University of Kaiserslautern
Department of Computer Science
Database and Information Systems

Seminar
Recent Trends in Database Research

Summer Semester 2013

Inhaltverzeichnis

Die Verwendung von Phase-change Memory in modernen DBMS-Architekturen

Jan Philipp Stärz

University of Kaiserslautern

Abstract. Die folgende wissenschaftliche Arbeit behandelt Phase-change Memory (PCM) und dessen möglichen Einsatz in modernen DBMS-Architekturen.

Der erste Teil dieser Arbeit beschäftigt sich grundlegend mit PCM. Dabei wird zu Anfang dessen Funktionsweise erläutert sowie eine kurze Einordnung in bekannte Speicher gegeben. Im weiteren Verlauf werden die Vor- und Nachteile von PCM und denkbare Speicherarchitekturen vorgestellt. Darauf aufbauend folgen Strategien, die einen geringeren Verschleiß von PCM-Modulen erzielen.

Im zweiten Teil wird das DBMS-Logging-Schema PCMLogging vorgestellt, welches eine mögliche Verwendung von PCM aufzeigt, das Transaktions-Logging und die Recovery-Performance zu optimieren. Dabei werden die benötigten Datenstrukturen sowie das Vorgehen auf Seiten- und Satzebene behandelt. Auf die Perfomance-Aspekte von PCMLogging wird ebenfalls näher eingegangen.

Abschließend werden ein Fazit gezogen und ein Blick in die Zukunft von PCM und dessen Bedeutung für kommende Datenbanksysteme gegeben.

1 Einleitung

Um Mehrkernprozessoren mit Multi-Threading-Technologie voll ausschöpfen und große Arbeitsaufträge optimal parallelisieren zu können, benötigen wir zunehmend größeren Arbeitsspeicher. Durch die beschränkte Skalierbarkeit von DRAM ist man daher gezwungen, eine Alternative zu finden, die DRAM in ihrem Laufzeitverhalten ähnelt und skalierbar ist.
Phase-change Memory scheint hierfür ein möglicher Kandidat zu sein. Auch wenn er einige Vorteile, wie geringe Zugriffslatenzen, Skalierbarkeit und persistente Speicherung vereint, überwiegt dennoch der Nachteil der niedrigen Lebensdauer einer PCM-Zelle. Um PCM also praktikabel nutzen zu können, gilt es den Verschleiß so stark einzudämmen, dass er sich mit heutzutage gebräuchlichen Speichern messen kann. Dies ist das Ziel vom Data-Comparison Write (DCW) Scheme [3], den Seitenersetzungsstrategien least wear-unit difference (LWD), least page wear (LPW) und least frequently modified (LFM) [2] und dem probabilistischen Record-Swapping-Algorithmus [1].

In einer Zeit, in der immer mehr und größere Datenmengen verwaltet werden müssen, sind auch Datenbankforscher auf die attraktiven Vorteile von PCM aufmerksam geworden, die sich von neuartigen Datenbanksystemen mit integriertem PCM Performance-Gewinne erhoffen. So sollen beispielsweise Anfragen schneller abgewickelt oder der Recovery-Prozess beschleunigt werden, um zum einen die Mehrbenutzbarkeit zu steigern und zum anderen den Schaden durch Systemausfälle zu minimieren. Eine Möglichkeit stellt das PCMLogging dar, welches PCM zur DRAM-Unterstützung zwischen Prozessor und externem Speicher einsetzt, um so die Nicht-Volatilität und die schnelle Zugriffsgeschwindigkeit des PCM auszunutzen [1].

2 Was ist Phase-change Memory (PCM)?

Phase-Change Memory (PCM) ist ein neuartiger Speicher, der anstelle eines Kondensators – wie bei herkömmlichen DRAM – ein Phasenwechselmaterial zur Datenspeicherung besitzt. Als Phasenwechselmaterial wird eine Germanium-Antimon-Tellur-Verbindung ($Ge_2Sb_2Te_5$, kurz GST) verwendet, die je nach Beschaffenheit einen Datenwert speichert. Liegt das GST kristallin vor, so befindet sich die PCM-Zelle im SET-Zustand. In diesem Zustand weist das GST einen niedrigen elektrischen Widerstand auf, woraus eine logisch gespeicherte '1' folgt. Ist das GST hingegen im amorphen RESET-Zustand, besitzt es einen hohen elektrischen Widerstand, woraus eine logische '0' gelesen werden kann. Um eine PCM-Zelle in den Zustand SET zu versetzen, erhält die Zelle über einen längeren Zeitraum moderate elektrische Impulse. Konträr dazu ist der RESET-Zustand. Dieser wird herbeigeführt, indem man das Phasenwechselmaterial mit kurzen, hoch energetischen Impulsen wieder amorph schmilzt. Das Lesen der gespeicherten Daten geschieht durch das Messen des elektrischen Widerstands des Materials und benötigt somit nur wenig Energie [5]. Die wesentlichen Unterschiede der drei Operationen SET, RESET und READ werden in Abbildung 1 verdeutlicht.
Dadurch, dass zur Aufrechterhaltung des jeweiligen Zustands (kristallin oder amorph) kein elektrischer Strom benötigt wird, ist PCM ein nicht-volatiler Speicher. Außerdem ist PCM byte-adressierbar und ermöglicht die Änderung einzelner Bits, indem die jeweilige PCM-Zelle gesondert angesprochen wird [1]. Die Lebensdauer einer PCM-Zelle ist jedoch beschränkt, da jede auf sie angewendete Operation unweigerlich zum Zerfall des Phasenwechselmaterials führt [5].

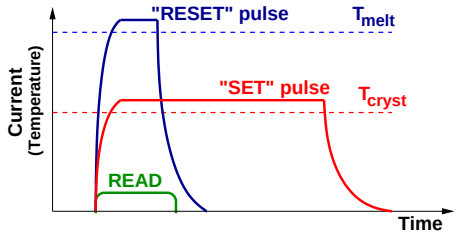


Abbildung 1. Energieaufwand für die PCM-Operationen SET, RESET und READ. Für GST ist $T_{melt} \approx 610°C$ und $T_{cryst} \approx 350°C$ [5]

3 Einordnung in bekannte Speicher

Ein Vergleich von PCM mit heute gebräuchlichen Speichern verdeutlicht das Potential dieser Technologie.

PCM erreicht eine zwei- bis vierfach höhere Dichte als DRAM und hat damit gegebenenfalls einen gleich hohen Wert wie NAND Flash. Eine PCM-Zelle kann ursprünglich nur ein Bit speichern, jedoch existiert bereits ein Prototyp, der zwei Bits pro Zelle (4 logische Zustände) verkörpert. Dieser Prototyp arbeitet mit zwei Zwischenphasen des Phasenwechselmaterials, deren Widerstände zwischen den beiden Extremwerten liegen [5].
Die Leselatenz von PCM liegt mit ca. 50ns sehr nahe an der von DRAM und ist damit deutlich besser als die von NAND Flash. Erreicht wird dieser Wert durch

	DRAM	NAND Flash	HDD	PCM
Dichte	1X	4X	N/A	2-4X
Leselatenz	20-50*ns*	~25μs	~5*ms*	~50*ns*
(Seitengröße)	(64B)	(4KB)	(512B)	(64B)
Schreiblatenz	20-50*ns*	~500μs	~5*ms*	~1μs
(Seitengröße)	(64B)	(4KB)	(512B)	(64B)
Idle-Verbrauch	~100 mW/GB	1-10 mW/GB	~10 W/TB	~1 mW/GB
Read-Verbrauch	0.8 J/GB	1.5 J/GB	65 J/GB	1 J/GB
Write-Verbrauch	1.2 J/GB	17.5 J/GB	65 J/GB	6 J/GB
Beständigkeit (in Schreibzyklen)	N/A	10^4-10^5	∞	10^6-10^8

Abbildung 2. Vergleich heutiger Speichertechnologien [1][5]

die kurze, energiearme Ansteuerung der PCM-Zellen, um deren Widerstand zu ermitteln. So ist es auch nicht verwunderlich, dass der Energieverbrauch einer PCM-Read-Operation nur 0.2 J/GB höher ist.
Eine im Vergleich zu DRAM hohe und zu NAND Flash niedrige Schreiblatenz lässt sich durch den Zeitaufwand der SET-Operation erklären. PCM ist byte-adressierbar und muss somit nicht wie Flashspeicher den gesamten Block zurücksetzen, in den Daten geschrieben werden sollen [3]. Daraus resultiert auch der merkliche Unterschied zwischen 6 und 17.5 J/GB für eine Write-Operation. Die Energiedifferenz zwischen DRAM und PCM von 4.8 J/GB liegt in der Umsetzung der Write-Operation einer PCM-Zelle. Ihr Verbrauch variiert im Falle des Data-Comparison Write (DCW) Scheme (siehe Kapitel 5.1) zwischen viel zugeführter Energie in kurzer Zeit (RESET-Operation), mäßiger Energiezufuhr über längere Zeit (SET-Operation) und sehr wenig Energieverbrauch (READ-Operation).
Der Verbrauch im Idle-Zustand von ca. 1 mW/GB wird mithilfe des Phasenwechselmaterials erreicht, welches persistentes Schreiben ermöglicht. Damit liegt PCM weit unter dem Idle-Energieverbauch von DRAM und kann sich mit dem Verbrauchsbestwert von NAND-Flashspeichern messen.
Die Lebensdauer des Phasenwechselmaterials ist bisher ein großer Schwachpunkt der PCM-Technologie. Da sich gegenwärtig vielversprechende Ansätze zur Verschleißreduzierung finden, wird die Beständigkeit zukünftiger PCM-Module auf $10^6 - 10^8$ Schreibzyklen geschätzt.

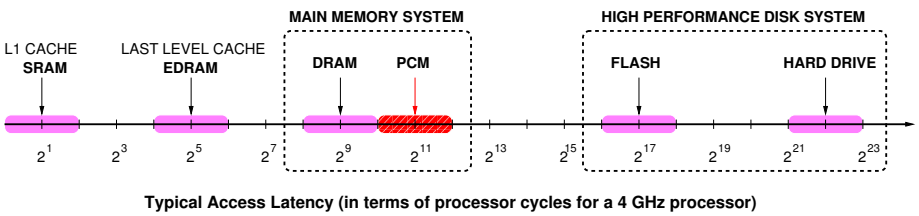


Abbildung 3. Einordnung von PCM in bekannte Speicher anhand der Zugriffslatenz in Prozessorzyklen [4]

4 Vor- und Nachteile von PCM, vorstellbare Architekturen

Bevor auf vorstellbare, PCM-beinhaltende Speicherhierarchien eingegangen wird, werden im Folgenden noch einmal explizit alle Vor- und Nachteile des PCM zusammengefasst.

Vorteile:

- Geringer Stromverbrauch
- Persistente Speicherung
- Byte-Adressierung
- Bit-Modifikation
- Skalierbarkeit

Nachteile:

- Hoher Verschleiß (derzeitiger Stand)

Abbildung 3 verdeutlicht, dass PCM nicht weit von den Eigenschaften des DRAM entfernt ist. Damit wird besonders seine Verwendung als Arbeitsspeicher interessant. Es sind daher drei Speicherarchitekturen denkbar (Abbildung 4).

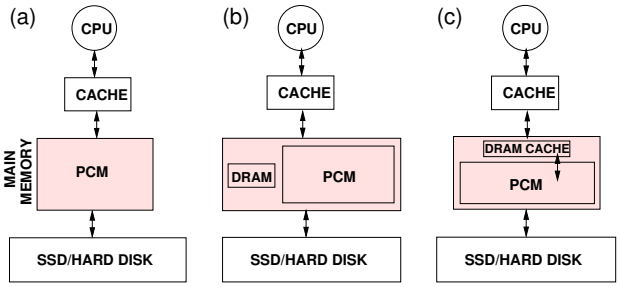


Abbildung 4. Denkbare Speicherhierarchien mit integriertem PCM [5]

Abb. 4 (a) zeigt eine vollkommene Ersetzung des volatilen DRAM durch persistenten PCM. Obwohl PCM höhere Zugriffszeiten hat, wird so weiter skalierbarer Arbeitsspeicher gewonnen. In Anbetracht der geringen Lebensdauer derzeitiger PCM-Module ist Architektur (a) jedoch noch nicht für den Langzeitbetrieb geeignet. Die Speicherhierarchien (b) und (c) enthalten beide neben dem PCM ein DRAM-Modul, um Daten zu puffern, auf die häufig zugegriffen wird. Das Ziel beider Hybriden ist es, dadurch die Performance des Arbeitsspeichers zu erhöhen und gleichzeitig den PCM-Verschleiß zu verringern [4]. Während bei (b) DRAM und PCM an der I/O-Schnittstelle liegen, um Software die direkte Kontrolle über den DRAM-Puffer zu ermöglichen, stellt der DRAM-Puffer in (c) lediglich eine weitere datendurchreichende Cache-Schicht dar, die den PCM entlasten soll [1][4]. Der größere Nutzen von Architektur (b) wird in Kapitel 6 anhand des PCMLogging verdeutlicht.

5 Lindern der Nachteile

5.1 Data-Comparison Write Scheme

Viele PCM-thematisierende Arbeiten stellen Strategien vor, die entweder die Anzahl der Schreibzugriffe auf PCM reduzieren oder das wiederholte Schreiben gleichmäßig

über alle Zellen verteilen sollen [2]. Das Data-Comparison Write (DCW) Scheme hat ersteres zum Ziel.
Das konventionelle Schreibschema von PCM schreibt direkt in eine Zelle, unabhängig davon, welche Daten sich zuvor in der Zelle befanden (Abbildung 5 (a)). Das DCW Scheme soll genau dies verhindern, indem es die Notwendigkeit für das Schreiben prüft. Bevor es zur Entscheidung kommt, ob in die PCM-Zelle geschrieben wird, wird der aktuelle Wert der Zelle gelesen. Im Anschluss liegen der aktuelle Wert (Read Data) und der zu schreibende Wert (Write Data) vor. Diese werden miteinander verglichen. Sind Read und Write Data gleich, so ist kein Schreiben nötig, andernfalls wird nach dem konventionellen Schreibschema verfahren (Abbildung 5 (b)). Dadurch halbiert das DCW Scheme die Anzahl der tatsächlichen Schreibzugriffe auf eine PCM-Zelle, wenn alle Bit-Modifikationen gleichwahrscheinlich sind. Lediglich der Verbrauch einer Schreiboperation ist durch die vorher ausgeführte Leseoperation höher. Dieser Mehraufwand ist jedoch durch die geringen Energiekosten eines READ vernachlässigbar [3]. Eine ausführliche Implementierung eines DCW-PCM-Moduls sowie die Testergebnisse eines ersten Prototyps finden sich in [3].

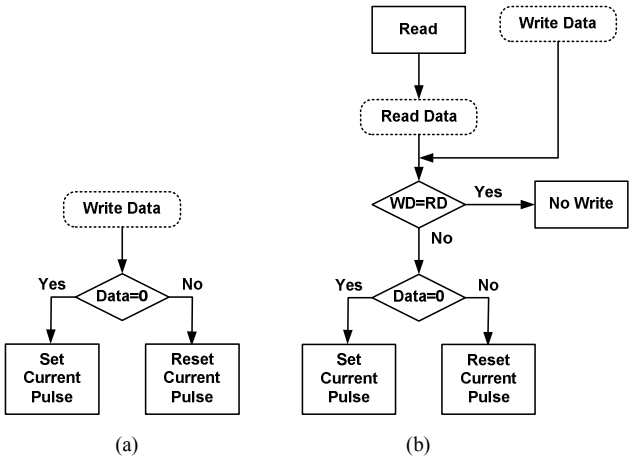


Abbildung 5. Ablaufdiagramme des (a) konventionellen PCM-Schreibschemas und des (b) Data-Comparison Write Scheme [3]

5.2 Seitenersetzungsstrategien

Der Algorithmus least wear-unit difference (LWD) ist ein Seitenersetzungsalgorithmus, der seine Ersetzungsentscheidung anhand einer kleinsten Wear-Unit-Differenz (WD) trifft. Wird eine angeforderte Seite p nicht im PCM gefunden (page miss), so vergleicht der LWD-Algorithmus p mit jeder Seite q im Speicher und verdrängt diejenige, die die kleinste Wear-Unit-Differenz zu p aufweist. Haben mehrere Seiten im Speicher die kleinste WD zu p, wird nach least recently used (LRU) verfahren [2].
Die WD zweier Seiten wird ermittelt, indem jedes korrespondierende Paar von PCM-Einheiten (wear units), z.B. durch gleichen Adressabstand innerhalb ihrer Seite, verglichen wird. Voneinander verschiedene Paare werden gezählt. Kurz gesagt, beschreibt die WD somit die physikalische Ähnlichkeit der zwei betrachteten Seiten. LWD wählt demnach die Seite als Verdrängungskandidaten, die der angeforderten

physikalisch am nächsten kommt, wodurch unter Verwendung des DCW Scheme weniger Bit-Modifikationen beim Schreiben benötigt werden. Bei einer Seitengröße von P und einer Speicherkapazität von B werden B * P viele Vergleiche zum Ermitteln aller WD benötigt, woraus für LWD eine Zeitkomplexität von $\mathcal{O}(B * P)$ folgt [2].
Was LWD jedoch nicht abdeckt, ist die Anzahl der bisher getätigten Überschreibungen pro Seite (wear status). Er beeinflusst also nicht die Verteilung der Schreibzugriffe, sodass es vorkommen kann, dass eine Seite öfter ersetzt wird als andere. Des Weiteren ist die Wahrscheinlichkeit für zwei physikalisch identische Seiten mit $1/(2^{32})$ bei einer Wear-Unit-Größe von 4 Byte sehr gering, wodurch LWD nicht den seitenübergreifenden Verschleiß reduziert [2].

Der Algorithmus least page wear (LPW) verfolgt einen anderen Ansatz als LWD, indem er mit einem wear status (siehe 4.2.1) arbeitet, anstatt eine Wear-Unit-Differenz zu ermitteln. Wird eine Seite aus dem PCM verdrängt, so wird ihr Verschleißzähler (wear counter) um die Anzahl der zu ändernden PCM-Zellen (wear units) erhöht. LPW wählt dann die Seite zur Verdrängung aus, die den geringsten Verschleißzähler besitzt. Der Verschleißzähler kann jedoch nur den wear status approximieren, da er nicht die Verschleißverteilung innerhalb einer Seite repräsentiert. So kann es sein, dass eine Seite verdrängt wird, die wenige, oft modifizierte wear units beinhaltet, trotzdem aber einen niedrigen Verschleißzähler aufweist. Da zur Entscheidung, welche Seite ersetzt werden soll, die mit dem niedrigsten Verschleißzähler im Speicher gefunden werden muss, besitzt LPW eine Zeitkomplexität von $\mathcal{O}(B)$. Hierbei ist B die Größe des Speichers [2].

Der Algorithmus least frequently modified arbeitet nach einem ähnlichen Prinzip wie der klassische Algorithmus least frequently used (LFU). Um den wear status abzubilden, verwendet LFM pro Seite einen Modifikationszähler, der inkrementiert wird, wenn die Seite aus dem Speicher verdrängt wird. Der Modifikationszähler bleibt dafür nach der Verdrängung erhalten und wird nicht wie bei LRU zum Zeitpunkt der Auslagerung zurückgesetzt. LFM wählt demnach die Seite im Speicher aus, die den niedrigsten Modifikationszähler besitzt und damit bisher am seltensten ersetzt, bzw. überschrieben wurde. Um diese Seite zu ermitteln, muss der LFM-Algorithmus über den gesamten Speicher der Größe B gehen. Daraus folgt eine Zeitkomplexität von $\mathcal{O}(B)$ [2].

Nach dem Beispiel des LFM ist auch ein Algorithmus least recently modified (LRM), basierend auf dem klassischen Algorithmus least recently used (LRU), möglich, der die am längsten nicht modifizierte Seite verdrängt [2].

Testergebnisse zeigen, dass die vier vorgestellten Seitenersetzungsstrategien effektiv die Lebensdauer eines PCM-Moduls erhöhen können, obwohl sie nur die Verschleißverteilung auf Seitenebene beeinflussen, nicht aber die Verteilung innerhalb der Seiten. Für detailliertere Informationen sei auf die Erstveröffentlichung der Algorithmen (siehe [2]) verwiesen.

5.3 Probabilistischer Record-Swapping-Algorithmus

Während das in Kapitel 5.1 vorgestellte Data-Comparison Write (DCW) Scheme die Anzahl der Schreibzugriffe auf PCM reduziert, geht der probabilistische Record-Swapping-Algorithmus den Weg der Schreiblastverteilung über das gesamte PCM-Modul. Damit ist sein Ziel die Vermeidung von hochfrequentiertem Schreiben in

immer dieselbe PCM-Stelle. Anwendung findet er beispielsweise in Datenbanksystemen [1].

Soll ein Schreibzugriff auf PCM erfolgen, wird geprüft, ob der Speicherinhalt, auf den ein im DRAM gespeicherte Swap-Pointer zeigt, im Vergleich zum zu schreibenden Datensatz kalt ist, also länger nicht mehr modifiziert wurde (cold record). Ist dies der Fall, wird der zu schreibende Datensatz an die Stelle des cold record geschrieben, nachdem dieser an eine freie Speicheradresse verschoben wurde (swap). Im Anschluss wird der Swap-Pointer auf die nächste PCM-Adresse gesetzt. Als Swap-Kriterium verwendet der Algorithmus die aktuelle Transaktionsnummer (XID) X_c und die XID X_t des Datensatzes t, der vom Swap-Pointer referenziert wird. Ist δ die systemweite Swap-Schwelle, so wird der zu schreibende Datensatz und t vertauscht, wenn $X_c - X_t \geq \delta$ gilt. Da die XID eine monoton steigende Zahl ist, stellt die Differenz ein gutes Maß für die Zeit der letzten Modifikation von t dar. Für den Fall $X_c - X_t < \delta$ ist eine fest gewählte Swap-Schwelle nicht optimal, da sie potentiell zu einer längeren Wartezeit führen kann, bis ein Swap-Kandidat gefunden wird. Ist δ zu groß gewählt, kommt es zu geringerer Schreibverteilung bei ebenfalls geringem Schreibmehraufwand durch den Algorithmus, wohingegen ein zu kleines δ zu hohen Schreibmehraufwand mit sich bringt. Daher wird eine Wahrscheinlichkeit von $\frac{X_c - X_t}{\delta}$ vorgeschlagen, die Datensätze trotzdem zu vertauschen [1].
Der probabilistische Record-Swapping-Algorithmus vermeidet demnach, dass ein cold record eine Speicheradresse über eine längere Zeit für sich beansprucht, indem er die Speicherplätze von cold records durch eintreffende, potentiell heiße Datensätze (hot records) ersetzt. Potentielle Hotspots erhalten so kalte Adressen. Trotz des Schreibmehraufwandes auf dem PCM, zeigen erste Testergebnisse eine Lebensdauerverlängerung um das 5-fache bei einer um 34% erhöhten Anzahl an Schreibzugriffen [1].

6 PCMLogging

In diesem Kapitel wird ein Logging-Schema vorgestellt, welches auf einer PCM-Architektur arbeitet, wie sie in Abbildung 4 (b) zu sehen ist. Die Grundidee des PCMLogging besteht darin, den Transaktionslog und die modifizierten Daten zu kombinieren und im PCM zu speichern, um so die Persistenz vom PCM geschickt zu nutzen. In Kapitel 6.2 wird PCMLogging auf Seitenebene vorgestellt, um das Vorgehen leichter verdeutlichen zu können. Um die Transaktionsverarbeitung von PCM-Logging weiter zu optimieren, wird in Kapitel 6.3 auf eine Erweiterung auf Satzebene eingegangen. Jegliches Wissen über PCMLogging in diesem Kapitel wurde aus [1] generiert.

6.1 Datenstrukturen und Grundlagen

Folgende Datenstrukturen werden für das PCMLogging im DRAM, bzw. PCM benötigt:

- Mapping Table: Die Mapping-Tabelle wird im DRAM gehalten und dient der Abbildung logischer Seiten-IDs zu physischen PCM-Adressen. Sie liegt nicht im PCM, da ihre Einträge häufig modifiziert werden und PCM langsamer ist als DRAM.
- Inverse Mapping: Das inverse Mapping ist Teil der Metadaten einer Seite (kurz PID) und wird für die Rekonstruktion der initialen Mapping Table beim Systemstart benötigt.
- FreeSlotBitmap: Die FreeSlotBitmap speichert, welche PCM-Speicherplätze noch nicht belegt sind. Sie ist im PCM abgelegt.

– ActiveTxList: Mithilfe dieser Liste werden sich alle Transaktionen gemerkt, die noch laufen und deren modifizierten Seiten (dirty pages) sich im PCM befinden. Sie wird für das Transaktions-Recovery benötigt und ist ebenfalls im PCM gespeichert.
– Transaction Table (TT): Die Transaktionstabelle beinhaltet alle noch laufenden Transaktionen. Für jede Transaktion wird dabei eine Liste aller modifizierten Seiten gespeichert, die sich im DRAM oder PCM befinden. Bei einem Commit oder Abort kann so schnell auf die durch die Transaktion geänderten Seiten zugegriffen werden.
– Dirty Page Table (DPT): Die Dirty Page Table hält Referenzen auf zuvor mit einem Commit abgeschlossene Versionen jeder Seite, die durch eine laufende Transaktion modifiziert wurde.

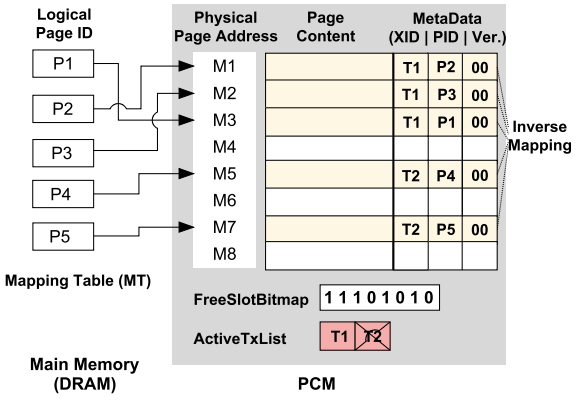


Abbildung 6. Seitenformat und Mapping Table [1]

Anzumerken ist, dass nur dirty pages im PCM gespeichert werden, die aus dem DRAM verdrängt wurden oder deren Änderungstransaktion mit einem Commit abschließt, um Dauerhaftigkeit der Daten zu gewährleisten. Des Weiteren arbeitet PCMLogging nach dem Shadow-Paging-Verfahren, indem eine Seitenmodifikation nicht auf der vorherigen Version einer Seite, sondern auf einer Kopie stattfindet. Für die Versionenübersicht ist die Dirty Page Table zuständig, die im Falle eines Systemabsturzes oder Transaktions-Rollback eine Undo-Operation ermöglicht. Die XID der letzten ändernden Transaktion sowie eine Versionsnummer sind ebenfalls Teil der Metadaten einer Seite [1].
Um Atomarität zu garantieren, wird eine Seite einer Transaktion erst in den PCM geschrieben, wenn die korrespondierende XID in die ActiveTxList aufgenommen wurde. Die XID wird erst aus der ActiveTxList entfernt, wenn die zugehörige Transaktion mit einem Commit abschließt und alle dirty pages aus dem DRAM in den PCM verdrängt wurden. Wird also während des Recovery-Prozesses eine Transaktion in der ActiveTxList gefunden, so war sie zum Zeitpunkt des Systemabsturzes noch nicht abgeschlossen. Beispielsweise würden die Änderungen von Transaktion 1 in Abbildung 6 verworfen werden, wenn wir annehmen, dass nur Seiten zum externen Speicher zurückgeschrieben werden, die von einer mit Commit abgeschlossenen Transaktion modifiziert wurden. Die Speicherplätze M5 und M7 bleiben unverändert, da Transaktion 2 rechtzeitig mit einem Commit abgeschlossen wurde. Die FreeSlotBitmap wird angepasst zu 00001010. Gegebenenfalls werden die Seiten in M5 und M7 auf ein externes Speichermedium zurückge-schrieben [1].

6.2 Vorgehen auf Seitenebene

In diesem Kapitel werden eine PCM-Granularität von genau einer Seite und ein Nebenläufigkeitsverhalten auf Seitenebene vorausgesetzt. Durch Seitensperren kann zu jeder Zeit nur höchstens eine Transaktion eine Seite modifizieren. Notwendige Operationen werden von PCMLogging wie folgt umgesetzt:

- **Verdrängung modifizierter Seiten zum PCM (DRAM-Puffer mit Steal-Politik).** Falls der DRAM voll wird oder eine Transaktion mit einem Commit abschließt, müssen modifizierte Seiten (dirty pages) verdrängt werden. Bevor eine ausgewählte Seite M in den PCM verschoben wird, muss die zugehörige Transaktions-XID in die ActiveTxList im PCM aufgenommen werden, falls sie nicht schon vorhanden ist. Besitzt M Vorgängerversionen, deren Änderungstransaktionen mit einem Commit abgeschlossen haben, erstellen wir eine Kopie M' von M mit erhöhter Versionsnummer und speichern M' an einen anderen Platz. M wird der Dirty Page Table mit Verweis auf M' hinzugefügt und der Eintrag der ändernden Transaktion in der Transaction Table wird aktualisiert. Ein Beispiel ist in Abbildung 7 (a) und (b) illustriert: T1 ist noch in Arbeit, während T2 bereits mit einem Commit abgeschlossen hat. T3 modifiziert die Seite P5, befindlich im DRAM (siehe Abb. 7 (a)). Anschließend wird P5 zu PCM-Speicherplatz M8 verdrängt. T3 wird der ActiveTxList hinzugefügt, P5 verweist in der Mapping Table ab sofort auf M8, in der Transaction Table zeigt T3 auf M8 und M7 wird als vorherige Version von M8 in der Dirty Page Table aufgenommen (siehe Abb. 7 (b)).
- **Commit.** Möchte eine Transaktion mit einem Commit abschließen, wird mithilfe der Transaction Table jede noch im DRAM befindliche dirty page in den PCM geschrieben. Anschließend wird der Commit gültig gemacht, indem die zur Transaktion korrespondierende XID aus der ActiveTxList gelöscht wird. Es folgt eine Anpassung der FreeSlotBitmap anhand der Dirty Page Table. Alle vorherigen Versionen der verdrängten Seiten werden verworfen, indem deren FreeSlotBitmap-Einträge auf 0 gesetzt werden. Abschließend werden die zur Transaktion zugehörigen Einträge in der Dirty Page Table und der Transaction Table gelöscht. Abbildung 7 (c) veranschaulicht das PCM-Logging-Verhalten bei einem Commit: T3 möchte mit einem Commit abschließen. Da alle von T3 modifizierten Seiten bereits im PCM liegen, wird T3 aus der ActiveTxList entfernt und die vorherige Version von P5 in M7 sowie deren Eintrag in der Dirty Page Table werden verworfen (siehe FreeSlotBitmap und DPT). Zum Schluss wird T3 aus der Transaction Table herausgestrichen.
- **Abort.** Wird eine Transaktion abgebrochen, werden alle zugehörigen dirty pages aus dem PCM verworfen (durch eine Änderung der FreeSlotBitmap). Falls eine dieser Seiten in der Dirty Page Table vorkommt, wird die vorherige, zuletzt committete Version wiederhergestellt, indem die Mapping Table angepasst wird. Abschließend werden die Transaktions-XID aus der ActiveTxList und zugehörige Einträge aus der Transaction und der Dirty Page Table entfernt. Abbildung 7 (d) zeigt einen Abbruch von T3. Demnach wird die aktuelle Version in M8 verworfen, der Verweis auf die letzte gültige Version in M7 in der Mapping Table wiederhergestellt und es werden korrespondierende Einträge in der ActiveTxList, der Dirty Page Table und der Transaction Table entfernt.
- **Recovery.** Nach einem Systemabsturz werden alle Seiten im PCM geprüft, ob sie die zuletzt gültige Version sind. Dabei werden alle Speicherplätze überprüft, die in der FreeSlotBitmap den Wert 1 haben. Jede Seite, die von einer vor dem Absturz noch laufenden Transaktion (siehe ActiveTxList) modifiziert wurde, wird verworfen, indem an der korrespondierenden Stelle in der FreeSlotBitmap eine 0 geschrieben wird. Aus den übrig bleibenden wird die Mapping Table rekonstruiert. Selbst wenn der Fall eintritt, dass der Absturz in Abbildung 7 (c)

erfolgt, nachdem T3 von der ActiveTxList entfernt und M7 noch nicht verworfen wurde, kann mithilfe der Versionsnummern der Seiten in M7 und M8 die letzte committete Seitenmodifikation identifiziert werden. Erwähnenswert ist, dass der gesamte Recovery-Vorgang keinen Zugriff auf einen externen Speicher benötigt [1].

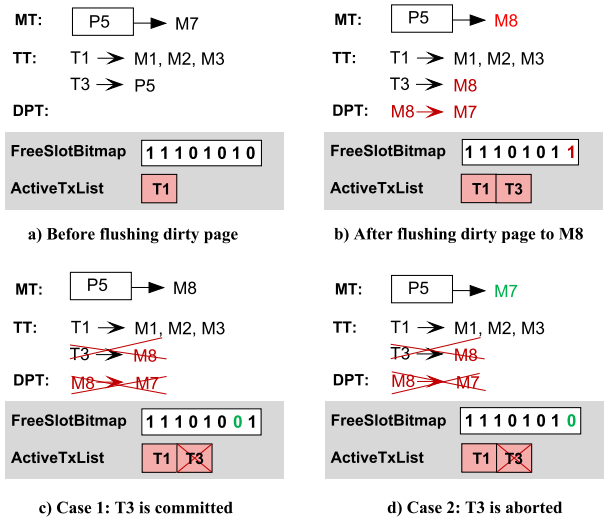


Abbildung 7. Ein Beispiel zur Verdeutlichung des PCMLogging (MT: Mapping Table, TT: Transaction Table, DPT: Dirty Page Table) [1]

6.3 Vorgehen auf Satzebene

In diesem Kapitel wird eine PCMLogging-Erweiterung vorgestellt, die nicht mehr auf Seiten-, sondern auf Satzebene arbeitet. Dazu werden anstelle von modifizierten Seiten nun modifizierte Sätze im PCM zwischengespeichert.
Ein PCMLogging-Schema auf Satzebene hat mehrere Vorteile. Zum einen wird weniger Speicherplatz auf dem PCM benötigt, da Sätze nur kleine Teile einer Seite sind. Zum anderen reduziert es die Anzahl der Schreibzugriffe auf den PCM, was eine Lebensdauerverlängerung des Moduls und weniger Verdrängungen zur Folge hat. Des Weiteren werden satzbasierte Nebenläufigkeitsverfahren unterstützt, welche bei modernen Datenbanksystemen ein hohes Maß an Nebenläufigkeit gewährleisten [1].
Die Datenstrukturen von PCMLogging werden wie folgt erweitert bzw. angepasst:

- Die Speicherplätze des PCM werden an die Satzgröße angepasst. Das Satzformat gleicht dem Seitenformat bis auf die Nutzdaten, die nun Sätze sind und einem zusätzlichen Metadatenfeld namens SNO. Die Slot-Nummer (SNO) und die Seiten-ID bilden die Satz-ID (RID). Zur Verwaltung freien Speichers kann ebenfalls eine FreeSlotBitmap verwendet werden.
- In der Mapping Table werden immer noch modifizierte Seiten verwaltet, jedoch speichert jede Seite alle Abbildungen auf modifizierte Sätze (dirty records), z.B. mithilfe eines binären Baums basierend auf der RID.
- Mit der Dirty Page Table wird ähnlich verfahren, sodass jede Seite in der Tabelle Verweise zu Vorgängerversionen der jeweiligen Sätze enthält.

- Zusätzlich wird eine Liste für jede Seite angelegt, die speichert, welche zugehörigen dirty records sich im DRAM befinden. Wenn eine Transaktion mit einem Commit abschließen will oder eine Seite aus dem DRAM verdrängt werden muss, so können zugehörige Sätze schnell identifiziert und in den PCM geschrieben werden.
- Die Transaction Table unterhält eine Liste aller laufenden Transaktionen und deren dirty records.
- Durch die Mapping Table kann ein schneller Zugriff auf einen Satz im PCM erfolgen, wenn er vorliegt. Wird eine ganze Seite angefordert, so wird diese vom externen Speicher geladen und mit den zuletzt gültigen, seitenrelevanten Sätzen, die im PCM zwischengespeichert sind, zusammengeführt. Da die Ladevorgänge parallel stattfinden, ist die Zugriffslatenz des PCM gegenüber der des externen Speichers vernachlässigbar.
- Damit der PCM nicht vollläuft, sollten während des Betriebes oder idealerweise in Systemleerlaufzeiten Sätze mit Commit abgeschlossener Transaktionen auf das externe Speichermedium zurückgeschrieben werden. Hierzu wird die zu den Sätzen korrespondierende Seite vom externen Speicher geladen, mit den im PCM befindlichen Sätzen zusammengeführt und wieder zurückgeschrieben.

Die Arbeitsweise der vier Operationen des PCMLogging ist auf Satzebene wie folgt realisiert:

- **Verdrängung modifizierter Sätze zum PCM.** Satz t, der zuletzt durch Transaktion T modifiziert wurde, soll verdrängt werden. Ist t der erste dirty record, der im PCM zwischengespeichert werden soll (siehe ActiveTxList), so wird T der ActiveTxList hinzugefügt. Anschließend wird t in einen freien Speicherplatz vom PCM geschrieben. Existiert eine frühere Version von t im PCM, deren Änderungstransaktion mit einem Commit abgeschlossen hat, so verweist t in der Dirty Page Table ab sofort auf diese. Ansonsten, falls bereits eine Kopie von t vorliegt, wird diese verworfen, indem das korrespondierenden Bits in der FreeSlotBitmap auf 0 gesetzt wird. Abschließend verweist T in der Transaction Table zusätzlich auf t und der Adresseintrag von t in der Mapping Table wird aktualisiert.
- **Commit.** Transaktion T will mit einem Commit abschließen. Zu Beginn wird jeder von T modifizierten Satz aus dem DRAM (zu finden in der Transaction Table) verdrängt. Anschließend wird T von der ActiveTxList im PCM entfernt und vorherige Versionen der zu T gehörigen dirty records (zu finden über die Dirty Page Table) werden verworfen, indem deren korrespondieren Bits in der FreeSlotBitmap auf 0 gesetzt werden. Abschließend werden die Einträge von T in der Transaction Table und der Dirty Page Table gelöscht.
- **Abort.** Transaktion T wird abgebrochen, erhält also ein Abort. Mithilfe der Transaction Table werden alle dirty records sowohl im DRAM, als auch im PCM gefunden und durch Änderungen an der FreeSlotBitmap verworfen. Wenn T noch in der ActiveTxList zu finden ist, werden mithilfe der Dirty Page Table die zuletzt gültigen Versionen der dirty records von T wiederhergestellt. Außerdem wird die Mapping Table auf die vorherigen Versionen aktualisiert und T wird von der ActiveTxList entfernt. Abschließend werden alle Einträge, die von T sind, in der Transaction Table und DirtyPageTable gelöscht.
- **Recovery.** Alle Sätze, die nach der FreeSlotBitmap valide sind (an der korrespondierenden Stelle ist eine 1 gespeichert), werden überprüft, ob deren XID in der ActiveTxList steht oder bereits eine höhere Version vorliegt. Erfüllt ein Satz t diese Bedingung, wird sie durch Änderung der FreeSlotBitmap verworfen. Ansonsten wird für t ein Eintrag in der Mapping Table im DRAM angelegt. Letzterer Fall kann beispielsweise dann auftreten, wenn ein Satz aus dem DRAM

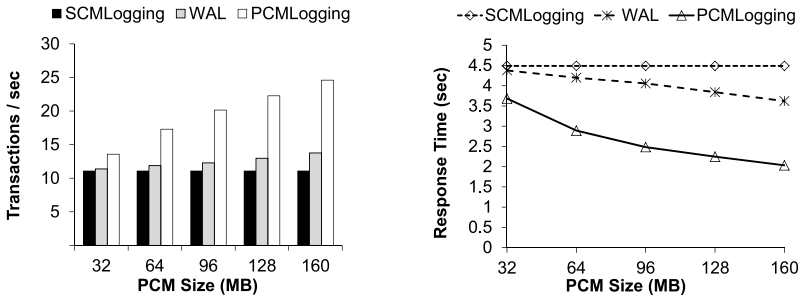
zum PCM verdrängt wird und es vor der Aktualisierung der Mapping Table zu einem Systemabsturz kommt.

6.4 Performance-Aspekte

In [1] wird die Verwendung PCMLogging umfangreich für Datenbanken simuliert und analysiert. In diesem Kapitel wird lediglich das Gesamtergebnis verschiedener Tests vorgestellt. Für ausführlichere Informationen, beispielweise zu der verwendeten Testumgebung, sei auf [1] verwiesen.
Verglichen wurden drei Logging-Schemata:

- SCMLogging: Das PCM-Modul dient ausschließlich der Speicherung des Transaktionslogs.
- WAL (nach Write-ahead logging): Einteilung des PCM in zwei Zonen. In der ersten Zone werden modifizierte Seiten zwischengespeichert, in der zweiten Transaktionslogeinträge. Wird eine Seite aus dem DRAM verdrängt, wird sie in erster Zone gespeichert. Läuft diese Zone voll, wird ein klassischer Ersetzungsalgorithmus wie LRU angewendet, um erneut Speicherplatz zu schaffen. Endet eine Transaktion mit einem Commit, werden alle Logeinträge im DRAM in die Logzone des PCM heruntergeschrieben. Ist diese Zone voll, kann entweder ein Checkpoint-Prozess oder ein Zurückschreiben auf den externen Speicher erfolgen. WAL verwendet Update-in-Place.
- PCMLogging auf Satzebene (siehe Kapitel 6.3).

Festzuhalten ist, dass WAL in allen Testfällen besser als SCMLogging abschneidet. Das hat zum einen den Grund, dass WAL Daten im PCM zwischenspeichert, was die I/O-Kosten senkt, und zum anderen, dass das SCMLogging jeden neuen Logeintrag in den PCM schreibt, was höhere Latenzen verursacht, als das Schreiben in den DRAM, wie es WAL macht [1].



(a) Transaktionsdurchsatz (b) Antwortdauer der Transaktionen

Abbildung 8. Gesamtergebnis verschiedener Performance-Tests [1]

Abbildung 8 verdeutlicht die Effizienz von PCMLogging und bestätigt dadurch eine optimalere Ausschöpfung der PCM-Eigenschaften. PCMLogging erreicht mit zunehmender Größe einen um 19.2% bis 78.3% höheren Transaktionsdurchsatz als WAL. Des Weiteren erhalten wir bei steigender Modulgröße eine kürzere Antwortzeit des Systems.
Folgende Kriterien tragen zu den Testergebnissen bei:

- Mit steigender Größe des PCM nimmt die Wahrscheinlichkeit zu, dass ein zu lesender Satz oder eine zu lesende Seite bereits im PCM zwischengespeichert ist.

PCMLogging verwendet das gesamte Modul zum Zwischenspeichern von Sätzen mit integrierten Logeinträgen, während WAL Seiten und Logeinträge getrennt speichert. Demnach benötigt PCMLogging weniger I/O-Aktionen als WAL, da mehr Zwischenspeicher vorhanden ist. Zusätzlich ist die Wahrscheinlichkeit höher, dass mehrere modifizierte Sätze zu einer Seite gehören und dadurch beim Zurückschreiben auf den externen Speicher weniger I/O-Kosten anfallen, da weniger Seiten geladen und geschrieben werden müssen.
- Wenn der PCM voll ist, werden beim PCMLogging lediglich Sätze aus dem PCM gelesen, um diese auf den externen Speicher zurückzuschreiben, wohingegen es beim WAL ganze Seiten sind. Außerdem werden beim PCMLogging anschließend nur die angeforderten Sätze in den PCM geschrieben, während beim WAL eine ganze Seite geschrieben werden muss. Daraus folgt ein deutlich höherer Schreib-I/O-Aufwand beim WAL.
- PCMLogging vermeidet zeitaufwändige Operationen, wie die Ersetzung gesamter Seiten durch modifizierte, wodurch Transaktionen ihre Sperren kürzer halten als beim WAL oder SCMLogging. Als Konsequenz unterstützt PCMLogging mehr nebenläufige Transaktionen, ohne die Antwortzeit signifikant zu erhöhen.
- Der Commit-Prozess von PCMLogging benötigt verhältnismäßig wenig Zeit, da nur wenige Schreibzugriffe auf den PCM getätigt werden müssen.

Abschließend ist nicht außer Acht zu lassen, dass PCMLogging die Datenredundanz zwischen Logeinträgen und zwischengespeicherten Daten reduziert und der Recovery-Prozess ohne Checkpoints auskommt, da kein explizites Log gehalten wird. Zusätzlich ist der Recovery-Prozess sehr simpel und effizient [1].

7 Fazit

Als Fazit ist festzuhalten, dass Phasenwechselspeicher (PCM) die vielversprechendste Speichertechnologie ist, um heutigen DRAM zu unterstützen, bzw. zu erweitern. Aufgrund des geringen Stromverbrauchs, niedriger Zugriffslatenzen, Byte-Adressierbarkeit und vor allem der Skalierbarkeit steht PCM dem DRAM näher als SSDs. Eine weitere Eigenschaft, die es gekonnt zu nutzen gilt, ist die persistente Speicherung. Hier eröffnen sich neue Möglichkeiten, wie z.B. effizienteres Transaktionlogging und Recovery in PCM-gestützten Datenbanksystemen durch neuartige PCM-spezifische Algorithmen. Die Simulationsresultate des in dieser Arbeit vorgestellten PCMLogging-Schemas unterstreichen diese Behauptung. Ein schwerwiegender Nachteil der PCM-Technik ist jedoch der hohe Verschleiß, weshalb es nicht verwunderlich ist, dass sich viele Forschungsarbeiten mit der Verlängerung der Lebensdauer eines PCM-Moduls beschäftigen.
Erste Hardware-Hersteller sind auf PCM und dessen zukünftigen Nutzen aufmerksam geworden. So veröffentlichte beispielsweise der Halbleiterhersteller Numonyx ihr erstes PCM-Produkt im Jahr 2010. Das Unternehmen Samsung kündigte vor kurzem die Produktion eines 8GB-PCM-Moduls sowie die Entwicklung von Mobiltelefonen mit integriertem PCM an. In naher Zukunft sind ebenfalls Computersysteme mit PCM als Teil der Speicherhierarchie vorstellbar [1].

Literaturverzeichnis

[1] S. Gao, J. Xu, T. Härder, B. He, B. Choi and H. Hu, "PCMLogging: Optimizing Transaction Logging and Recovery Performance with PCM", submitted, 2013

[2] Y. Ou, S. Gao, J. Xu and T. Härder, "Wear-Aware Algorithms for PCM-Based Database Buffer Pools", in Proc. DaMoN, 2013

[3] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee and B.-G. Yu, "A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme", in Proc. ISCAS, pages 3014-3017, 2007

[4] M. K. Qureshi, V. Srinivasan and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology", in Proc. ISCA, 2009

[5] S. Chen, P. B. Gibbons and S. Nath, "Rethinking Database Algorithms for Phase Change Memory", in Proc. CIDR, pages 2131, 2011