarchies, therefore arise.

CCS CONCEPTS

• **Computer systems organization** → *Embedded hardware*; • **Hardware** → **Emerging architectures**.

KEYWORDS

NVM, Embedded Systems, Hybrid Cache, Memory Hierarchy, Intermittent Computing, System Design

1 INTRODUCTION

As battery-less Internet of Things (IoT) devices relying on energy harvesting are increasingly finding their way into our everyday lives in the form of wearables, smart buildings, healthcare, etc., the memory hierarchy design of such systems is also becoming increasingly relevant. As for devices, e.g., powered by solar panels, power outages can be expected frequently, the field of intermittent computing [15, 28] needs to be considered for such systems. More precisely, during a power outage, all contents of a volatile memory are lost, unless moved to a non-volatile memory in a triggered backup process powered by, e.g., additional capacitors [18]. With NVM technologies such as Phase-Change RAM (PCRAM) or Spin-Transfer Torque RAM (STT-RAM) having matured enough to become commercially viable [1, 17], they can provide a solution for intermittently powered devices in order to reduce the worst-case backup overhead. STT-RAM in particular has already been identified as a suitble technology for the implementation of caches [27, 29]. However, the introduction of NVM is expected to generate additional overhead compared to conventional SRAM in terms of both write latencies and required write energy consumption. A completely non-volatile cache hierarchy or even Non-Volatile Processors (NVPs) [16, 26] are thus not always feasible. Therefore, hybrid caches combining both volatile and non-volatile technologies, thus realizing a trade-off, have already been identified as a field of research [2, 12, 24, 29]. However, the hybridization of caches leads to an increase in the size of the design space, including but not limited to the degree to which non-volatility should be introduced. Additionally, the choice of a suitable cache replacement policy is becoming more complex as, for hybrid caches, a need for adapted replacement policies arises. While replacement policies are a key factor in overall cache performance, conventional policies such as LRU do not consider the different characteristics of the underlying memory technologies in their decisions. More precisely, due to the aforementioned NVM write overhead, architecture-aware replacement policies should generally avoid placing write-intensive data in the non-volatile section of a hybrid cache. As furthermore only a few existing replacement policies exist to support intermittent computing, it is of great interest to investigate and analyze how the degree of non-volatility and the choice of cache replacement policy influence performance depending on the application running on the device and the expected stability of the power supply. To the best of our knowledge, this paper is the first to treat all the above factors together in a comparative analysis to raise awareness for trade-off cache designs and respective replacement policies. We thus underline the importance of automated design tools for domain and/or even application-specific hybrid cache hierarchies.

The paper is structured as follows: First, in Section 2 we present related work to our topic and outline our key contributions. In Section 3, we present a number of concepts and design decisions regarding the design of hybrid caches. Section 4 contains a more thorough explanation of the architecture-aware cache replacement policies investigated in this paper. Finally, after presenting our experimental setup in Section 5, we discuss the results under stable and intermittent power supplies in Sections 6 and 7, respectively. Concluding notes are provided in Section 8.

2 RELATED WORK AND CONTRIBUTIONS

One of the earliest works investigating the potential of hybrid caches is provided by [29]. Here, a cache replacement policy for caches featuring a volatile and non-volatile section in each cache set, with access-based placement decisions and a migration mechanism in case the currently observed access pattern is more efficiently served in the opposing cache section, is presented. More sophisticated replacement policies featuring, e.g., prediction mechanisms in order to avoid placing write-intensive data to a non-volatile cache section are introduced in a number of later works, such as [2, 14, 21]. Most of these approaches focus on exploiting NVM's high density in order to realize large last-level caches.

The topic of designing hybrid memory hierarchies for the purpose of intermittent computing, thus exploiting the non-volatility property itself, is also an active field of research. While approaches such as [4, 20, 23] already focus on using NVM in order to realize efficient checkpointing, i.e., saving the current system state to NVM at specific points in time, they do not consider the hybridization taking place within explicit levels of the memory hierarchy themselves. More recent works, consider the potential of hybrid caches in the field of intermittent computing more closely by providing their own cache policies designed to deal with power outages [3, 30].

Regarding Design Space Exploration (DSE) tools for hybrid memory hierarchies, in [22], a machine-learning-based method is proposed. However, this approach solely focuses on the design of hybrid main memories in addition to not considering the field of intermittent computing. While intermittent computing is introduced for MemCork [25], caches are again not included in the analysis. Whereas HyCSim [9] promises a lightweight simulation framework for the purpose of exploring hybrid last-level caches, it cannot yet provide an automated DSE on its own, in addition to not including the possibility of power outages in its approach. Therefore, to our knowledge, there are currently no automated DSE tools considering hybrid memories on all levels of the hierarchy, including caches and their various policies, while also offering support for intermittent computing.

Taking a first step towards such design tools, our main contributions of this paper are listed in the following:

- Creation of a modular, flexible simulation framework for hybrid cache designs, including intermittent power supplies.
- Comparative study of hybrid cache designs carrying different degrees of non-volatility and different cache replacement policies under both stable and unstable power supplies.
- Analyzing the interaction of application and architecture characteristics in hybrid cache designs.
- Demonstrating the need and potential for automated design tools for hybrid memory hierarchies in the field of intermittent computing.

3 DESIGN DECISIONS FOR HYBRID CACHES

In this section, we outline a number of design decisions specific to the concept of hybrid memory hierarchies. For the rest of this paper, we assume write-back caches, as write-through caches generate a significant amount of additional writes unsuitable for NVM.

3.1 Degree of Non-Volatility

One of the most crucial decision in the design space of hybrid memory hierarchies lies in the degree to which non-volatility should be introduced at each level of the memory hierarchy. While designing efficient hybridized **main memories** is possible [7], it proves unsuitable for intermittent computing as volatile main memory contents would have to be saved to an additional non-volatile backup storage, e.g., an additional flash memory. Especially for smaller embedded devices, such as Microcontroller Units (MCUs) this additional overhead is often not acceptable. A completely non-volatile main memory is thus assumed for the rest of this paper.

Regarding the cache hierarchy, for the hybridization at a **single cache level**, we split each cache set into volatile and non-volatile ways, with the concrete ratio being subject to configuration. The mapping between data and cache sets is determined statically and does not change over the time of execution. Furthermore, data mapped to the same cache set can vary in their write intensity. Additionally, access patterns to the same data are subject to change over the time of execution. In a hybrid cache set, NVM writes can therefore be avoided while still being able to exploit NVM reads and the non-volatility property. This leads to cache policies having to make a decision regarding the cache section on placement, as well as being offered the possibility to support the migration of data between cache sections. Note that in our case, the ratio of non-volatile cache lines in a cache set is the same for all sets at the observed cache level. A more complex approach could be implemented by applying different degrees of non-volatility to each cache set individually. This can be beneficial for applications where certain address regions are considered to be highly write-intensive, while others, mainly mapped to different cache sets, are rather read-intensive. However, this requires deep insight and analysis into application access patterns and creates a more complex hardware design since, for instance, connections between physical cache lines and comparators cannot be drawn symmetrically for each cache set. This approach of an uneven distribution of non-volatile cache lines is therefore not considered further in this paper.

After a power outage, we want to maintain the validity of a non-volatile cache line, as well as ensure that a volatile cache line is not erroneously treated as valid. The **tag array** with the corresponding **cache flags** (i.e., valid bit, dirty bit) thus has to be hybridized in the same fashion as the data array, meaning the volatility of the corresponding tag and data array entries should always match. An examplary overview of the examined memory hierarchies can be seen in Figure 1.

With regard to a **multi-level cache hierarchy**, the size of the design space further increases, as the degree of non-volatility can be set individually for each cache in the hierarchy. However, employing multiple hybrid caches comes with additional challenges for intermittent computing concerning *cache inclusivity*. To provide an example, consider the data A being stored in a non-volatile cache line in the L1 cache whilst being stored in a volatile L2 cache line. Upon a power failure, a writeback for A in the non-volatile L1 cache does not have to be initiated, whereas in the volatile L2 cache, A is written back to the next lower hierarchy in case it is modified as it will get lost from the L2 cache afterwards. While cache coherence protocols guarantee that the L2 cache did not

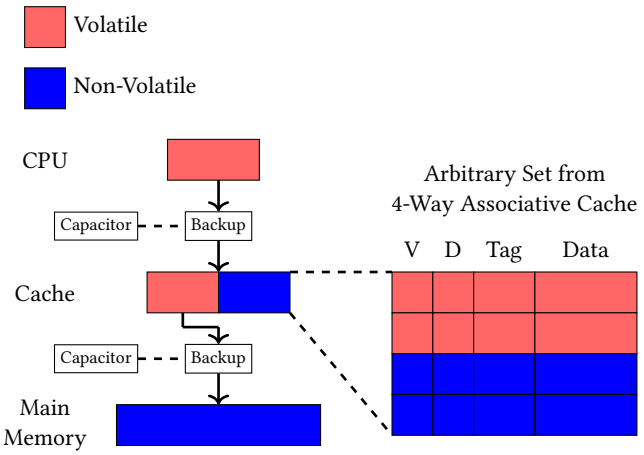

Figure 1: Example of a hybrid memory hierarchy, featuring a non-volatile main memory and a hybrid cache with an exemplary cache set shown on the right explaining the hybridization at cache set level. Valid and dirty bits are denoted as "V" and "D", with volatile and non-volatile components shown in red and blue, respectively. The direction of the backup process necessary upon power outages with the required backup capacitors is further displayed.

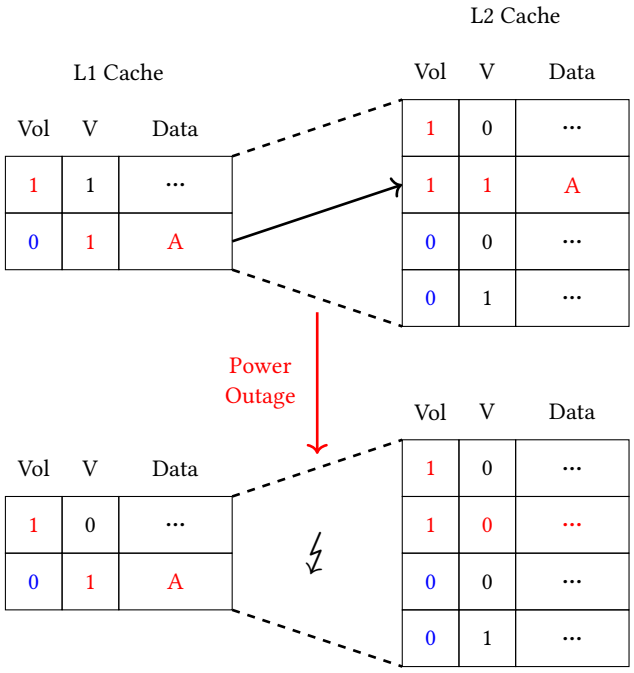

Figure 2: Example how for hybrid memory hierarchies, the inclusion policy for data "A" can be violated in case of a power outage. The "Vol" field indicates whether a cache line is volatile (1) or not (0). The valid bit is denoted as "V".

provide a more recent version of *A* than the L1 cache, the cache inclusivity might be violated as for inclusive caches, data can only exist in an upper-level memory in case a copy of it is contained in all lower-level memories. Therefore, for our example, the cache inclusion policy has to be triggered during power outages even for *A* being unmodified, thus initiating a writeback and evicting *A* from the L1 cache, instead of simply dropping *A* from the L2 cache. This behavior is further visualised in Figure 2. This greatly diminishes the benefits of NVM, since for inclusive caches, even data stored in a non-volatile section are no longer guaranteed to withstand a power outage without generating additional backup overhead. However, even for *non-inclusive caches*, a writeback can still be required for a non-volatile cache line during power outages. Consider the modified data *B* in a volatile L1 cache whose tag is not contained in a fully non-volatile L2 cache. When backing up *B* from the L1 cache, a victim *C* is thus chosen in the L2 cache. In case *C* is also modified, this leads to an additional writeback, as the evicted data *C* from the L2 cache has to be written back to the next lower level of the memory hierarchy. This cannot occur in inclusive caches, as there will always be a cache line in the L2 cache containing an earlier version of *B*, thus not requiring an additional writeback when backing up the L1 cache. Therefore, even the cache inclusion policy has to be considered when designing a hybrid memory hierarchy. For this paper, the role of the inclusion policy can be neglected, as we restrict our experiments to designs featuring a single cache level.

3.2 Power Outage Behavior

The approach chosen in this paper for saving the system state is based on approaches such as Hibernus [4]. Here, instead of relying on checkpoints set within the application, the system dynamically reacts to power failures by detecting that the supply voltage drops below a threshold and then saving the current execution state and all volatile data to a non-volatile memory. After regaining power, the execution can thus continue where it was halted. More precisely, in our case, the backup steps are performed sequentially along the memory hierarchy starting at the CPU side. First, the CPU is blocked for any new intruction fetches, while executing all currently issued instructions to completion. Even a pipelined out-of-order CPU is thus stopped leaving behind a consistent state. Afterwards, the CPU registers containing, e.g., the Program Counter (PC) have to be saved. For our case, a backup to non-volatile shadow registers can be assumed, allowing the register saving to be parallelized with the backup of modified volatile cache lines. Starting at the cache closest to the CPU, we first have to wait until all outstanding requests are completed. Despite an already drained CPU pipeline, this is still relevant, as, for instance, a request to write back the result of an instruction otherwise completed and thus no longer in the pipeline can still have caused a cache miss yet to be handled. Afterwards, the necessary writebacks can be issued. Evidently, a writeback has to be performed for all modified data stored in a volatile cache line. Once all writebacks at this level of the cache hierarchy have been performed, we can eventually move on to the next cache level. Since, as established in Section 3.1, in our case,

the main memory is assumed to be non-volatile, it does not require any special handling. After harvesting enough energy to resume the execution, i.e., having harvested enough energy in order to deal with a potential successive power outage, the program can simply continue at the next instruction as indicated by the backed-up PC.

In battery-less devices, this process can be powered by a capacitor whose remaining energy is large enough to adhere to the worst case, where all cache lines are modified [15]. The lower the required energy to save the system state in the worst case, the smaller the device's capacitor can therefore be scaled while achieving the same amount of factual progress. Furthermore, the remaining energy the system has to provide to deal with a power outage in the worst case is thus determined by the architecture itself rather than the cache replacement policy or the expected power outage frequency.

3.3 Volatility of Replacement Policy Metadata

An open question when designing hybrid caches fit for intermittent computing lies in the behavior of the cache replacement policies during power outages. While there may be a number of possibilities regarding the implementation, almost all known replacement policies require some type of metadata for their victim selection, e.g., counters, time stamps, queues, etc. For a volatile cache, the technology chosen for the storage of this metainformation is not of concern, as all cache lines are invalid after a power outage. However, for a hybrid cache where non-volatile cache lines remain valid after a power outage, this decision is becoming part of the design space. For the first example, consider an idealized LRU strategy implemented by adding a time stamp to each cache line, indicating when a cache line was last accessed. On the one hand, as this time stamp is written on every access to the corresponding cache line, storing the metadata in NVM can lead to significant write overhead. On the other hand, keeping all LRU metainformation in a volatile state can result in poor replacement decisions after a power outage. Depending on the concrete implementation, valid non-volatile cache lines might even be selected as victims, despite invalid volatile cache lines being available. Inversely, as replacement policies are merely heuristics, losing their metadata can arguably also be seen as a chance to counter mispredictions regarding suitable replacement candidates. For LRU, a hybridization of the metadata in line with the tag and data arrays can be seen as a potential solution, as volatile cache lines become invalid after a power outage and thus do not benefit from non-volatile metadata. However, the effects of different technological implementations still have to be evaluated, depending on the access patterns employed by the application and the expected frequency of power outages.

Nonetheless, the approach of hybridizing replacement policy metadata cannot be applied to all types of common replacement policies. Consider a cache employing a round-robin policy, a replacement policy supported by most ARM processors. Here, a single counter per cache set, cycling through the way indices, is pointing at the next cache line to be replaced. A hybridization is thus not possible. Moreover, independent of the chosen underlying technology, this policy may perform poorly after a power outage, as valid non-volatile cache lines can be indexed as the victim for the next replacement rounds before eventually cycling back to invalid volatile cache lines. Even these seemingly minor details are therefore becoming relevant when designing hybrid caches for the field of intermittent computing.

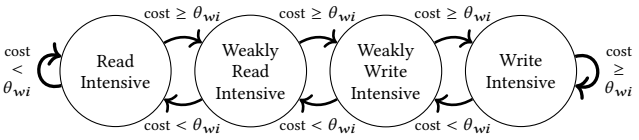

Figure 3: State machine showing transitions between four different write intensity states.

4 REPLACEMENT POLICIES

Cache replacement policies are crucial to managing the limited space available in cache memory. LRU is one of the most prominent policies. However, when dealing with hybrid caches, policies not only have to consider how frequently or recently data in caches was used, but they also have to anticipate the access patterns to the cache lines. In this section, we present two differently characterized cache replacement policies from literature designed for hybrid caches, which will be used in our evaluations.

4.1 Write Intensity Policy

In the following, we outline a state-of-the-art hybrid cache replacement policy originally presented in [2]. We will refer to this policy as the *WI policy*.

The goal of the WI policy is to minimize the number of writes to non-volatile cache lines as they have a relatively high dynamic energy consumption without unnecessarily increasing the miss rate to a greater extent. For this purpose, upon a cache miss, the current state of a state machine with four different write intensity states is used to predict the write intensity of succeeding hitting accesses to this cache line. Such a state machine is displayed in Figure 3. For identifying a data access so that it can be unambiguously associated with a state machine, the PC of the instruction triggering the data access is used. Without a priori knowledge of the application, cache misses can occur at every possible PC. Note that providing a state machine to track the write intensity for a cache miss at every possible PC is infeasible to realize in hardware. Therefore, only 256 state machines are used here, with the PC being hashed to index a prediction table containing the current state of the corresponding state machine, naturally leading to collisions. If, upon placement, the write intensity state machine is currently in one of the two more write-intensive states, the data is placed in the volatile section. If it is in one of the two more read-intensive states, it is placed in the non-volatile section. Finding a replacement candidate in the chosen section can be performed using LRU as all remaining cache lines share the same technology.

For this policy, a cost field is added to each cache line. This cost is used when updating the state. The cost field is initially set to zero. Upon a *read hit*, the cost field is decremented by one, in case of a *write hit*, it is incremented by 24 according to [2]. When a cache line eventually gets *evicted*, the cost field is then compared against a predetermined threshold θ_{wi}. If the costs are higher or equal

to θ_{wi}, the state machine that was used to initially determine the placement of the now to be evicted cache line transitions to the next more write-intensive state. Vice versa, if the costs are lower than θ_{wi}, the state machine transitions to the next more read-intensive state.

The threshold thus has to be carefully balanced in order to perform beneficial placement decisions. As applications can greatly differ in their access patterns, a suitable setting would have to be chosen individually for each application. While Ahn et al. [2] therefore propose a dynamic threshold adjustment during runtime, this comes at the cost of additional hardware overhead. Since for small IoT devices, this overhead is to be avoided with embedded systems further designed to only run specific types of applications, for this paper, we stick to a statically determined threshold as it proves feasible for our cause.

4.2 Confidence-based Migration Policy

The second architecture-aware replacement policy we will analyze and compare is based on the works of Badri et al. [3] with some minor modifications of our own. Unlike the WI policy, this policy is designed for the purpose of intermittent computing. We will refer to this policy as the *CM policy*.

Here, similar to the WI policy, a global table indexed with the hashed PC is used to decide on the cache section upon placement. However, for the CM policy, the table entries solely contain a single bit indicating the cache section the requested data was located in on its latest eviction. Instead of state machines aiming at predicting the write intensity for future accesses, here, proactive migration between cache sections is performed to react to changing access patterns. More precisely, two counters, namely the read intensity counter ric_i and write intensity counter wic_i, in addition to a confidence field $conf_i$, all initially set to zero, are introduced for each cache line i. Upon a *read hit* to cache line i, ric_i is incremented by one, whereas wic_i is incremented by one in the case of a *write hit*. Moreover, a threshold θ_{cm} is introduced. Read accesses are considered suitable for the non-volatile section, whereas write accesses are more efficiently performed in the volatile section. Therefore, once ric_i hits the threshold θ_{cm}, with i being a non-volatile cache line, as well as for wic_j hitting θ_{cm}, with cache line j being volatile, we can be confident that the data is located in the most suitable cache section. In these cases, $conf_i$ and $conf_j$ are thus incremented, with the corresponding counters being reset to zero. The value of the confidence field is further capped at a value of three.

Inversely, θ_{cm} can also be hit by wic_k, with cache line k being non-volatile, or ric_l, with cache line l being volatile. In these cases, the section the data is currently placed in is likely not the most suitable choice. The migration mechanism will thus swap the contents of the cache line hitting the threshold, with a cache line from the opposing section in the same cache set. To be more precise, when looking for a swap candidate in the volatile section, a cache line i fulfilling Equation (1) is selected. Vice versa, a swap candidate in the non-volatile section is identified by choosing a cache line i fulfilling Equation (2). Additionally, Equation (1) and Equation (2) are also used to select a victim in the volatile and non-volatile section upon initial placement, if necessary.

$$\text{argmin}_i\{wic_i + (conf_i \cdot \theta_{cm})\} \tag{1}$$

$$\text{argmin}_i\{ric_i + (conf_i \cdot \theta_{cm})\} \tag{2}$$

Equation (1) and Equation (2) are supposed to reflect the least write- and read-intensive cache lines in the volatile and non-volatile cache section, respectively. However, in the initial proposal [3], solely the wic_i and ric_i counters are used to identify suitable target cache lines. This can only poorly display the actual access frequency, since the counters wic_i and ric_i get reset once the threshold θ_{cm} is hit. On the contrary, the addition of the number of times the threshold has been hit, i.e., $conf_i$, times the number of accesses required to hit the threshold, i.e., θ_{cm}, does reflect the actual number of accesses until the confidence cap is reached, and is therefore employed by us. Note how the counter used in the sum matches the type of access leading to an increased confidence, as Equation (1) and Equation (2) are only used within the volatile and non-volatile cache sections, respectively. Furthermore, instead of swapping the section of two cache lines once θ_{cm} is hit by a counter indicating accesses unsuitable for the current cache section, the initial approach [3] simply evicts the migration target from the opposing section. As in this case, unlike during initial placement, we do not require any additional space in the cache, with misses generally to be avoided, we determined this deviation from the original approach to be suitable. In order to avoid accessing any invalid intermediate states, we further model the overhead of a swap operation by blocking accesses to the concerning cache set until the swap is completed.

In case of a power outage, the CM policy additionally features a backup mechanism utilizing the confidence field. With high confidence values implying a high number of accesses, when encountering a power failure, the confidence values between the volatile and non-volatile cache lines are compared, in order to determine which data should be saved at this level of the cache hierarchy. For each cache set, the CM policy compares whether the highest confident volatile cache line has a higher confidence value than the least confident non-volatile cache line and if so, swaps the location of the corresponding contents. This process is continued until no additional suitable pairs for migration can be identified. By keeping frequently accessed data as close to the CPU as possible, CM thus attempts to minimize miss rates and latencies, as the data is expected to be accessed again soon after regaining power.

5 EXPERIMENTAL SETUP

In this section, we display the applications and architecture settings used to perform our comparative study. Additionally, we outline the way our simulation behaves in the event of a triggered power outage.

Regarding the simulation, the cycle-accurate system simulator gem5 [5] is used, coupled with NVMain 2.0 [19] in order to simulate non-volatile main memories. We have extended the gem5 simulation in order to allow for the configuration of caches featuring a variable degree of non-volatile cache lines per set. Furthermore, we have added support for different read and write latencies and energy consumptions depending on the cache section, along with various statistics specific to hybrid caches.

5.1 Power Outage Simulation

Concerning the simulation of power outages, we employ a *power outage controller*, both triggering the power outage and the corresponding necessary backup process. Here, an interface featuring a power outage start cycle, a power outage periodicity and a maximum number of simulated power outages is provided. More precisely, users can specify the simulation cycle at which the first power outage is to be triggered. Additionally, users provide a number of cycles for which the execution will continue after completing the backup process, i.e., having dealt with the encountered power outage, until the next power outage is triggered. This interface goes in line with the expected power outage pattern in intermittently powered devices. For instance, in, e.g., solar-powered devices, the execution is performed continuously until a certain point at which the sun starts to set. Energy can now only be harvested sparingly by exploiting sunlight reflected by the moon. Successive power outages are thereby expected in short frequency after regaining enough power to continue the execution until the eventual sunrise.

5.2 Architectures

Our architectures used for our evaluation are based around embedded systems, such as devices based on the ARM Cortex-M85 which employs caches as well as a pipelined CPU. More precisely, using the ARM ISA, we will make use of gem5's generalized `O3CPU` CPU model, featuring a 5-stage pipeline with instructions issued in-order and potentially executed out-of-order. The simulated single-core CPU will further be clocked at 480 MHz, with the system clock set to 240 MHz. Regarding the caches, in line with the ARM Cortex-M85, we employ a single-level cache hierarchy featuring a 32 KB large 4-way associative data cache and a 32 KB large 2-way associative instruction cache. Evidently, the degree of non-volatility in the data cache is subject to exploration and part of our evaluation. The volatile cache sections are implemented in SRAM, with the non-volatile sections being implemented in STT-RAM technology. As the instruction cache is read-only with read requests expected to be more efficiently served in NVM technology, we will stick to a fully non-volatile instruction cache throughout this paper and instead focus on the data cache. With regard to the main memory, a 4 GB non-volatile PCRAM memory is applied, modelled after the technological characteristics given in [6]. While this may be larger than most RAMs found in conventional MCUs, it allows us to model a non-volatile main memory with fairly accurate parameters regarding latencies and energy consumption, while also providing enough space for a small embedded OS in order to simplify our simulation workflow.

The parameters, including latency and energy parameters for the cache are further displayed in Table 1. We obtained the cache parameters using NVSim [8] by performing simulations of our own as well as comparing the results against works more focused on accurately simulating STT-RAM caches such as [10], as accurate measurements on NVM parameters are hard to come by. Note that in our evaluation, most cache misses also go hand in hand with a cache write, as even for a read miss to be cached, a cache line is eventually filled, i.e., written with the requested data.

5.3 Applications

To measure the performance of the different hybrid cache designs, we employ three different benchmarks, each with their own access pattern characteristics.

AES As a state-of-the-art block cipher, the Advanced Encryption Standard (AES) is a suitable application to be running on MCUs. Concerning the characteristics of the application, here, during the multiple rounds of shifting rows and mixing columns within a block, we have data that is first read and subsequently overwritten. Furthermore, as the so-called S-Box provides data that is used in every round of the encryption in order to determine the substitution of the input bytes, we also have data that is highly read-intensive without ever being overwritten. We employ the AES implementation provided by [11] as our first test application, encrypting a 512 KB large input file with a 32 Byte large key.

Image Processing Since image processing is one of the most common tasks for embedded sensors, it is a natural field of interest for our evaluation. We employ a conventional 2D-convolution on a 640×640 large grayscale input image using a 3×3 large kernel. Regarding the access pattern, we have the special case that all data during the main execution loop is either read-only or write-only. More precisely, the input image and kernel are never overwritten, with the output image never being required to be read.

Merge Sort As sorting data is a frequently required task in many embedded applications, it is used as our third benchmark application. Since it can be easily parallelized for future works and provides a third type of access pattern, we have selected merge sort for an input array containing 65,536 integers as the base of our sorting algorithm. First, the input array is subsequently split into two equally sized sub-arrays until the sub-arrays solely contain a single entry, the split being a write-intensive task. In the second phase, the original input array is overwritten by subsequently merging the smaller sub-arrays in the correct order. All involved arrays are therefore both read and written, with the algorithm generally being fairly write-intensive, especially compared to other sorting algorithms.

All applications will be compiled into unikernels using the modular, lightweight OS Unikraft [13] to support basic system calls such as `malloc` and allow for the abstraction of virtual addresses. In order to not distort our evaluation with memory accesses caused by the Unikraft boot process, all statistics generated by gem5 will be reset between completing the Unikraft boot and starting the actual application.

6 EXPERIMENTS UNDER CONTINUOUS POWER SUPPLIES

For the first set of experiments, we compare systems according to the parameters given in Section 5.2 with the ratio of non-volatile cache lines being set to either 25%, 50% or 75%. We further compare the two architecture-aware replacement policies, WI and CM, presented in Section 4 against a conventional LRU approach. The power supply is assumed to be stable for the experiments carried out in this section. A design employing no non-volatile cache lines and the LRU policy serves as a baseline approach. The first performance indicator is provided by normalizing the dynamic energy

	Read Latency	Write Latency	Read Energy (per access)	Write Energy (per access)
SRAM Cache	2 Cycles @240 MHz	2 Cycles @240 MHz	0.009 nJ	0.009 nJ
STT-RAM Cache	2 Cycles @240 MHz	8 Cycles @240 MHz	0.007 nJ	0.056 nJ
PCRAM Main Memory	48 Cycles @400 MHz (tRCD)		0.081 nJ	1.685 nJ

Table 1: Characterized read/write latencies and average read/write access energies of considered memories.

	AES	Image Processing	Merge Sort
θ_{wi}	100	10	-2
θ_{cm}	8	10	150

Table 2: Replacement policy thresholds for our three test applications.

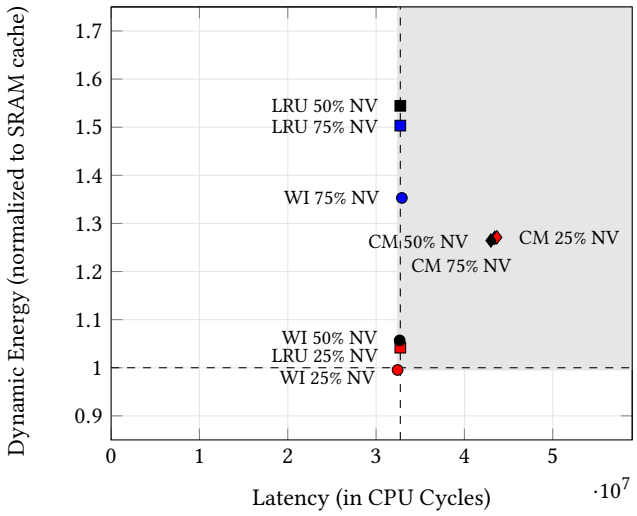

Figure 4: Comparison of different architectural design options for the AES application. The performance of the baseline architecture is displayed by dashed lines. The dominated space is shown in gray.

consumption caused by accesses to the memory hierarchy of the tested approach, to the dynamic energy consumption of the baseline's memory hierarchy employing the purely volatile cache. The second performance indicator is given by the number of CPU cycles needed to run the applications presented in Section 5.3 to completion. The thresholds θ_{wi} and θ_{cm} for the WI and CM replacement policies, respectively, have been set to suitable values determined statically and are named in Table 2.

6.1 Discussion of Results for AES

Figure 4 displays the results for the AES application. The dashed lines indicate the normalized dynamic energy consumption and the latency of the baseline. Considering the results, a number of observations can be made. First, the design point featuring the WI policy and a non-volatility degree of 25% proves to be the optimum among the compared approaches. Whereas for the WI policy, high degrees of non-volatility generate additional energy overhead, this trend does not apply to the other tested replacement policies. In fact, for the CM policy, the degree of non-volatility does not appear to significantly influence energy consumption at all. This is mainly due to the swap mechanism avoiding unsuitable types of access to the two cache sections, regardless of the size of the section. On the one hand, the swap mechanism also generates additional energy and latency overhead as the cache sets are blocked for the time of the swap. On the other hand, for a high degree of non-volatility of 75%, the advantage of actively swapping cache sections balances out the swap overhead as WI with 75% non-volatile cache lines is being outperformed in the energy dimension. The disadvantage of the WI policy is that it can only place data to a different section after it is evicted, and the state machines have transitioned from the write- to the read-intensive states, or vice versa. Especially for a larger non-volatile section where evictions are more sparse, write-intensive data erroneously placed in the non-volatile section will thus generate more NVM write overhead compared to the CM policy, where it will always be swapped out after a fixed number of unsuitable accesses.

For LRU, the trend with regard to the degree of non-volatility is also not linear. In this case, where the policy does not consider the underlying memory technology, from an outside perspective, an inherent arbitrariness regarding suitable section placements can be observed, making it hard to predict the performance. In fact, the approach combining LRU with 25% of all cache lines being non-volatile manages to perform only 6.3% of its writes in the non-volatile section. Inversely, the approach with 50% non-volatile cache lines performs more than 86% of its writes in the non-volatile section, while simultaneously only serving 31.4% of its reads in the non-volatile section. Combining LRU with a 50/50 volatility split thus performs even worse than LRU with a 75% degree of non-volatility, where 98.4% of the read requests are served efficiently in the non-volatile section.

Moreover, the optimal point of WI applied to a 25% non-volatile cache manages to outperform the baseline in both the energy and latency dimensions. This is achieved by exploiting efficient non-volatile reads and the strict separation of the two cache sections helping to occasionally realize more suitable replacement decisions, as the overall time spent dealing with the handling of misses decreases compared to the baseline.

However, the points in the dominated space can also not be dismissed entirely, as the degree of non-volatility does not necessarily have to be a design decision. In cases where the chip size is strictly limited by design, a high degree of non-volatility may

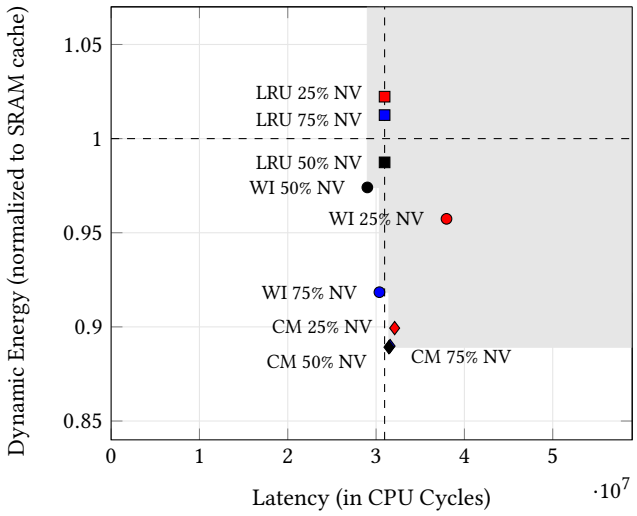

Figure 5: Comparison of different architectural design options for the image processing application. The performance of the baseline architecture is displayed by dashed lines. The dominated space is shown in gray.

be the only possibility to realize caches of a certain size due to NVM's high density. When applying a high degree of non-volatility becomes a necessity rather than a choice, we can also see different replacement policies, in this case CM, becoming a suitable choice of replacement policy. For a non-volatility degree of 75%, we have to further apply weights to the different performance indicators in order to determine the most suitable design choice, as, when restricting the analysis to this non-volatility degree, both WI and CM provide Pareto-optimal design points. Thus, even for a single point dominating the overall design space in our evaluation, the design decision might not always be straightforward.

6.2 Discussion of Results for Image Processing

With regard to the image processing application, the Pareto front is shown in Figure 5. Here, a lot of the requests to the cache come in the form of read requests. Therefore, energy savings after introducing NVM technology can be expected, as can be observed by setting the degree of non-volatility to 25%. While revolving around the small non-volatile section generates additional latencies, especially for the WI policy not featuring proactive migration, exploiting the energy-efficient non-volatile reads leads to energy savings in the big picture. Setting the degree of non-volatility to 75% further emphasizes this effect, even for LRU, although LRU does not manage to achieve energy savings compared to the baseline approach. Here, WI even manages to achieve a lower latency compared to the baseline. The explanation behind this lies in the characteristics of the image processing application in combination with the WI policy. More precisely, due to the strict WI placement decisions and the application processing parts of the data read-only and others write-only, the policy manages to perform suitable predictions fairly easily, placing input data in the non-volatile and output data in the volatile cache section. Therefore, data from the input image does not tend to be unsuitably replaced by data for the output image, leading to lower average latencies when requiring input data for the next computation.

Arguably, two outliers can be identified with regard to the WI and LRU points featuring a 50/50 split in terms of volatile and non-volatile cache lines. For LRU, intelligent placement decisions leading to the lowest energy consumption among the LRU design points are again performed unconsciously, as the policy is not architecture-aware. For WI and 50% of cache lines being non-volatile, the chain of events leading to the evaluation point, not fitting in line with the WI points featuring a different degree of non-volatility, is more complex. Initially, the WI policy places all data in the volatile section, as it has not gathered enough information to perform a suitable prediction. Therefore, at the beginning of the execution, input data is only placed in the non-volatile section after being evicted at least once. The smaller the volatile section is scaled, the higher the miss rates are at the beginning of the execution, before eventually distributing the data to the corresponding suitable sections. Many cache misses in a short amount of time further lead to a higher latency in order to handle the miss, compared to misses stretched over a longer period of time. The point featuring 75% non-volatile cache lines is thus slower at the beginning of the execution than the one featuring 25% non-volatile cache lines. However, due to the small non-volatile section, the 25% non-volatile approach will generate more misses throughout the main part of the execution. For the observed execution time, applying a 50% non-volatile design balances out these two drawbacks, hence why it features the lowest overall latency. Nevertheless, with regard to the energy consumption, over the observed runtime, the evenly split approach combines the drawbacks of both other non-volatility degrees to its disadvantage. More precisely, due to the smaller non-volatile cache section, it cannot exploit the non-volatile section as efficiently as the 75% non-volatile approach, while at the same time not achieving the low number of non-volatile writes of the 25% approach. The higher number of non-volatile writes for a higher degree of non-volatility can be explained by mispredictions, as output data wrongly predicted to be read-intensive due to, e.g., collisions in the prediction table will remain in the non-volatile section for a longer time, causing write overhead before eventually being evicted.

Due to the data being split into read- and write-only sections, CM's migration will never actively migrate data to an unsuitable section, with WI mispredictions being more severe. In case data initially unsuitably placed needs to be moved to the more fitting cache section, CM can still migrate a second set of data, i.e., the swap target, to an unsuitable section. However, this data "passively" migrated to the section in which it is accessed less efficiently is quickly swapped back after hitting the threshold θ_{cm}. Throughout all tested degrees of non-volatility, CM thus manages to significantly reduce the number of non-volatile writes, leading to an energy consumption below the WI comparison points. Moreover, especially for a low degree of non-volatility of 25%, WI generates higher miss rates as the data to be read from the non-volatile section will frequently replace each other. For this non-volatility degree, CM's swap mechanism provides an advantage as input data in the non-volatile section is swapped back to the volatile section rather than evicted after input data in the volatile section has hit the threshold

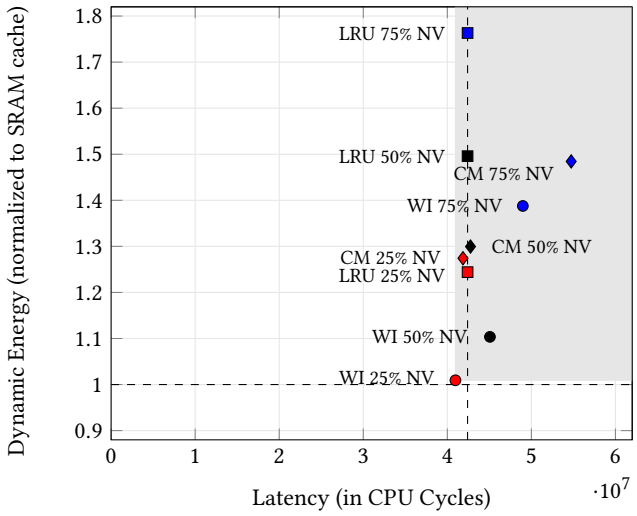

Figure 6: Comparison of different architectural design options for the merge sort application. The performance of the baseline architecture is displayed by dashed lines. The dominated space is shown in gray.

θ_{cm}, leading to a lower miss rate and thus lower latency compared to WI.

In summary, for this application, we can clearly see that the performance is heavily influenced by the interplay of the application characteristics with the replacement policy behavior and the underlying degree of non-volatility under specific circumstances.

6.3 Discussion of Results for Merge Sort

Concerning the merge sort application results displayed in Figure 6, a clear trend can be observed when changing the degree of non-volatility for a fixed replacement policy. Regardless of the chosen replacement policy, both the energy consumption and latency rise when applying a higher degree of non-volatility. While for LRU this is to be expected, as the architecture-unaware replacement policy places the data of the write-intensive application in the non-volatile section more frequently, for WI and CM the explanation differs.

For WI, the higher degree of non-volatility also leads to a higher number of non-volatile writes. This is partially due to write-intensive data unsuitably placed in the non-volatile section remaining there for longer, as it will take longer for data to eventually be evicted, with WI not being able to swap data between cache sections without the data being evicted first. Simultaneously, since here, cache accesses mainly revolve around the volatile section, a higher degree of non-volatility leads to higher miss rates, leading to a higher overall latency on top of generating additional energy overhead due to the increased number of main memory accesses.

For CM, the trend regarding energy consumption and latency is the same as for WI. However, due to CM allowing for data to swap cache sections proactively rather than waiting for eviction, the number of non-volatile writes to the cache barely increases with an increasing degree of non-volatility. The energy consumption increase is thus mainly owing to increased miss rates and, therefore, an increased number of main memory accesses. This also reflects the fact that when increasing the degree of non-volatility, the energy overhead increases less dramatically for the CM policy compared to WI. For a low degree of non-volatility of 25%, WI provides an optimal solution among the compared hybrid designs, beating the baseline in the latency dimension as it achieves a lower miss rate and lower miss latencies. Similar to the image processing application, the hybrid design allows for splitting data into the two cache sections, which would otherwise have replaced each other in a conventional cache design. Nevertheless, for 50% of cache lines being non-volatile, none of the three tested policies manages to dominate one of the other two, as the underlying degree of non-volatility influences the performance of each replacement policy differently.

6.4 Intermediate Summary

To summarize, even though for an application like merge sort, the trend concerning a varying degree of non-volatility for a fixed replacement policy might be clear, there is no clear trend concerning a varying replacement policy for a fixed degree of non-volatility. A design space exploration depending on the design objectives and hardware prerequisites is thus still required. Furthermore, for all three applications, we could observe that even under stable power supplies, the introduction of NVM to the cache hierarchy can provide advantages in both the latency and the energy consumption direction, depending on the characteristics of the application. Additionally, the architecture-aware replacement policies have proven their worth, outperforming LRU in the majority of the studied cases.

7 EXPERIMENTS UNDER INTERMITTENT POWER SUPPLIES

In this section, we repeat the experiments with the architectures and applications presented in Section 5.2 and Section 5.3, respectively. The replacement policies presented in Section 4 and varying degrees of non-volatility are again the subject of our comparison. For the two architecture-aware replacement policies, we employ the same threshold settings as named in Table 2. However, here, the power supply is not assumed to be stable. Instead, every 2,500,000 CPU cycles, we trigger a power outage event in the simulation, leading to the architecture starting the backup process as outlined in Section 3.2. While a real-world system would remain in a sleep state until enough energy is harvested to continue the execution, this hibernation time is left out of our analysis. The latency and energy consumption performance indicators are thus determined solely by the execution and backup phases. The baseline approach employing a conventional SRAM cache is subject to the same power outage pattern. As discussed in Section 3.3, the behavior of the metadata needed for the replacement policies during power outages remains an open question. For our experiments, we have decided on keeping all metadata specific to a cache line, i.e., the LRU timing information, the WI cost field, and the CM ric_i, wic_i and $conf_i$ fields volatile. Conversely, the WI prediction table and CM previous placement table can be implemented in NVM as they are expected to be written far less frequently.

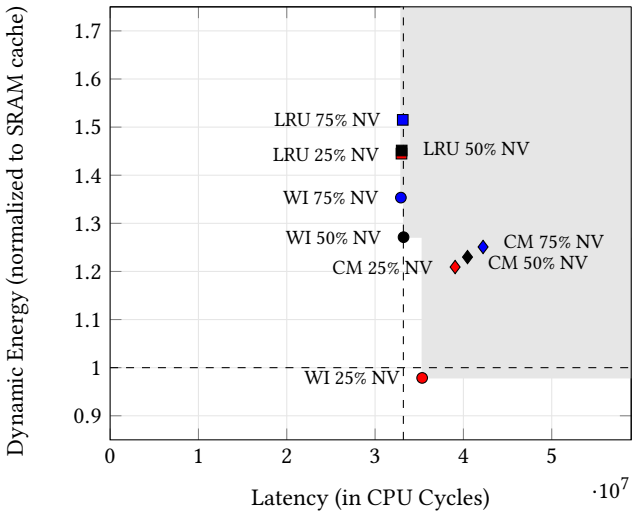

Figure 7: Comparison of different architectural design options for the AES application, including power outages. The performance of the baseline architecture is displayed by dashed lines. The dominated space is shown in gray.

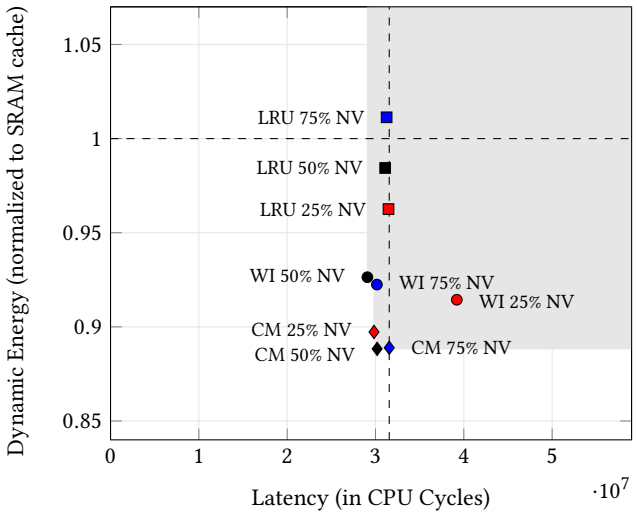

Figure 8: Comparison of different architectural design options for the image processing application, including power outages. The performance of the baseline architecture is displayed by dashed lines. The dominated space is shown in gray.

7.1 Discussion of Results for AES

With regard to the results for the AES application, depicted in Figure 7, both expected and unexpected results can be reported. Concerning LRU, while the average number of cycles required to back up the system state decreases with an increasing degree of non-volatility, the overall latency barely decreases, only narrowly beating the baseline. More precisely, the average number of backup cycles is reduced to 80.3% of the baseline approach with 25% non-volatile cache lines and reduced to 21.6% of the baseline in the case of 75% of cache lines being non-volatile. However, as all metadata regarding the LRU replacement decisions is lost after a power outage, valid non-volatile cache lines may be chosen as a replacement candidate, despite there still being a sufficient amount of invalid volatile cache lines. With an increasing degree of non-volatility, these unsuitable replacement decisions become more frequent, diminishing the latency advantage. This further causes additional energy overhead as writebacks avoided during the backup process are instead performed after regaining power due to the unsuitable replacement policy decisions.

For WI, the effect of power outages on the latency is more straightforward, as the latency decreases with an increasing degree of non-volatility, beating the baseline in the case of 75% of all cache lines being non-volatile. Excluding the baseline, all WI design points are part of the Pareto front, with energy consumption increasing with the degree of non-volatility, which can also be observed under a stable power supply. However, for the intermittent-aware CM policy, the design points employing a lower degree of non-volatility dominate the CM design points with a higher degree of non-volatility, while remaining dominated by the WI designs. Here, the latency required to handle cache misses decreases with a decreasing degree of non-volatility. CM will attempt to keep frequently accessed data in the non-volatile section during the backup process. For a smaller non-volatile section, the effect of losing more of the important data after a power outage thus may not be as pronounced for this policy compared to WI. Furthermore, subsequent cache misses initially requiring placement in the volatile section can be placed in a larger, now completely invalid, volatile cache section, thus not requiring costly writebacks as frequently as compared to a higher degree of non-volatility, i.e., a smaller volatile cache section. Note that the distribution of accesses per section remains fairly similar throughout all degrees of non-volatility for the CM policy due to the proactive swap mechanism.

Nevertheless, the intermittent-aware CM policy does not manage to outperform the intermittent-unaware WI policy, with CM only being able to provide Pareto-optimal points on the condition of a fixed degree of non-volatility of either 50% or 75%. Moreover, while the introduction of NVM can provide advantages, for this application, it does not manage to consistently outbalance the introduced overhead, despite an unstable power supply.

7.2 Discussion of Results for Image Processing

Due to the characteristics of the image processing application, even under stable power supplies we already managed to achieve significant energy savings by introducing hybrid caches. When subjected to power outages, as depicted in Figure 8, the advantage of hybrid caches with a suitable replacement policy is emphasized further. More precisely, while LRU remains fairly unsuitable, depending on the degree of non-volatility, both architecture-aware policies manage to not only achieve energy savings but also latency gains compared to the baseline approach. Notably, the design points employing the CM policy now achieve latency gains compared to the baseline, whereas CM invoked a minor latency overhead under a continuous power supply. With one exception, across all tested

degrees of non-volatility, CM achieves the lowest number of average cycles required to back up the system state. The exception can be found by applying LRU to a non-volatility degree of 75%. Here, LRU can place most of the output image data, the only data being modified and thus requiring a potential backup, in the non-volatile section, leading to short backup times. Nevertheless, this also leads to significant energy overhead during the execution phases, which is reflected by LRU with a 75% degree of non-volatility being the only design point failing to beat the baseline in the energy dimension.

An interesting side effect of introducing power outages can be observed by analyzing the design points employing the WI policy and the lower degrees of non-volatility of 25% and 50%. For these design points, compared against the WI policy approach with the higher degree of non-volatility of 75% the normalized dynamic energy consumption is reduced far more significantly after introducing power outages. As output image data stored in the volatile section gets evicted from its location in case a power outage is encountered, the cost field, indicating the write intensity, might be set at a far lower value compared to when the data would have been evicted in the continuous execution. Therefore, the cost value might be below the threshold θ_{wi}, leading to the corresponding write intensity state machine erroneously moving to the next more read-intensive state. Data for the output image is thus more frequently unsuitably placed in the non-volatile section, requiring further evictions until the state machine can readjust itself to a more write-intensive state. Especially for the highest degree of non-volatility of 75% this can be hazardous. The larger the non-volatile section, once erroneously placed, the longer the output data will remain there, causing additional non-volatile writes until it is eventually evicted. Additionally, due to the smaller volatile section, the output data tends to have lower values in the cost field, as they are evicted more frequently in regular execution, further emphasizing the effect of the undesired drift towards the read-intensive states.

In summary, for the read-intensive image processing application, cache hybridization has demonstrated its potential, even more so under unstable power supplies. However, as the Pareto front consists of both architecture-aware policies under varying degrees of non-volatility, further investigation regarding the design objectives is required in order to decide on a specific design.

7.3 Discussion of Results for Merge Sort

For the merge sort application, the experimental results after including power outages are illustrated in Figure 9. Whereas under a continuous power supply, combining WI with a 25% degree of non-volatility was identified as the optimum among the examined hybrid approaches, after introducing power outages, multiple hybrid design points are part of the Pareto front. Here, CM, in combination with a low degree of non-volatility of either 25% or 50%, manages to achieve additional latency gains compared to WI under the same non-volatility degrees. These latency gains are in spite of CM requiring a higher average number of cycles to secure the system state during a power outage compared to WI. In fact, CM requires 127% and 111% of WI's cycles to back up the system for a degree of non-volatility of 25% and 50%, respectively. The latency advantage regarding CM is thus gained during the regular

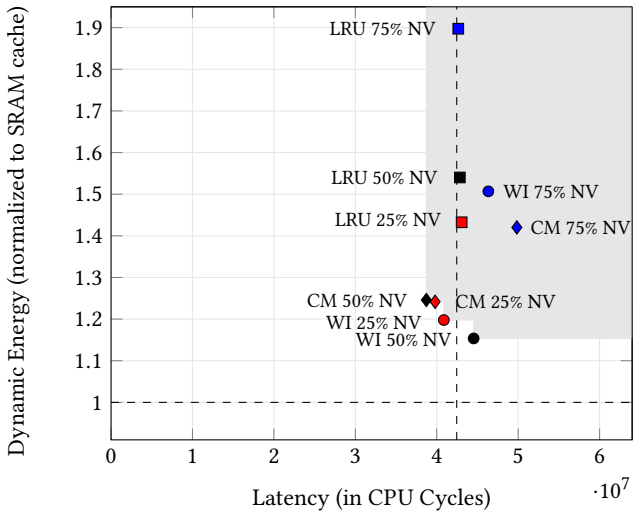

Figure 9: Comparison of different architectural design options for the merge sort application, including power outages. The performance of the baseline architecture is displayed by dashed lines. The dominated space is shown in gray.

execution phase, as CM achieves lower miss rates compared to WI. This can partially be explained by CM's backup mechanism during power outages in combination with the high write intensity of the merge sort application. Since the merge sort implementation features a large amount of write-intensive data, the architecture-aware replacement policies will tend to place a large portion of the data in the volatile section. As the volatile cache lines are lost from the cache after a power outage, this leads to a high miss rate after regaining power. However, the intermittent-aware CM policy features a mechanism for which frequently accessed cache lines are swapped to the non-volatile section during a power outage and thus saved in favor of less frequently accessed data. Whereas WI loses a large amount of frequently accessed data from the cache during a power outage, CM can keep the write-intensive data important to this application closer to the CPU, thus saving time after regaining power. This backup to the non-volatile cache section also comes at the cost of additional non-volatile writes before CM eventually swaps write-intensive data back to the volatile section. Nevertheless, the determination of what CM considers important data, i.e., data to be saved in the non-volatile section of the observed cache, is merely a heuristic based around a cache line's confidence field, which is reset after a power outage. Therefore, it is difficult to assess how consciously CM performs suitable backup decisions for this application.

The second major change in the results after introducing power outages can be identified by examining the WI design points with a 25% degree of non-volatility. While still being Pareto optimal, the energy overhead of the WI 25% non-volatility approach has increased for the intermittent power supply, being outperformed by the design point carrying the higher 50% non-volatility degree. As mentioned in Section 7.2, WI under intermittent power supplies

tends to erroneously update the write intensity state machines to the more read-intensive states. On the one hand, for the read-intensive image processing application, this is a larger issue for a high degree of non-volatility, as for a smaller non-volatile section, the few misplaced write-intensive data are quickly evicted, thus swiftly switching back to more write-intensive states. On the other hand, for merge sort, this changes due to different application characteristics. Here, as a larger volatile section is more heavily occupied with write-intensive data compared to the image processing application, a higher number of write-intensive data are subject to the erroneous drift towards a read-intensive prediction. For a low degree of non-volatility, these data erroneously placed in the non-volatile section still tend to be replaced quicker compared to a higher degree of non-volatility. However, for merge sort, this cannot balance out the increase in non-volatile writes caused by the aforementioned larger amount of mispredictions for a low degree of non-volatility, i.e., a larger volatile section.

7.4 Summary

The above results further underline the argument that the introduction of power outages can have a range of specific effects on the performance of hybrid caches, depending on the selected replacement policy and the degree of non-volatility. Furthermore, these effects are difficult to predict without proper experiments and analyses. Simultaneously, the introduction of hybrid caches to the memory hierarchy has proven to be able to provide benefits compared to a conventional SRAM cache design. However, for all tested applications, despite consistently achieving lower backup times, even under intermittent power supplies, the overhead caused by NVM can be significant, thus requiring care in the design process. Nevertheless, the worst case concerning the energy required for the backup during a power outage, to which an intermittently powered system has to adhere, improves with the degree of non-volatility.

8 CONCLUSION AND FUTURE RESEARCH

In conclusion, in this paper, we have demonstrated how the introduction of hybrid volatile/non-volatile caches to the memory hierarchy can provide a suitable alternative to conventional cache memory designs. Depending on the application, even for a continuous power supply, the examined hybrid cache designs have been able to achieve both lower latencies and lower energy consumptions than conventional SRAM caches. However, it has also become obvious that a lot of care has to be put into designing such systems. More precisely, the combination of application and architecture characteristics, as well as the behavior of the replacement policy, can lead to a number of hard-to-predict cache effects that are beneficial or harmful to the overall performance. Therefore, despite architecture-aware replacement policies having shown their worth in our evaluations, we do not expect the development of another replacement policy for hybrid caches to be the most important target for future research. Instead, we expect future research focusing on characterizing these cache effects to be the more promising direction. Especially for the purpose of intermittent computing, a whole different range of cache effects comes into play, as NVM overhead needs to be balanced against the advantage of keeping data closer to the CPU after a power outage. The long-term goal would thus lead to an automated design tool that can aid in the process of finding efficient systems featuring hybridized memory hierarchies according to classifiable application characteristics, user-provided code annotations, known architecture traits and expected power supply behavior.

ACKNOWLEDGMENTS

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program "SPP 2377: Disruptive Memory Technologies" under project *Co-Design of Persistent, Energy-efficient and High-speed Embedded Processor Systems with Hybrid Volatility Memory Organisation* (HYPNOS, project number 502213043)

REFERENCES

[1] 2012. Micron Announces Availability of Phase Change Memory for Mobile Devices. https://investors.micron.com/news-releases/news-release-details/micron-announces-availability-phase-change-memory-mobile-devices?ReleaseID=692563. Accessed: 2024-05-15.

[2] Junwhan Ahn, Sungjoo Yoo, and Kiyoung Choi. 2016. Prediction Hybrid Cache: An Energy-Efficient STT-RAM Cache Architecture. *IEEE Trans. Comput.* 65, 3 (2016), 940–951. https://doi.org/10.1109/TC.2015.2435772

[3] SatyaJaswanth Badri, Mukesh Saini, and Neeraj Goel. 2023. Efficient placement and migration policies for an STT-RAM based hybrid L1 cache for intermittently powered systems. *Design Automation for Embedded Systems* (May 2023). https://doi.org/10.1007/s10617-023-09272-w

[4] Domenico Balsamo, Alex S. Weddell, Geoff V. Merrett, Bashir M. Al-Hashimi, Davide Brunelli, and Luca Benini. 2015. Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems. *IEEE Embedded Systems Letters* 7, 1 (2015), 15–18. https://doi.org/10.1109/LES.2014.2371494

[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2 (aug 2011), 1–7. https://doi.org/10.1145/2024716.2024718

[6] Youngdon Choi, Ickhyun Song, Mu-Hui Park, Hoeju Chung, Sanghoan Chang, Beakhyoung Cho, Jinyoung Kim, Younghoon Oh, Duckmin Kwon, Jung Sunwoo, Junho Shin, Yoohwan Rho, Changsoo Lee, Min Gu Kang, Jaeyun Lee, Yongjin Kwon, Soehee Kim, Jaehwan Kim, Yong-Jun Lee, Qi Wang, Sooho Cha, Sujin Ahn, Hideki Horii, Jaewook Lee, Kisung Kim, Hansung Joo, Kwangjin Lee, Yeong-Taek Lee, Jeihwan Yoo, and Gitae Jeong. 2012. A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth. In *2012 IEEE International Solid-State Circuits Conference*. 46–48. https://doi.org/10.1109/ISSCC.2012.6176872

[7] Gaurav Dhiman, Raid Ayoub, and Tajana Rosing. 2009. PDRAM: a hybrid PRAM and DRAM main memory system. In *Proceedings of the 46th Annual Design Automation Conference* (San Francisco, California) *(DAC '09)*. Association for Computing Machinery, New York, NY, USA, 664–469. https://doi.org/10.1145/1629911.1630086

[8] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. 2012. 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 (2012), 994–1007. https://doi.org/10.1109/TCAD.2012.2185930

[9] Carlos Escuin, Asif Ali Khan, Pablo Ibañez, Teresa Monreal, Victor Viñals, and Jeronimo Castrillon. 2022. HyCSim: A rapid design space exploration tool for emerging hybrid last-level caches. In *System Engineering for Constrained Embedded Systems* (Budapest, Hungary) *(DroneSE and RAPIDO)*. Association for Computing Machinery, New York, NY, USA, 53–58. https://doi.org/10.1145/3522784.3522801

[10] Dhruv Gajaria and Tosiron Adegbija. 2022. Evaluating the performance and energy of STT-RAM caches for real-world wearable workloads. *Future Generation Computer Systems* 136 (2022), 231–240. https://doi.org/10.1016/j.future.2022.05.023

[11] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*. 3–14. https://doi.org/10.1109/WWC.2001.990739

[12] Mohsen Imani, Shruti Patil, and Tajana Rosing. 2016. Low power data-aware STT-RAM based hybrid cache architecture. In *2016 17th International Symposium on Quality Electronic Design (ISQED)*. 88–94. https://doi.org/10.1109/ISQED.2016.7479181

[13] Simon Kuenzer, Vlad-Andrei Bădoiu, Hugo Lefeuvre, Sharan Santhanam, Alexander Jung, Gaulthier Gain, Cyril Soldani, Costin Lupu, Ştefan Teodorescu, Costi

Răducanu, Cristian Banu, Laurent Mathy, Răzvan Deaconescu, Costin Raiciu, and Felipe Huici. 2021. Unikraft: fast, specialized unikernels the easy way. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 376–394. https://doi.org/10.1145/3447786.3456248

[14] Jing-Yuan Luo, Hsiang-Yun Cheng, Ing-Chao Lin, and Da-Wei Chang. 2019. TAP: Reducing the Energy of Asymmetric Hybrid Last-Level Cache via Thrashing Aware Placement and Migration. *IEEE Trans. Comput.* 68, 12 (2019), 1704–1719. https://doi.org/10.1109/TC.2019.2917208

[15] Dong Ma, Guohao Lan, Mahbub Hassan, Wen Hu, and Sajal K. Das. 2020. Sensing, Computing, and Communications for Energy Harvesting IoTs: A Survey. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 1222–1250. https://doi.org/10.1109/COMST.2019.2962526

[16] Kaisheng Ma, Xueqing Li, Karthik Swaminathan, Yang Zheng, Shuangchen Li, Yongpan Liu, Yuan Xie, John Jack Sampson, and Vijaykrishnan Narayanan. 2016. Nonvolatile Processor Architectures: Efficient, Reliable Progress with Unstable Power. *IEEE Micro* 36, 3 (2016), 72–83. https://doi.org/10.1109/MM.2016.35

[17] Dylan McGrath. 2019. Samsung Says It's Shipping 28-nm Embedded MRAM. https://www.eetimes.com/samsung-says-its-shipping-28-nm-embedded-mram/. Accessed: 2024-05-15.

[18] Dushyanth Narayanan and Orion Hodson. 2012. Whole-system persistence. *SIGPLAN Not.* 47, 4 (mar 2012), 401–410. https://doi.org/10.1145/2248487.2151018

[19] Matthew Poremba et al. 2015. NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems. *IEEE Computer Architecture Letters* 14 (2015), 140–143.

[20] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2011. 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* (Newport Beach, California, USA) *(ASPLOS XVI)*. Association for Computing Machinery, New York, NY, USA, 159–170. https://doi.org/10.1145/1950365.1950386

[21] Nour Sayed, Longfei Mao, and Mehdi B. Tahoori. 2021. Dynamic Behavior Predictions for Fast and Efficient Hybrid STT-MRAM Caches. *J. Emerg. Technol. Comput. Syst.* 17, 1, Article 9 (jan 2021), 21 pages. https://doi.org/10.1145/3423135

[22] Satyabrata Sen and Neena Imam. 2019. Machine learning based design space exploration for hybrid main-memory design. In *Proceedings of the International Symposium on Memory Systems* (Washington, District of Columbia, USA) *(MEMSYS '19)*. Association for Computing Machinery, New York, NY, USA, 480–489. https://doi.org/10.1145/3357526.3357544

[23] Sivert T. Sliper, William Wang, Nikos Nikoleris, Alex S. Weddell, Anand Savanth, Pranay Prabhat, and Geoff V. Merrett. 2023. Pragmatic Memory-System Support for Intermittent Computing Using Emerging Nonvolatile Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 1 (2023), 95–108. https://doi.org/10.1109/TCAD.2022.3168263

[24] Clinton W. Smullen, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R. Stan. 2011. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. 50–61. https://doi.org/10.1109/HPCA.2011.5749716

[25] Theo Soriano, David Novo, Guillaume Prenat, Gregory Di Pendina, and Pascal Benoit. 2022. MemCork: Exploration of Hybrid Memory Architectures for Intermittent Computing at the Edge. In *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*. 1–6. https://doi.org/10.1109/VLSI-SoC54400.2022.9939630

[26] Fang Su et al. 2017. Nonvolatile processors: Why is it trending?. In *Proc. of DATE*, David Atienza and Giorgio Di Natale (Eds.). IEEE, 966–971. https://doi.org/10.23919/DATE.2017.7927131

[27] Guangyu Sun, Xiangyu Dong, Yuan Xie, Jian Li, and Yiran Chen. 2009. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. 239–249. https://doi.org/10.1109/HPCA.2009.4798259

[28] Milijana Surbatovich, Brandon Lucia, and Limin Jia. 2020. Towards a formal foundation of intermittent computing. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 163 (nov 2020), 31 pages. https://doi.org/10.1145/3428231

[29] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, and Yuan Xie. 2009. Power and performance of read-write aware Hybrid Caches with non-volatile memories. In *2009 Design, Automation & Test in Europe Conference & Exhibition*. 737–742. https://doi.org/10.1109/DATE.2009.5090762

[30] Mimi Xie, Chen Pan, Youtao Zhang, Jingtong Hu, Yongpan Liu, and Chun Jason Xue. 2019. A Novel STT-RAM-Based Hybrid Cache for Intermittently Powered Processors in IoT Devices. *IEEE Micro* 39, 1 (2019), 24–32. https://doi.org/10.1109/MM.2018.2890257