Achieving Crash Consistency by Employing Persistent L1 Cache

Akshay Krishna Ramanathan[1*] Sara Mahdizadeh Shahri[2*] Yi Xiao[1] Vijaykrishnan Narayanan[1]
(1) The Pennsylvania State University *(2) University of Michigan*
{axr499, ymx5148, vxn9}@psu.edu smahdiz@umich.edu

***Abstract*—Emerging non-volatile memory technologies promise the opportunity for maintaining persistent data in memory. However, providing crash-consistency in such systems can be costly as any update to the persistent data has to reach the persistent domain in a specific order, imposing high overhead. Prior works, proposed solutions both in software (SW) and hardware (HW) to address this problem but fall short to remove this overhead completely. In this work, we propose Non-Volatile Cache (NVC) architecture design that employs a hybrid volatile, non-volatile memory cell employing monolithic 3D and Ferroelectric technology in L1 data cache to guarantee crash consistency with almost no performance overhead. We show that NVC achieves up to 5.1x speedup over state-of-the-art (SOTA) SW undo logging and 11% improvement over SOTA HW solution without yielding the conventional architecture, while incurring 7% hardware overhead.**

***Index Terms*—Non-volatile cache, Persistent applications, Monolithic-3D integration, SRAM, Ferro-electric FET**

I. Introduction

Emerging non-volatile memory technologies like Intel's 3D XPoint allow programmers to maintain persistent data (e.g., files, database tables) in memory and access them using processor load and store instructions. This architecture represents a significant departure from and performance improvements over conventional architecture with a volatile main memory and a separate, slow, block storage device SSDs and HDDs [1]–[3].

Crash consistency refers to ensuring that an application's data is safe and recoverable even in the presence of failures [5]. The persistent (non-volatile) data stored on these PM devices (like files and databases) has to be updated in a specific order to ensure that the data is safe and recoverable in the presence of a system failure [1], [3], [4]. For example, within a database transaction, the updates to the database log have to be persisted before the updates to the actual database table to ensure the crash consistency of the database. However, placing constraints on the order of updates to PM significantly constraints the performance of these persistent applications [4].

Prior work has sought to address this problem through combination of software and/or hardware techniques [1], [4]. The central idea behind these proposals is to batch, coalesce, and reorder updates to PM, where possible without losing any crash consistency guarantees, to improve the performance of these applications. However, the fundamental limitation of all of these works is the fact that for the data to be considered "persisted", it has to be flushed all the way to the memory controller of the corresponding PM device [6]. For example, Intel's `clwb` and `sfence` instructions which ensure that some data has been persisted are shown to be sources of significant performance degradation [4].

In this work, we seek an alternative approach to reducing the overheads of persisting data. We leverage the novel Monolithic 3D (M3D) technology [7] based multi-layer hybrid non-volatile/volatile SRAM memory to effectively move the point of persistence from the memory controller to the L1 cache. Such a change obviates the need for expensive data movements from the L1 cache to the memory controller to make the said data persistent. Moreover, simply making the data persistent in the L1 cache is not sufficient. We still need to ensure the crash consistency of the data being modified. We design a NVC architecture enabled by the M3D integration process by stacking and connecting non-volatile device [8] to the underlying SRAM to ensure crash consistency.

While the NVC and the underlying M3D non-volatile SRAM technology may be used to efficiently implement a number of crash consistency mechanisms, in this paper we focus on the popular undo logging technique. Undo logging is a mechanism that is used to ensure failure atomic updates to PM, i.e., either all or none of a set of updates are reflected in PM in the event of system failure [1], [2]. A common example of a set of updates is a transaction.

Undo logging techniques work by performing four important actions: (i) *backup data* - when a location is modified for the first time within a transaction, a backup of the data is persisted before the data is modified, (ii) *in-place update* - once a backup has been created, the actual data has to be modified and persisted before the end of the transaction, (iii) *commit* - at the end of the transaction, after all the in-place updates have been performed, backups created during the transaction have to be effectively discarded atomically, marking the successful completion of the transaction, and (iv) *restore* - if a transaction fails before completion then all locations that have been modified during the transaction have to be atomically restored using their backups. Our proposed NVC provides an efficient mechanism to perform all of these actions.

While the common case performance achieved through NVC is appealing, the challenge in designing NVC comes from correctly handling the uncommon cases, like cache-block evictions in the middle of a transaction and cache-block invalidations

*Authors contributed equally. We would like to thank A. Kolli at Google, USA for his support and feedback throughout the work. This work was supported in part by National Science Foundation (NSF) under Grant 2008365 and Semiconductor Research Corporation (SRC) Center for Research in Intelligent Storage and Processing in Memory (CRISP).

due to coherence activity. Overall, this paper demonstrates how to leverage M3D non-volatile SRAM to build systems that provide meaningful crash consistency guarantees at virtually no cost compared to prior work, Kiln [3] which changes the conventional architecture by replacing volatile (SRAM) LLC with non-volatile memories, thereby incurring penalties for general execution. Whereas, in NVC, only persistent transactions uses the non-volatile segment of L1 and rest of the general executions uses the volatile segment of the L1 cache.

II. Background: FeFET Integration in M3D Process

Recent advances in non-volatile devices have resulted in their incorporation in main memory, or even as last level caches. The non-volatile memories exhibit similar density as DRAM, while providing substantially low standby power consumption. However, these non-volatile memories suffer from expensive write latency and energy compared to the DRAM. Furthermore, endurance is an additional concern. In order to mitigate these concerns, we design a novel multi-layer hybrid volatile/non-volatile memory so as to use the non-volatile memory only for the logging purpose, and not for regular memory accesses.

When designing multi-layer hybrid memories, the compatibility of the non-volatile device with CMOS is another design issue. The M3D integration of heterogeneous devices needs strict requirement on selection of materials as explained in roadmap on emerging hardware and technology for machine learning [9]. It shows that Hafnium Oxide (HfO_x) based ferroelectric layer have good compatibility with restrictive M3D process flows that in turn enables CMOS transistors and FeFET in multi-layer designs [10]. In this work, we focus on FeFET technology due to its promise for easy CMOS integration using monolithic 3-D integration process which enables multi-layer designs with fine grained inter-layer connections.

The structure of FeFET device is shown in Fig. 1 (a). The FeFET has a similar structure to MOSFET with an additional ferroelectric (FE) layer in the gate stack. Two stable states of polarization in the FE layer provide the non-volatile memory behavior of the FeFET. These states can be switched by appropriate voltage bias on the gate terminal of the device. The FeFET device can be simplified into an equivalent circuit model shown in Fig. 1 (b) with capacitance associated with FE (depicted as C_{fe}) in series with capacitance of the underlying MOSFET (C_{MOS}). Let us consider the operation of this device. When the polarization of FE layer is towards the substrate of the MOSFET, it creates a channel between the source (S) and drain (D) making the device be in "ON" state even when no external gate voltage (V_G) is applied. Similarly, if the polarization is pointing away from the substrate, the channel is not formed making the device be in "OFF" state. By increasing the thickness of the FE (T_{fe}) beyond a certain value introduces a hysteresis, i.e, non volatility in the device characteristics [8].

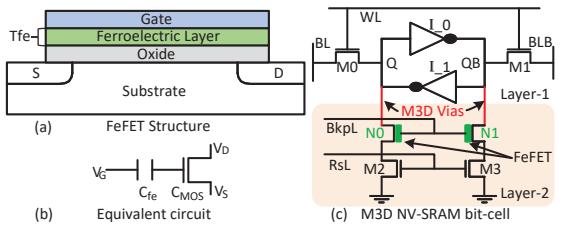


Fig. 1. (a) Structure of a Ferroelectric FinFET (FeFET) device. (b) Equivalent circuit of FeFET. (c) Structure of a M3D Non-Volatile SRAM bit-cell.

III. M3D Non-volatile SRAM Cache

In this section, we will be discussing about the multi-layer non-volatile SRAM cache designs (referred to as M3D NVSRAM cache) using M3D process to reduce the area footprint of additional non-volatile devices.

A. Circuit design

While the M3D NVSRAM circuit structure is similar to Li et. al [11]., we obtain substantial benefits due to transition to our proposed 3D structure Fig. 1(c) by leveraging the recent process advances. The NVSRAM design in a 2-D design requires 65% more area than the std. 6T SRAM bitcell which increases the bitline parasitics, thereby the access costs. In contrast, the proposed M3D NVSRAM design has its bitline parasitics remain unaltered compared to the standard 6T SRAM bitcell, in turn not affecting the access costs. Now, we will be looking into the structure of a M3D NVSRAM bit-cell.

Fig. 1(c) shows the proposed structure of the M3D NVSRAM bit-cell. The layer-1 consists of standard 6-T SRAM bit-cell. The layer-2 consists of the non-volatile elements (N0 and N1) and discharge transistors (M2 and M3). The inter-layer connections are made using the M3D vias shown in the Fig. 1(c). The Word lines (WL) and Bit lines (BL) are used during the memory read/write operations, whereas the Backup line (BkpL) and Restore line (RsL) are used during the back-up and restore operations involving the non-volatile FeFETs. The area of the layer-2 cells is pitch-matched with the area of the layer-1 std. 6T SRAM bit-cell, thereby avoiding area overheads in single dimensional space. The operations of the M3D NVSRAM bit-cell is similar to the [11], mainly the HW governing the control signal is unique and explained in Fig. 4.

B. Memory, backup and restore operations

The read/write operations of the M3D NVSRAM bit-cell is through the standard 6-T SRAM bit-cell in the layer-1. The main feature of M3D NVSRAM bit-cell is the non-volatility added to the SRAM cell, namely backup and restore operations which uses the control lines in layer-2 (BkpL, RsL).

The backup operation involves two steps to save the states into the FeFETs, N0 and N1 and is shown in the Fig. 2(a). Whenever the backup operation is performed, the RsL is set to ground (gnd) thereby the N0/N1's active channel terminal is not connected to the gnd. Then in the first step, the BkpL is set to Vdd and the appropriate FeFET is written to high state (conducting state) according to the node Q/QB state. In Fig. 2(a), the node QB is gnd, hence the N1 has a gate-source potential (Vgs) of Vdd thereby writing it to high state. Whereas, the N0 is not changed since the Vgs is 0. Then, in order to write the complementary value onto the other FeFET, another step is performed by making the BkpL gnd. In this depiction, the N0 experiences Vgs of -Vdd, thereby writing it to low state (not conducting state) and N1 is not changed since its Vgs is

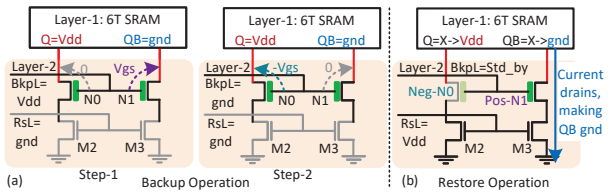


Fig. 2. Working of (a) Backup and (b) Restore operations. Backup operation is a two step process and restore operation requires one step.

0. Therefore, the state of the SRAM cell in layer-1 is stored into the non-volatile elements N0 and N1 using the backup operation. Note that, after backing up a data onto the FeFET, any change in the volatile/SRAM won't modify the previous backed up value unless the next back-up signals are enabled.

The restore operation is a single step process, the value in the FeFETs gets written into it's corresponding SRAM bit cell. The operation is shown in Fig. 2(b), the BkpL is maintained in a standby voltage (Std_by) and the RsL is set to Vdd. The Std_by is the voltage where the FeFETs cannot be disturbed, i.e., cannot be written and used for reading the state of the FeFET. Taking the previous backed up states for the FeFETs, the node QB connects to the gnd through the N1, and the other node turns to Vdd due to the inverter action of the 6T-SRAM.

We extend the bit-level backup and restore operations to cache-block level as *Block-Backup*, *Block-Restore* operations and furthermore, we have a *Cache-Commit* operation which can backup the content of several cache blocks atomically at once with the help of metadata explained in Sec. IV(B). The activation signals (BkpL & RsL) for the backup and restore operations are handled by the finite state machine (FSM) described in Fig. 4 which will be explained in Sec. IV(B). Also, for ensuring atomicity of backup and restore operations during any power failure, the power supply unit has sufficient capacitance for ensuring the successful completion of the write operations on the non-volatile part of the memory cell. This ensures robustness to unanticipated power failures.

IV. NVC Architecture

Fig. 3(a) illustrates the high-level architecture design of NVC in which L1 data cache, extra memory space for thread counter value (TCNT#) and a small persistent write-back buffer, which we call persistent queue (PQ), are all persistent enabled by the M3D NVSRAM design in addition to the persistent main memory PCM (shown as PM).

During the execution of a transaction (example showed in Fig. 5), NVC uses *Block-Backup* to create a persistent copy for

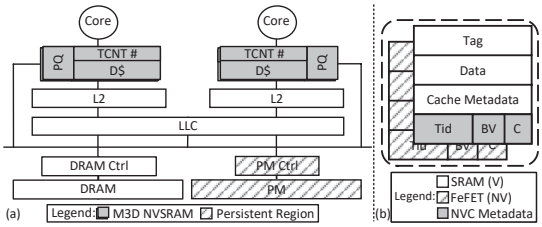


Fig. 3. (a) NVC high-level overview. (b) NVC logical view of L1 data cache-block. Each cache-block is equipped with NVC metadata to make tracking the latest persistent version of a data feasible in the system.

the cache-block before L1 data cache could service the update requests. When the execution of the transaction is completed, NVC issues *Cache-Commit* to persist all cache-blocks that were modified during the transaction atomically in the non-volatile segment of the corresponding cache-blocks. Consequently, NVC guarantees that either all or none of updates in the transaction are visible in the presence of failure with almost no performance cost in comparison to prior works [2], [3]. First, NVC does not require to create and persist a copy of the data (log) in PM. Second, the in-place update is not required to be persisted in PM and can be done in L1 data cache in conjunction with committing the transaction. Third, upon committing a transaction, there is no need to create and persist a special log in PM or flush all the modified cache-blocks to the point of persistency.

A. ISA Extension

To specify the boundaries of a failure atomic region, we extended the ISA with a pair of new instructions, FAR_BEGIN and FAR_END, similar to what prior work proposed [18]. Using this instructions, we can identify the region of interest that should be persisted and it enables correct transitions in the FSM (Fig.4). We assume race-free data access through the use of appropriate concurrency control mechanisms alongside the proposed instructions are employed to add isolation property to the failure atomic region. The extended instructions are applied to the program through a fairly simple compiler pass to identify the failure atomic regions and emit the instructions.

B. L1 Data Cache

Providing crash consistency at L1 Data Cache. Technically, we can use M3D NVSRAM in any level of cache hierarchy to provide crash consistency but we chose only L1 data cache because of the following reasons: (i) lower hardware overhead, (ii) inexpensive commit operation and lower implementation complexity, due to smaller L1 size, (iii) no unified LLC assumption as Kiln [3], orthogonal to current systems.

Fig. 3(b) illustrates the logical overview of L1 data cache-blocks in NVC, (i) a volatile component using SRAM (denoted as *V*) and (ii) a non-volatile component using FeFET (denoted as *NV*). As a result, the NVC uses L1 data cache as the conventional volatile cache and also provides non-volatility for PM instructions. Each L1 data cache-block maintains a minimal metadata to track the latest persistent version of data in the system. Backup valid bit (BV) identifies if the cache-block's NV segment contains a valid data and change in the value of it needs to be persisted through *Block-Backup*. Also, the id of the thread (Tid) that is responsible for modifying the block is kept in the block. Currently, our design supports two threads per core. Tid is a volatile metadata and upon changing it, there is no need for the new value to be persisted as it is meant for selective bits of the *Cache-Commit*. In addition, since NVC has to extend the coherence protocol to provide transferring the persistent copy of the data between cores, a single bit (C) is dedicated to identify whether if the block can be committed or not. C bit also is considered a volatile metadata, similar to Tid, and does not need to be persisted upon modification.

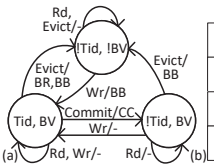
NVC uses the described metadata to interpret the state of a cache-block that belongs to PM (not for regular reads/writes) and ensures crash consistency.The behaviour of L1 data cache will stay the same as conventional caches for requests to blocks residing in volatile memory. Fig. 4(a) illustrates the state of a L1 data cache-block with regards to different events in the cache. Providing an example to interpret the state machine, assume the current state of the cache-block is in "!Tid, BV" state and it is going to be evicted. Then, the HW logic governing the FSM will execute the *Block-Backup* operation and then change the state to "Tid, BV". The reads and writes of the cache-block belonging to PM does not require the backup and restore operations, whereas the eviction and commits requires it.

Upon eviction: In the common case, the NVC handles read, write and commit requests to the L1 data cache in a very straight-forward manner as shown in the state diagram, Fig. 4(a). However, eviction happens due to cache replacement policy or as a result of coherence activities which needs to be addressed by the NVC. Upon an eviction decision for a cache-block that belongs to PM, if the non-volatile segment of the cache-block does not contain a valid persistent copy, i.e BV is invalid, the eviction can happen immediately. However, if BV is set to valid and a persistent copy exists, the status of the transaction which the persistent copy is associated with decides the next steps. In case, the corresponding transaction has been committed, i.e Tid is invalid, the latest copy of the persistent of the data is in this cache-block and has to written back to PM. Thus, the data is first copied and persisted in persistent queue, where it stays until it is sent to PM and the L1 data cache receives the ack for its persistence in PM. This is crucial to ensure that persistent copies of data are not lost when transferring from persistent L1 data cache to PM. After, the data is persisted in the persistent queue, NVC persists the invalidation of BV via *Block-Backup*.

If Tid is not valid, meaning the cache-block is associated with an ongoing transaction, the non-volatile and volatile segments of the cache-block contains different version of the data. The non-volatile segment has the old copy of the data before any change in the transaction. On the other hand, the volatile segment has the latest value of the data that has been changed in the transaction, but not committed. As a result, NVC has to persist both of these copies for maintaining enough information to be able to roll back changes if the transaction aborts or discard the logs if the transaction commits. The modified data is persisted by overwriting the associated memory location in PM and the old version of data will be persisted in a specified log region in PM. We detail the logging operation and content of the log in Sec. IV-C. Note that the value in the non-volatile segment of the cache-block is retrieved using *Block-Restore*. At the end, invalidation of BV is persisted through *Block-Backup*.

Example of a transaction: Fig. 5 shows an example code in which persistent variables *saving* and *invest* are modified in a critical section with step-by-step illustration of operations that are executed in NVC's L1 data cache to guarantee persistency.

In the first scenario, *saving* is 40 and *invest* is 60. When L1 data cache receives the write request to add 20 to *saving*



Operations	Step-1		Step-2	
	BkpL(V)	RsL(V)	BkpL(V)	RsL(V)
Block-Backup (BB)	Vdd	gnd	gnd	gnd
Cache-Commit (CC)	Vdd	gnd	gnd	gnd
Block-Restore (BR)	Std_by	Vdd		

Fig. 4. (a)L1 data cache-block state machine. The state machine generates 3 outputs, namely Block-Backup, Block-Restore, Cache-Commit, and its corresponding activation signals (BkpL & RsL) to the M3D NVSRAM bit-cell with steps are shown in the table (b).

(Line 2), first *Block-Backup* is issued to backup the old value of *saving*, I(b). Next, L1 data cache services the write request and V segment of the corresponding contains the new value of *saving*, I(c). In this case, since the if statement in Line 3 holds true, *invest* is going to be updated. When L1 data cache receives the corresponding request for update to *invest*, similar to update *saving* (Line 4), it first issues *Block-Backup*, I(d) and then executes the request, I(e). When detecting the end of critical atomic region (Line 6), core informs the L1 data cache to commit the transaction. Consequently, a *Cache-Commit* operation is issued and all blocks modified in the transaction, cache-blocks associated with *saving* and *invest*, are backed up atomically.

The initial value of *saving* is equal to 10 in the second scenario. As a result, the condition in Line 3 is not satisfied and the second update is also to variable *saving*. Since we have already backed up the old value of cache-block associated with *saving* in this transaction and BV is set, L1 data cache execute the write request without issuing *Block-Backup* again, II(d). The *Cache-Commit* operation at the end commits all modifications in the transaction, in this case only value of block corresponding to *saving*.

C. Logging

Although NVC relies on M3D NVSRAM to achieve two versions of a persistent cache-block efficiently, if a modified cache-block that resides in PM has to be evicted from L1 data cache before committing the corresponding transaction, NVC uses logging to guarantee crash consistency. Note that, the cache block when evicted at L1 goes through the PQ (which is also persistent) and then to the PM. So, upon any failure the PQ is also checked for any logs.

NVC determines a log region in the PM and keeps track of the next available log entry memory address in the region to allocate in case of a log creation. The log is created on the flight and consists: (i) the data, (ii) the block address, (iii) Tid and the current value of TCNT counter for the corresponding thread ($TCNT_{Log}$) (iv) the order of log issued in this transaction, i.e., a volatile counter associated with each transaction that shows

```
// example code

1 account.lock();
2  saving += 20;
3  if(saving > 50)
4    invest += 10;
5  else saving += 10;
6 account.unlock();
```

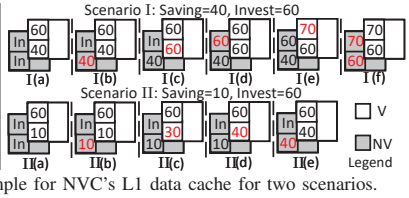


Fig. 5. Running example for NVC's L1 data cache for two scenarios.

TABLE I
SIMULATION CONFIGURATION (22NM)

Core	4-cores, 2GHz OoO 8-wide Dispatch, Commit, 192-entry ROB, 32/32-entry LSQ	D-Cache	64KB, 4-way, 64B 2ns hit lat, 8 MSHRs 3/1.5ns backup/restore
I-Cache	32KB, 4-way, 64B 1ns hit lat, 8 MSHRs	Persistent Queue	16 entries, 20ns delay to PM controller
L2	256KB unified, 8-way, 4.5ns hit lat, 16 MSHRs	LLC	2MB per core, 16-way, 12ns hit lat, 32 MSHRs
Memory controller	128/64-entry RWQ for both DRAM and PCM	DRAM; PCM	DDR4-2400; Rd/Wr lat 346/500ns

which log entry is older in that transaction. NVC resets this counter when the transaction commits.

D. Recovery

NVC uses the content of L1 data cache and TCNT counters, alongside any valid log entries that are present in the log region to guarantee that the system will be recoverable to a consistent state. The recovery mechanism will restore cache blocks and consider the ones with valid BV. Also, the valid log entries needs to be recovered. So, the recovery mechanism considers persistent queue as an extension of L1 data cache and restores valid entries upon recovery.

For identifying if a log entry is valid or not, the recovery mechanism uses the Tid field of the log entry to check the TCNT counter associated with the log. Next, recovery mechanism compares the $TCNT_{Log}$ field of the log with the value of TCNT counter. If $TCNT_{Log}$ is smaller than TCNT counter, it indicates that log belongs to a transaction that is committed and thus, it should be discarded. In case, multiple log entries exist for the same data in the same transaction, recovery mechanism compares the order field of the logs and only restore the oldest.

It is possible that a valid copy of the data in L1 data cache and a valid log both exist in the system for the same data. This could happen when a cache-block that belongs to a transaction that has not been committed is evicted and returned to L1 data cache again in the same transaction, then a failure happens before committing the transaction. In such a scenario, recovery mechanism ignores the the copy of the data in the cache-block as the log entry contains the old value of data.

V. EVALUATION

A. Methodology

Circuit/Design analysis: We perform simulations of M3D NVSRAM circuit in hpsice employing 20nm PTM model [12] for the CMOS transistors and the FeFET model calibrated based on L-K equation in [13]. The M3D process degrades the layer-2 transistors performance [14] which was incorporated in the simulation model. The M3D via was modeled with a resistance of 100Ω and capacitance of 1.2fF. The described state machine implemented using SystemVerilog RTL needed for the L1 data cache block in Fig. 4 was synthesized using Synopsys Design Compiler and the cache related parameters were measured using CACTI [15] for 22nm technology. The area overhead of the state machines for the entire L1 data cache, shown in Fig.4, amounts to 5%. With the required additional metadata and persistent queue (PQ), NVC incurs 7% in total.
Architectural Simulation infrastructure: We evaluate NVC using the gem5 architectural simulator [16] in full-system mode, configured as Table I. We model a four-core system with the ARMv8 ISA with 8-entry persistent queue per L1 data cache and use the recent Intel's Optane memory characterization study [17] to model a PM device in our system.
Workloads: The workloads contain several PM-centric multi-threaded workloads from [18], like tree-based data structures (RB), database (TPCC), and a persistent key-value store benchmark, YCSB. Fig. 6(a) illustrates the distribution of persistent stores per transactions in all workloads, (ignoring transactions that contain no persistent update) showing that a wide range of transaction sizes are evaluated.
Designs evaluated: In this work, we evaluate four different designs. (i) **Software Logging (SW)** refers to state-of-the-art SFR-based undo logging [2]. (ii) **Kiln** refers to the state-of-the-art hardware solution with persistent LLC [3]. (iii) **NVC** refers to our proposed design with M3D NVSRAM L1 Data cache. (iv) **Ideal** refers to the system with non-volatile memory that provides no crash consistency guarantee. Ideal is added to our evaluation to show the upper-bound on what NVC can achieve.

B. Performance Comparison

Fig. 6(b) shows the performance comparison across different design configurations. The main takeaways are:
Both NVC and Kiln outperform SW significantly in all workloads: The SW encounters high performance overhead as it has to ensure the undo-log containing the old version of the data is persisted in PM before any modification to it. This is in contrast to Kiln and NVC which have persistent caches do not need to update in PM (long critical path) to guarantee persistency. Kiln and NVC can achieve up to 4.6x and 5.1x in CQ and 2.8x and 3.0x on average speedup over SW. Note that although NVC persists data that is evicted from L1 by making sure it reaches PM, this event is not on the critical path of the execution and has almost no performance overhead.
NVC outperforms Kiln consistently: M3D NVSRAM L1 data cache in NVC does not introduce any overhead for volatile updates and provides efficient persist operations in L1 data cache. Moreover, the commit operation in Kiln relies on all the modified block in the transaction to be flushed to LLC. In comparison, the commit operation in NVC can happen atomically in L1 data cache. NVC gains up to 11% in CQ and 5% on average over Kiln across workloads. The gains seem modest attributing to the bypassing of the higher level caches and queuing structures (in Kiln) for these requests, thereby not impacting the critical path.
NVC achieves close to Ideal performance: NVC can reach close-to-Ideal performance as the atomic operations of M3D NVSRAM L1 data cache provide efficient persist and commit operations. We observed minimal 1% performance improvement in Ideal over NVC on average. This also implies that the performance implications of NVC are negligible.

C. Cache Energy Overhead

The read, write, backup and restore energies per block in M3D NVSRAM are 0.0344nJ, 0.0368nJ, 0.0010nJ, 0.0006nJ respectively. The backup and restore energies are low compared to the memory operations mainly attributing to the avoidance

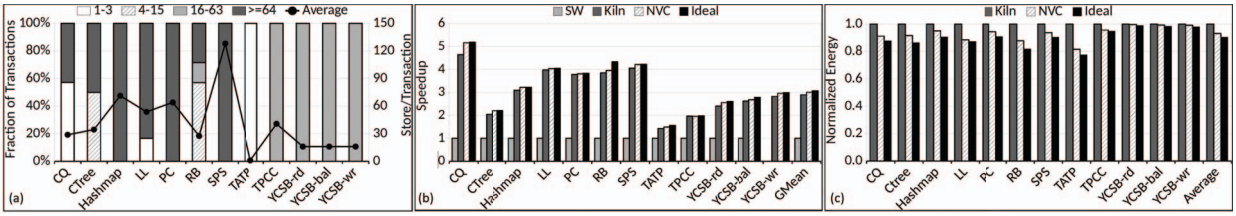


Fig. 6. (a)The distribution of persistent updates in a transaction across workloads. TATP contains transactions with only one persistent update inside, on the other hand, the number of updates in transactions of Hashmap can be over hundreds. (b)Normalized speedup over SW for all design configurations. (c)Normalized cache hierarchy energy consumption over Kiln.

of the highly parasitic H-tree buses. The memory operations (read/write) uses both the data and address buses. Whereas the backup and restore operation uses only the address bus and also avoids the costly periphery logics, thereby consuming less energy. Also, these operations uses the local ground path in each cell thereby avoiding any parasitics related costs.

We study the cache hierarchy energy consumption in Kiln, NVC and Ideal, assuming energy characterization of 8MB STT-RAM LLC for Kiln shown in Fig. 6(c). NVC consistently reduces the energy consumption in the cache hierarchy. The main benefits over Kiln is mainly because of the large 8MB L3 STT-RAM cache employed in Kiln, accounting for the expensive read/write costs (due to large bus parasitics) needed for the persistent requests , whereas NVC requires minimal access costs of small L1 data cache. Further, Kiln affects the performance and energy consumption of both volatile and non-volatile updates in LLC as STT-RAM lacks the dual functionality in which M3D NVSRAM offers. This is opposed to NVC which avoids issuing non-volatile writes for volatile updates, thereby reducing the expensive non-volatile write costs. NVC reduces the cache hierarchy energy up to 12.2% in RB and 7% on average.

D. Transaction Size Sensitivity Analysis

We modified plausible benchmarks to be able to vary the transaction size artificially as a parameter. The performance of NVC w.r.t SW is shown in Fig. 7, the speedup over SW will increase with higher transaction size. This is because the overhead of persisting and enforcing persistent orders increases with higher number of persistent updates in a transaction. On the other hand, NVC avoids such overheads by providing a persistent L1 data cache.

VI. Conclusion

In this work, we proposed NVC, an architecture design that employs M3D NVSRAM technology to move the point of persistence all the way to L1 data cache and eliminates almost all overheads with regards to guaranteeing crash consistency. We improved the performance of the state-of-the-art SW undo logging up to 5.1x and almost completely close the gap with the system with no persistency guarantee while incurring as hardware overhead only ≈7% increase in L1 data cache area.

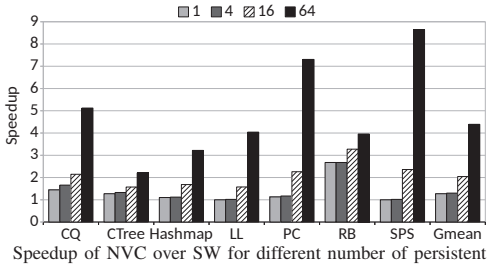


Fig. 7. Speedup of NVC over SW for different number of persistent updates in a transaction.

References

[1] J. Coburn et al., "NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories," SIGARCH Comput. Archit. News 39, 1, 105-118, 2011.
[2] V. Gogte et al., "Persistency for synchronization-free regions," 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, 46–61, 2018.
[3] J. Zhao et al., "Kiln: Closing the Performance Gap Between Systems With and Without Persistence Support," in ACM MICRO-46, 2013.
[4] V. Gogte et al., "Relaxed Persist Ordering Using Strand Persistency," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 652-665, 2020.
[5] V. Chidambaram, T. S. Pillai, A. C. Arpaci-Dusseau, andR. H. Arpaci-Dusseau, "Optimistic crash consistency,"in Proceedings of the 24th ACM Symposium on Operating Systems Principles. USA, p. 228–243, 2013.
[6] "https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html", 2016.
[7] M. M. Shulaker et al., "Monolithic 3D integration: A path from concept to reality," 2015 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1197-1202, 2015.
[8] A. I. Khan, C. W. Yeung, C. Hu, and S. Salahuddin, "Ferroelectric negative capacitance MOSFET: Capacitance tuning antiferroelectric operation," in IEDM Tech. Dig., USA, pp. 11.3.1–11.3.4, 2011.
[9] K. Berggren et al., "Roadmap on emerging hardware and technology for machine learning," Nanotechnology, 2021.
[10] S. Dutta et al., "Monolithic 3D Integration of High Endurance Multi-Bit Ferroelectric FET for Accelerating Compute-In-Memory," 2020 IEEE International Electron Devices Meeting (IEDM), pp. 36.4.1-36.4.4, 2020.
[11] X. Li et al., "Design of Nonvolatile SRAM with Ferroelectric FETs for Energy-Efficient Backup and Restore," IEEE Transactions on Electron Devices, 3037-3040, 2017.
[12] http://ptm.asu.edu/
[13] A. Aziz, S. Ghosh, S. Datta and S. K. Gupta, "Physics-Based Circuit-Compatible SPICE Model for Ferroelectric Transistors," in IEEE Electron Device Letters, vol. 37, no. 6, pp. 805-808, 2016.
[14] F. -K. Hsueh et al., "Monolithic 3D SRAM-CIM Macro Fabricated with BEOL Gate-All-Around MOSFETs," 2019 IEEE International Electron Devices Meeting (IEDM), pp. 3.3.1-3.3.4, 2019.
[15] R. Balasubramonian, A.B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas. "CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories," ACM Trans. Archit. Code Optim. 14, 2, Article 14, 2017.
[16] N. Binkert et al., "The gem5 simulator," SIGARCH Comput. Archit. News 39, 1-7, 2011.
[17] J. Izraelevitz et al., "Basic performance measurements of the intel optane DC persistent memory module," arXiv preprint arXiv:1903.05714, 2019.
[18] A. Kolli et al., "Language-level persistency," SIGARCH Comput. Archit. News 45, 481–493, 2017.