Emerging Non-Volatile Memory and Initial Experiences with PCM Main Memory

Axel Grönberg
Abstract

Emerging Non-Volatile Memory and Initial Experiences with PCM Main Memory

Axel Grönberg

A group of new non-volatile memory technologies with characteristics making them worthy of consideration for different parts of the memory hierarchy, including the main memory, are emerging. In this thesis I discuss the state of STT-RAM, ReRAM and PCM technologies which are three of the front runners in this group of new technologies. I also simulate the performance of PCM used as main memory using Intel's binary instrumentation framework Pin and compare it to DRAM to explore three research questions. Firstly, in the case of horizontally integrated PCM and DRAM I test a data mapping policy where an application’s stack is mapped to DRAM and the heap is mapped to PCM. I find that in the case of my simulation this mapping have no benefits since most of the stack is continually kept in the cache which causes the DRAM to end up unutilized. Secondly, I compare the read latency between PCM and DRAM and find an average increase 48 % for PCM. Thirdly, I compare the energy costs of two write policies for PCM. The first being write-through of dirty bytes at byte granularity and the second being full row buffer write-back. I find that the first method has on average less than a third of the energy cost compared to the second method.
Acknowledgements

I would sincerely like to thank my supervisor, Per Ekemark, and my reviewer, Professor Stefanos Kaxiras, for their support, guidance, and patience with me throughout this project.
Contents

Acknowledgements 1

Contents 2

1. Introduction 3

2. Background 4
   2.1 Emerging Non-Volatile Memory Technologies 4
   2.2 Architectures and System Design 5
   2.3 Spin Transfer Torque Random Access Memory 6
   2.4 Resistive Random Access Memory 8
   2.5 Phase Change Memory 10
   2.6 DRAM Architecture and Operation 12
   2.7 Intel’s Pin 13

3. Research Questions 14

4. Methodology 15
   4.1 Simulation Structure and Cost Models 15
   4.2 Experiment 19

5. Results and Discussion 19
   5.1 Research Question 1: Stack-based Data Placement 20
   5.2 Research Question 2: Read Latency in DRAM and PCM 22
   5.3 Research Question 3: PCM Write Policy Energy Optimization 23

6. Related Work 25

7. Conclusions and Future Work 27

8. References 28
1. Introduction
For a computer system to scale, the different interdependent systems within it must scale together, in order to avoid performance bottlenecks. Sadly, in recent times there are multiple trends causing the memory system to become such a bottleneck. Applications have become more and more data intensive. The rise of big data, IoT, machine learning, social networks, and video playback all contribute to this [1, p. 14:2]. Among main memory technologies, DRAM has been the dominating technology for many decades, and while there has been strong progress made when it comes to size, capacity, and performance, current predictions state that the scaling of DRAM will hit a wall sometime within the coming decade. The decrease in size of the DRAM cell have enabled higher speed and density for a lower per-bit price, but this scaling seems to come to a halt due to the technological difficulties of scaling nodes below 30 nm (due to lower data retention time, and a too low sense margin, among other reasons) [2], [3]. The volatile nature of DRAM technology also poses major problems related to energy and performance. A theoretical 64 Gb DRAM device would, with current technology, have to spend 46 % of its time refreshing its memory cells, which would account for 47 % its power use. Compare this to a typical 4 Gb device, only spending 8 % of its time and 15 % of its energy on refreshing its memory cells [2].

These problems with DRAM technology make the emergence of new non-volatile memory (NVM) technologies extremely interesting and it is important to consider these as replacements or compliments to the current DRAM technology for use in main memory (as well as in other parts of the memory hierarchy). The emerging NVM technologies are a group of quite different memory technologies that have speeds approaching that of DRAM (something that an earlier non-volatile technology, namely NAND Flash, has not come close to), potential for scaling to lower feature sizes, and non-volatility in common. These new technologies are, however, not strictly better than DRAM. Instead, they have their own advantages and disadvantages, making their potential of being integrated into the memory hierarchy an important research topic.

While there are additional interesting emerging NVMs not discussed at length here (most notably FeRAM), this paper will present three of the major upcoming NVM technologies: spin transfer torque RAM (STT-RAM), resistive RAM (ReRAM), and phase change memory (PCM). PCM will be given extra focus, mainly due to it being the most mature technology [4, p. 28], and the thesis will explore some aspects of PCM’s possible performance as a main
memory replacement as well as in combination with DRAM with the help of performance models, benchmarks and dynamic binary instrumentation.

2. Background
In this chapter, I describe the emerging NVM technologies as a whole and a few variants, PCM, ReRAM, and STT-RAM, more closely. I then detail how DRAM is structured and operates. Lastly, I describe Intel’s dynamic instrumentation framework, Pin, which is be used for the experiments in this thesis.

2.1 Emerging Non-Volatile Memory Technologies
As mentioned in the introduction, DRAM is now coming up to its limits when it comes to scaling to smaller nodes. NAND Flash technology, an earlier NVM technology that has become very widespread, is facing similar problems with scaling [3]. Flash turned out to mainly be employed as a storage technology and to hardly influence the main memory level of the hierarchy at all. While Flash does have a lower per-bit cost compared to DRAM and is non-volatile, its write endurance is only on the scale of $10^4 – 10^5$ (very low compared to DRAM’s practically unlimited write endurance) which is far too low considering the write frequency of main memory. Flash also has multiple orders of magnitude higher read and write latency as well as an extra “ERASE” operation required to overwrite previous data, which also adds additional latency [5, p. 1538]. The combination of these features made Flash much more suited for storage than main memory. The emerging NVM technologies discussed in this thesis all improve upon the limitations of NAND Flash substantially and therefore warrant being studied as possible main memory technologies.

The new non-volatile technologies each have their own unique strengths and weaknesses, both in how they are made and how they perform. In the following sections, I discuss three out of the main up and coming NVM technologies, namely: spin-transfer torque random access memory (STT-RAM), resistive random access memory (ReRAM), and phase-change memory (PCM). These three technologies were deemed the most promising by the International Technology Roadmap for Semiconductors organized workshop on Emerging Research Devices (ERD) in 2014 [4], and are commonly brought up in surveys of NVM technology [1], [5], [4]. Ferroelectric field-effect transistor (FeFet) memory is also commonly cited as a possible future NVM memory technology but is not discussed here as it is deemed less mature than the other technologies. There are of course more possibilities than to just use
these memories in the main memory part of the memory hierarchy, such as cache or as storage.

These emerging NVM technologies have a number of things in common. They have better scalability than DRAM: they provide higher density compared to traditional technologies and have lower energy costs. The lower energy costs are, however, not derived from lower read or write energy, these costs are higher (especially the write energy). Instead, the energy cost benefits are directly derived from the non-volatility of the technology. It avoids the costs of having to regularly refresh every memory cell as DRAM technology is forced to due to power leakage of the capacitors in its memory cells. Another property they have in common is their lower endurance (which is more strongly affected by writes) when compared to the practically infinite endurance of DRAM, albeit much better than that of NAND Flash [1].

The persistency provided by NVM technologies may have a large impact on systems as a whole and could bring dramatic qualitative changes. Main memory that is not volatile could change how future file systems are built and maybe even how entire operating systems are built. There might be no booting on restart, and if power is lost a system could more easily continue where it left off [6]. One could also imagine a microreboot which is more selective in what parts of the OS is to be reloaded, by loading particular checkpoints for example [6]. There are, however, not only advantages, non-volatility sometimes have its drawbacks. There is for instance the possibility that a process could forget or be unable to destroy decrypted data or an encryption key after it is done with it, thus causing these sensitive objects to remain in memory. This poses a possible security risk which is absent in the case of the volatile main memory.

2.2 Architectures and System Design
When considering how to integrate these new technologies into the memory hierarchy there are three main options: horizontal integration, vertical integration and replacement. Horizontal integration means that the NVM technology is placed on the same level as another technology. In relation to main memory the natural way to do this is placing the NVM as a main memory on the same bus as DRAM and there would be some mechanism for managing the placement of data in order to maximize performance. Vertical integration means that the technology is placed above or below another existing technology. In relation to main memory, this would entail placing the NVM between DRAM and storage, using the DRAM as a cache
since NVMs tend to have higher latency, higher energy costs, and lower endurance compared to DRAM. Replacement simply refers to replacing a current technology [1].

When these types of technologies initially were thought up, many hoped that at least one of them could become a “universal memory”, filling the needs of all level of the memory hierarchy: high density and endurance and low cost, energy, and latency. This would unify the hierarchy, removing the need for splitting it into CPU cache, main memory and storage, a division that have dominated architectures for many decades. The vision that any of these technologies could be that “universal memory” is now generally discounted. It is deemed very hard to achieve the optimization required by today’s applications with a single technology. At any level of the memory hierarchy the specific characteristics sought after can differ by many orders of magnitude compared to another level, which makes it hard for a single technology to adequately meet all of these needs [7, p. 193].

2.3 Spin Transfer Torque Random Access Memory
Spin transfer torque random access memory is a variation of magnetic RAM or MRAM, a type of RAM that is based on special materials whose magnetic orientation can be manipulated in order to program binary data. STT-RAM uses the same kind memory element as ordinary MRAM, namely, a magnetic tunnel junction (MTJ), which consists of two ferromagnets separated by a very thin insulation layer, usually MgO. The high and low resistance states are dependent on whether the two magnetic orientations of the ferromagnets are parallel or anti-parallel (opposite directions) [4, p. 27]. The two ferromagnetic layers are called the reference layer and the free layer, and it is the magnetic orientation of the free layer that is altered to change the bit representation. What sets STT-RAM apart from ordinary MRAM is how it switches the state of the free layer. Conventional MRAM generates a magnetic field by passing current through another component external to the MTJ to write to the memory cell: a write word line. STT-RAM on the other hand makes use of the spin transfer torque effect, which can change the magnetic orientation of the free layer by running a spin-polarized current through the MTJ from the reference layer to the free layer, or vice-versa. Since current only needs be passed through the MTJ and not the word write line, as in the case of conventional MRAM, this reduces the size of the memory cell. The smaller memory cell size also leads to a reduction in the write energy needed, which in turn leads to better scaling properties [8, p. 615].
STT-RAM have, compared to the other the emerging NVM technologies that are discussed in this thesis, demonstrated the fastest write speed and is estimated to have the highest endurance. It also received most votes as the most promising technology in the previously mentioned ERD workshop [4]. However, it does have some drawbacks. One of these drawbacks is that it suffers from thermal instability. This means that data can be lost due to influence from temperature. Thermal instability also increases as the technology decreases in size, making this a problem for STT-RAM scaling. Another issue is that the read operation also requires current which may change the magnetic orientation of the read memory cell (a read disturb). This is problem is exacerbated by the fact that the required current for writing decreases as the technology shrinks, while the current needed to read at the same time increases, thus increasing the chance of a read disturb and faulty behavior [1].

When it comes to architectural integration, STT-RAM seems to be best suited for the cache and memory levels of the memory hierarchy. This is due to its endurance (at least theoretically) being on the level of DRAM and its read latency sometimes is even lower than that of DRAM. However, its write performance compares unfavorably to DRAM. Most of recent research has been focused on in some way implementing STT-RAM into cache systems [1]. When used instead of SRAM, the technology has been shown to be able to reduce the static energy costs; however, the dynamic energy costs go up [9]. The data retention time is related to the write current, which has led to experiments in relaxing the retention time in order to save energy and cell size for STT-RAM caches [10]. When looking at main memory integration, Emre Kültürsay et al. [11] have shown that simply replacing DRAM with STT-RAM leads to worse performance due to write latency, as well as increased power costs due to the high write energy. However, the same researchers also showed that taking advantage of the fact that row buffers and sense amplifiers can be separated for STT-RAM (which is not possible for DRAM) leads to the potential of reducing the number of unnecessary writes by performing selective and partial writes. They also took advantage of the observation that writes usually have less locality, thus letting writes bypass the row buffers to increase row buffer hits on reads. These two optimizations lead to performance comparable to DRAM and a 60% decrease in power consumption. These results were based on simulation and there are worries that the actual technology will not be able to scale to the theoretical level, and that the write energy is too high for many applications. There are ideas for ways to solve these problems, by for instance relaxing retention times [12]. Density seems
to limit the possibilities of using STT-RAM for storage, but the technology may have potential as a storage cache [13].

The state of STT-RAM technology is quite mature, and physical phenomena associated with it are well understood. It seems most promising out of the technologies discussed in this thesis for embedded applications [4, p. 27]. There have for many years been prototypes shown by researchers, but also commercial STT-RAM products have been made available to the market by companies such as Everspin and NEC [14, p. 3]. Although STT-RAM is comparatively sensitive during manufacturing, the process for making the technology is continually improving [4, p. 28].

2.4 Resistive Random Access Memory
Resistive random access memory, or ReRAM, usually refers to two types of non-volatile memory technologies. The first, also known as Oxide-RAM (OxRAM), have memory cells consisting of two metal electrodes with a metal oxide in between them. When a certain electric voltage is applied to the cell it causes the creation of a conductive filament in the oxide made from oxygen vacancies, which change the resistive properties of the oxide. Applying a voltage of opposite polarity to the cell reverses the process, and breaks the filament, making it possible to have this resistive change represent binary data [1], [7]. The other type of ReRAM is called conductive bridge RAM (CBRAM), it has similar properties and the data is also programmed in high resistive and low resistive states through the creation of a conductive filament. Instead of an oxide, a solid electrolyte is used and the metal in one of the electrodes is swapped for a metal that easily oxidizes. When an electric field is applied to the memory cell, the metal ions oxidize and wander into the electrolyte and form conductive filaments creating a low resistive state. The process can then be reversed by changing the polarity of the voltage and thus returning to a high resistive state much like in the OxRAM technology [7], [15].

One of the big advantages of ReRAM is that it uses materials that are common in current semiconductor manufacturing, which should make it easier to adapt in production [7]. Another advantage of ReRAM memory is that it has a lower energy consumption compared to PCM, as well as smaller cell size as compared with MRAMs (such as STT-RAM). Its biggest challenge is its endurance. Its endurance is better than that of Flash but compared to PCM and
STT-RAM it is lacking [1, p. 14:22]. Another big problem with the technology, applicable to both OxRAM and CBRAM, is that the process of forming the filament, be it made of ions or oxygen vacancies, is stochastic in nature, causing the resistance between the two states to vary from device to device and also between different cycles on the same device. This resistance variation is much greater than in PCM and STT-RAM and makes sensing harder and adds a need for write-verification [16].

How ReRAM is to be integrated into the memory hierarchy is still an open question. If ReRAM were to be applied as main memory, its endurance is of main concern. One could imagine a horizontal integration of ReRAM with DRAM, where data is split between the memories in such a way as to not wear out the ReRAM prematurely. One could also use DRAM as a cache for the ReRAM to decrease the number of unnecessary writes to the ReRAM cells. Xu et al. [17] showed in their simulations that, with some optimizations to the ReRAM chip and some data encoding, they were able to reach more than 90% of DRAM performance. When looking to integrate ReRAM into caches, here too the main problem is its low endurance. When Dong et al. [18] replaced the typical SRAM with ReRAM in various levels of the cache in a simulation, they found that this in general reduced the energy costs for the cache but also reduced performance, mainly due to the slower write operations of ReRAM. One could see ReRAM being found useful in certain narrower applications with particular types of code. Komalan et al. [19] have for instance explored using ReRAM in caches for systems running applications with loop dominated code, which is less write intensive and thus more suitable for ReRAM’s properties. The endurance properties of ReRAM might make it most suited to compete with, or complement, Flash on the storage level of the memory hierarchy. The possibility for horizontally integrating the technology with NAND Flash, using ReRAM as a way to absorb the most intensive and random writes, could both increase performance of the system and increase the lifetime of the Flash storage.

ReRAM technology is quite mature but has taken longer than expected to reach the market due to the complexities behind controlling the memory cells. Panasonic was the first company to ship ReRAM in 2013, integrated into an 8-bit MCU [20]. As of 2017 there was only two companies (Panasonic and Adesto) providing standalone ReRAM [21]. There are, however, many companies working on their own ReRAM technology, such as Samsung, Crossbar, Sony, and others. ReRAM currently seems promising is in embedded systems, such as IoT devices, where the low power and low cost of ReRAM is especially valued, and where write
speeds are not necessarily very important. ReRAM comes in multiple configurations, the *1-transistor-1-resistor* configuration for embedded systems and a stacked configuration, which is intended for storage class memory in SSDs. Whether ReRAM will become the main technology in any of these areas remains to be seen, STT-RAM for instance is viewed as very promising for embedded systems and with many large companies betting on it, it is a strong competitor to ReRAM in that domain. Maybe ReRAM will find its place in some more niche field such as neuromorphic computing. This is a new form computing which uses analog not digital circuits, emulating neurons in the brain, and OxRAM has an analog property making it possible to integrate in such a system [20].

### 2.5 Phase Change Memory

The basic composition of a PCM cell are two electrodes separated by a resistive heater and a phase change material. The phase change material is usually a chalcogenide glass, most commonly Ge$_2$Sb$_2$Te$_5$ (GST), but other materials can be used. A phase change is caused by running current through the resistive heater and heating the chalcogenide. The phase change in the chalcogenide alters the material’s resistance, which is used to decide the binary information in a cell. The two main phases are a crystalline (low resistance) phase and an amorphous (high resistance) phase. The RESET operation, melting the material resulting in an amorphous state, is done through quick and intense heating. The SET operation on the other hand, resulting in a crystalline state, is done through longer, less intense heating. Reading from a PCM memory cell does not affect the phase of the material, allowing for non-destructive reads [22], [23].

Phase change memory has some very positive scaling behaviors as devices get smaller. It has decreased thermal conductivity (which leads to higher power efficiency), higher crystallization temperature (which leads to longer data retention), better endurance as cells gets smaller, etc. The main obstacle for smaller memory cells is the selector devices. Selector devices are used to reduce leakage current for memory cells arranged in crossbar arrays (an architecture where the memory cells are located in the junctions between wordlines and bitlines and allow for particularly low densities and stacking of memory arrays among other things [24]), which the emerging NVM memory cells most likely will be due some of its advantages. There are multiple types of selector devices being researched in order to achieve
the smallest cell size possible. The three main contenders are: bipolar junction transistors (BJT), vertical transistors and diodes [25], [4].

There are different degrees of crystallization (with different degrees of resistance) that the phase change material can occupy, allowing for the possibility that each memory cell to store multiple bits in the same area, which would multiply the technology’s memory density. There is a problem with this however, the resistance states drift with time towards higher resistance, complicating the process of distinguishing between the different states [1, p. 14:10], [7].

The write speed of PCM is in the tens of nanoseconds, it also has a relatively large programming current, and these are areas in which it compares unfavorably to STT-MRAM, RRAM and CBRAM. Another major problem with PCM cells is their limited endurance, similar to that of ReRAM. The cells can only sustain a certain number of writes operations before the reliability of writes become too low [1, p. 14:10].

The possibilities for integrating PCM are mainly at the main memory level and toward longer-term storage. This is mainly due to its expensive writes and limited endurance. PCM is in general not suited for a CPU cache, however, it is a competent competitor to Flash as storage, due to its faster reads and writes and higher endurance [1, pp. 10-11]. This storage alternative is already on the market in the form of Intel’s Optane. On the main memory level, there has been much research improving upon PCM’s weaknesses in order to increase system performance. The research has mainly focused on hiding write latency as well as on avoiding unnecessary writes to PCM in order to increase performance and mitigate wear. There have also been experiments where PCM have been used for checkpointing [26] or for storing file system metadata [27]. Qureshi et al. [28] have found a tripling of speed and of lifetime when vertically integrating PCM with a DRAM cache 3 % the size of the PCM. There have also been combinations of both vertical and horizontal integration, where PCM equipped with a small DRAM cache was horizontally integrated with DRAM, and data placement was governed by multiple algorithms in order to further improve performance and energy efficiency [29]. The hope is that one could be able to not only have performance comparable to DRAM, but also save energy thanks to the possibility of avoiding having to refresh memory cells like in DRAM.
PCM is the most mature out of the technologies discussed in this thesis [4, p. 24]. Many large companies have invested a lot into the technology; big companies such as Micron, IBM, Samsung, etc. have demonstrated prototypes or even have/have had products available on the market [1, p. 14:14]. In addition, the commercially available 3D XPoint technology developed jointly by Micron and Intel is believed by many to be based on PCM technology, including Chinese researchers in a paper from 2018 [30]. Although, this is denied by Intel, who’s 3D Xpoint product is available under the name Optane in an SSD version [31]. Intel is also providing a main memory solution based on the same technology named Optane DC Persistent Memory, intended for horizontal integration with DRAM in data centers [32]. Micron’s 3D XPoint technology QuantX is expected to be out in 2019. SK Hynix is also working on a competitor to the 3D XPoint based on PCM technology [33].

2.6 DRAM Architecture and Operation
A DRAM cell consists of a single transistor-capacitor pair, whose charge is used to represent digital information. Due to the incorporation of a capacitor into the design, the DRAM cells leak and need to be periodically refreshed. A DRAM memory array is built out of rows and columns of DRAM cells, and it is this grid pattern that is used to address the DRAM memory cells, giving each cell a row address and a column address. Each row is typically very long, containing a large number of bits, and a column is typically is 8 bits wide. These cells are connected to bit- and wordlines, which are used to read and write data to the cell. A DRAM chip contains many of these arrays, called banks, which can be controlled independently. The accesses to the different banks can be interleaved which can lead to higher bandwidth memory [34, pp. 316-317].

Each bank has its own row buffer (also known as sense amplifier) which is used to detect, store, and amplify the charge of the capacitors in the memory cell through the bitlines. A read or write requires a number of operations to be performed in order to bring the data to the row buffer and either modify it or send it over the data bus towards the processor. Each of these operations require a certain amount of time to perform and give rise to differing amounts of latency. Reading or writing data from an already open row (a row that currently is in the row buffer) takes less time than closing it and opening another to read or write to [34, p. 320].

Multiple DRAM chips are usually organized together to form a rank and the chips of the ranks typically work together to serve a memory request. A DIMM (or dual-inline memory
module) consists of one or two ranks which work independently from each other. The data in memory can be mapped in different ways to banks, chips and ranks, which can affect performance. A part of the memory address is used to identify the row address, another the column address in the memory arrays. There is also a bank identifier, which typically uses fewer bits due to the comparatively small number of banks. There is also a bit for identifying the addressed rank of a DIMM [34, p. 321].

2.7 Intel’s Pin
Dynamic binary instrumentation is used to gain detailed knowledge about the behavior of an existing application in its binary form through the insertion of code. Pin, which is a framework created by Intel, does this code insertion at runtime through its just-in-time (JIT) compiler and is widely used to instrument applications for everything from research to security purposes. Pin is available on multiple instruction-set architectures, including IA-32, x86-64 and MIC among others, and can be run on Window, Linux, and OS X operating systems [35].

Pin allows users to create so called Pintools which are written in C/C++. These tools can access Pin’s API and analyze an application on an instruction level. The Pin API abstracts away most of the underlying architectures peculiarities making the created Pintools quite portable. The API gives users access to both the actual addresses that a program would use as well as the data. It also gives users access to the CPU context, control flow and memory. Pin also gives the user the ability of somewhat altering the application’s behavior by overwriting the application’s registers or memory [36].

As previously mentioned, the instrumentation is enabled by a JIT compiler that takes the executable as input. The JIT compiler compiles code in basic blocks, and after compilation of each basic block Pin transfers control to the executable and lets it run but makes sure that Pin itself regains control at the end of that block. Every time a new block is compiled, Pin has the opportunity to instrument it by inserting code of its own. To speed up execution, Pin saves recurring blocks of code in a code cache to avoid recompilation [36].

Pin’s software architecture is made up by a virtual machine (VM), a code cache, and instrumentation API. The VM itself is made up by a JIT compiler, an emulator, and a dispatcher. The dispatcher launches the instrumented code and the emulator handles
operations such as system calls, which require special handling since it is not user level code. When an application is being instrumented there are three active binaries: Pin, the Pintool, and the application. Pin performs JIT compilation and code insertion while the Pintool contains calls to the instrumentation API. All of these share the same address space; Pin injects itself into the address space of the application much like a debugger does. After Pin is initialized, the Pintool is also loaded into the address space and initialized. The Pintool then calls upon Pin to start running the application. [36]

The slowdown caused by the instrumentation is mostly due to the added code rather than the extra compilation. For this reason, it is comparatively useful to optimize these instrumentation calls. Most analysis routines are rather simple (e.g. counting or tracing) and Pin does not have much power to simplify complex routines, one of the few ways in which Pin can optimize these calls is through inlining [36].

3. Research Questions
In the upcoming sections I will look into the following research questions.

1. **Stack-based Data Placement**
   In the case of horizontal integration of PCM with DRAM there is the question of where to place what kind of data in order to optimize performance, endurance, and energy. A simple policy would be to place the stack of an application in DRAM while the application’s heap is placed in PCM. The motivation behind this policy is the assumption that the stack is written and accessed more frequently compared to data on the heap, and it could potentially be beneficial to place the most frequently used data in DRAM, which does not have an endurance problem and is faster. In order to explore this policy, one can initially study how cache misses are distributed between stack addresses and heap addresses.

2. **Read Latency in DRAM and PCM**
   This thesis will also look into the difference in latency between PCM and DRAM. To do this, and make it comparatively simple, writes are assumed to be hidden thanks to optimizations such as prefetching, write buffers and scheduling. This way, I can simply count the number of row buffer hits and misses of read instructions. I then use models for DRAM, PCM and row buffer performance for hits and misses (and dirty misses in the case of PCM, due to PCM’s asymmetrical read/write latency) and compare the results in order to get an idea of the differences in the chosen workloads.
3. PCM Write Policy Energy Optimization

Lastly, I investigate a possible energy optimization to PCM. I want to see the amount of energy that can possibly be saved by immediate write-through and only writing back dirty bytes, i.e. performing partial writes on byte granularity, as opposed to writing the entire row in row buffer (including both dirty and clean bytes) when evicted (write-back). In order to investigate this, one can look at the number of dirty bytes in total and how much in energy this would cost according to an energy model, and compare that to the number of bytes that would have been written if the entire row buffer would have been written back and the energy this would cost. I also include the energy for writing to the row buffer. Then I compare those energy numbers.

4. Methodology

In this chapter, I describe the models and methods used to answer my research questions. In order to investigate these questions I simulate a simplified version of a memory system and use a number of benchmarks from the SPEC CPU2006 benchmark suite to supply representative application behavior. These benchmarks are instrumented with the binary instrumentation framework Pin from Intel, which feeds information about read and write instructions to the memory system representation, which here is a simple cache and row buffer simulator, in order to see how they interact and to record the operations taking place. I apply cost models to the information gathered in order to approximate energy and latency costs of PCM and DRAM when running these different applications and compare the results of the two different technologies.

4.1 Simulation Structure and Cost Models

In order to simulate the accesses to the main memory, I use a simple last level cache (LLC) simulator and a simple row buffer simulator (capable of representing multiple row buffers) and use the data the Pintool finds by instrumenting the benchmarks, as shown in figure 1. The simulators handle memory requests and simulates inner states of each component as well as tracks statistics on their usage.
The Pintool instruments each read and write instruction of a given application, gets the address for the instrumented operation and sends it into the cache simulator. The cache simulator checks to see if the address is present in the cache. It also checks to see if the address is to the heap or the stack. This is done by using the Pin API to access the process’ stack pointer and all addresses higher than or equal to the stack pointer is deemed to be to the stack while all addresses lower than the stack pointer is deemed to be to the heap, since the stack in x86 systems grows from higher addresses to lower. This is, of course, a minor simplification since this does not take in to account the data and text segments for instance. If the address is present in any of the cache lines in the cache, a hit is recorded, dirty bits are set if needed, and the corresponding cache line is put first in an abstraction of a linked list representing the usage order. Otherwise, the cache line containing the correct address is added to the appropriate place in the cache, a cache miss is recorded, dirty bits are updated if needed, and the address is then looked for in the row buffers. A row buffer hit or miss is recorded, the row buffer page is updated in the case of a miss and the row buffer’s dirty bit is updated in the case of a write. The process of how an instrumented instruction works itself through the simulators can be seen in figure 2. If the operation straddles a cache line this is also taken care of by initiating another appropriate access to the correct address and with the correct size.
The cache simulator uses a least recently used (LRU) cache replacement policy and can be run in both set-associative and direct-mapped mode. The size of cache lines can be configured as well as the size the entire cache and the size of the sets in the case of a set-associative cache. The data structure representing the LLC contains an array with pointers to each set of cache lines inside the cache, as well as internal statistics keeping track of relevant internal...
operations, and numbers needed for the simulation to run correctly. Each cache line is represented by another data structure containing the tag of that line as well as dirty bits for each byte in the cache line, in order to keep track of which bytes were written too. It also contains a pointer to the next line in the set. In order to keep track of the most recently used cache line the pointers of the cache lines are arranged into a linked list in which the cache lines are ordered according to the most recent access. After each access to a set, the pointers are rearranged to represent the updated order and when a cache line needs to be evicted, the last line in the set is evicted.

In a typical DRAM DIMM, a rank of chips work in unison to provide data for a memory access. This also means that the content of each individual row buffer of the same bank across the different chips in the rank change at the same time when serving a memory request. [34, p. 318]. In this simulator I have therefore chosen to model these cooperating row buffers as big extended row buffer data structures. The size of each actual row buffer is multiplied with the number of chips to get the size of the extended row buffer, which represents the row buffers for a certain bank in each chip. The number of extended row buffers in a particular simulation is thus dependent on the number of banks per chip. If we for instance have 1 chip with 1 bank, this makes 1 extended row buffer but if we on the other hand have 1 chip with 8 banks it would make 8 extended row buffers. The data in main memory can be mapped across main memory in different ways. One common way, which is used here, is row interleaving, this means that data addresses are mapped in such a way that consecutive rows are mapped to consecutive banks. This tries to take advantage of row buffer locality [37, p. 4]. A dirty bit on each extended row buffer keeps track of whether any data in it was written to or not. The row buffer can also be configured to let writes bypass the row buffer if that optimization is desired. The row buffer simulator keeps statistics on read and write hits and misses and whether the buffer was dirty or not.

In tables 1 and 2 you can see the simple memory timings as well as write energy costs used in the model to calculate and compare the latency and energy costs for each technology.

<table>
<thead>
<tr>
<th></th>
<th>DRAM</th>
<th>PCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row buffer hit</td>
<td>40 ns</td>
<td>40 ns</td>
</tr>
<tr>
<td>Row buffer clean miss</td>
<td>80 ns</td>
<td>128 ns</td>
</tr>
<tr>
<td>Row buffer dirty miss</td>
<td>80 ns</td>
<td>368 ns</td>
</tr>
</tbody>
</table>
Table 2. Energy costs for write operations [38].

<table>
<thead>
<tr>
<th></th>
<th>Write energy cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>0.39 pJ/bit.</td>
</tr>
<tr>
<td>PCM</td>
<td>16.82 pJ/bit.</td>
</tr>
<tr>
<td>Row Buffer</td>
<td>1.02 pJ/bit</td>
</tr>
</tbody>
</table>

4.2 Experiment

The benchmarks used are a representative subset of the SPEC CPU2006 benchmark suite, the same as the ones used by Emre Kültürsay et al. in a paper looking into STT-RAM as an energy efficient alternative to DRAM [11]. The benchmarks are compiled with gcc with the –O3 optimization flag.

The cache simulator is configured to model the LLC of an Intel Core i7-9700K, a high-end CPU introduced late 2018, making the LLC 12 MiB in size and 12-way set-associative [39]. The size of the row buffer is configured to be 4 KiB, as in Emre Kültürsay et al.’s paper, and the number of chips employed is configured to be 8, which gives extended row buffers of 32 KiB. Further, the number of banks on each chip is chosen to be 8, which gives 8 of these extended row buffers.

The simulator collects statistics on memory operations performed by the instrumented benchmarks. The most important numbers that are tracked include: hits and misses in the LLC as well as whether they are addressed to the heap or the stack, hits and misses of reads and writes to the row buffer, as well as the number of bytes written and the number of dirty row buffer evictions. These numbers are then used along with the cost models presented in section 4.1 to simulate the underlying PCM and DRAM technologies.

5. Results and Discussion

The results from the characterization of memory operations derived from running the simulations are shown below where we can see how these benchmarks differ in size and in amount of reads and writes as well as their geometric mean.
5.1 Research Question 1: Stack-based Data Placement

With regards to the first question, whether the division of data between DRAM and PCM in terms of stack and heap is a viable alternative, the results in table 3 quite clearly shows that it is not so. Table 3 shows the percentage of memory accesses to the heap and stack, how many cache misses there was, as well as what percentage of the misses was to the stack and heap respectively. One can see that the number of misses that are to the stack is less than one percent across all benchmarks, with a geometric average of 0.01 %. There doesn’t seem to be any obvious relation to the percentage of stack or heap access and the proportion between stack and heap misses. Using this division of data would in effect be equal to placing all the data in the PCM and not making use of the DRAM at all.
Table 3.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Cache accesses to stack</th>
<th>Cache accesses to heap</th>
<th>Cache misses to any address</th>
<th>Portion of total cache misses to stack</th>
<th>Portion of total cache misses to heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.perlbench</td>
<td>23.07%</td>
<td>76.93%</td>
<td>0.04%</td>
<td>0.01%</td>
<td>99.99%</td>
</tr>
<tr>
<td>416.gamess</td>
<td>20.29%</td>
<td>79.71%</td>
<td>0.00%</td>
<td>0.89%</td>
<td>99.11%</td>
</tr>
<tr>
<td>429.mcf</td>
<td>1.47%</td>
<td>98.53%</td>
<td>8.27%</td>
<td>0.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>435.gromacs</td>
<td>23.02%</td>
<td>76.98%</td>
<td>0.00%</td>
<td>0.04%</td>
<td>99.96%</td>
</tr>
<tr>
<td>436.cactusADM</td>
<td>79.09%</td>
<td>20.91%</td>
<td>1.17%</td>
<td>0.01%</td>
<td>99.99%</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>39.13%</td>
<td>60.87%</td>
<td>0.18%</td>
<td>0.16%</td>
<td>99.84%</td>
</tr>
<tr>
<td>454.calculix</td>
<td>60.36%</td>
<td>39.64%</td>
<td>0.13%</td>
<td>0.01%</td>
<td>99.99%</td>
</tr>
<tr>
<td>456.hmmer</td>
<td>22.95%</td>
<td>77.05%</td>
<td>0.00%</td>
<td>0.01%</td>
<td>99.99%</td>
</tr>
<tr>
<td>458.sjeng</td>
<td>36.00%</td>
<td>64.00%</td>
<td>0.10%</td>
<td>0.06%</td>
<td>99.94%</td>
</tr>
<tr>
<td>462.libquantum</td>
<td>4.67%</td>
<td>95.33%</td>
<td>21.15%</td>
<td>0.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>465.tonto</td>
<td>45.25%</td>
<td>54.75%</td>
<td>0.04%</td>
<td>0.02%</td>
<td>99.98%</td>
</tr>
<tr>
<td>470.lbm</td>
<td>0.35%</td>
<td>99.65%</td>
<td>5.43%</td>
<td>0.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>19.15%</td>
<td>80.85%</td>
<td>2.21%</td>
<td>0.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>483.xalancbmk</td>
<td>16.96%</td>
<td>83.04%</td>
<td>0.35%</td>
<td>0.82%</td>
<td>99.18%</td>
</tr>
<tr>
<td>G. mean</td>
<td>15.76%</td>
<td>67.46%</td>
<td>0.10%</td>
<td>0.01%</td>
<td>99.85%</td>
</tr>
</tbody>
</table>

These results show us that in the case of our simulator setup, separating data this way will not help performance or cost. One could imagine that this separation could have had a bigger effect if the cache was of smaller capacity. In that case the stack might not fit in the cache and might more often be read from memory and the split discussed might have had a positive impact on such a system.

It should be noted that since the stack is kept in the cache more or less the whole time one could argue for using the opposite data placement policy, mapping the heap to DRAM and the stack to PCM, since almost all misses are to the heap in the case of this experiment. More generally, this is an indication of how important cache locality is when designing this kind of data placement policy. It shows that it is not only the amount of read and writes that affect how often certain data will be read or written to in main memory, but that cache locality can also greatly affect this number. If cache locality is very high it doesn’t matter if the addresses are read or written a lot since this will mainly happen inside the cache and only when the data is loaded into or evicted from the cache does main memory need to be involved.
5.2 Research Question 2: Read Latency in DRAM and PCM

Here I look at how the memory read latency changes if we replace DRAM with PCM. In figure 4, one can see how common each type of row buffer event is for reads, and in figure 5 one can see what the read latency is with PCM compared to DRAM. We see across the board an increase in latency, as expected due to PCM’s longer memory timings. The benchmark most adversely affected by using PCM instead of DRAM are those that have the largest proportion of dirty misses, which is also to be expected due to its longer time requirement. The geometric mean ratio of PCM to DRAM read latency is 148% across the benchmarks. It is important to stress that this does not imply a mean increase in overall runtime of 48% for all the benchmarks since we are only looking at a subset of instructions, the read instructions (which make out roughly 26% of the total instructions in the geometric mean of the instrumented benchmarks), as opposed to looking at all operations performed by the applications.

Figure 4.
My results imply that one would need PCM to provide major improvements in another area to motivate employing it instead of DRAM as main memory. Such improvements could be lower energy costs or some more qualitative benefits that its non-volatility could infer.

My results also indicate that it is very important to keep the number of dirty row buffer misses down in order to keep the performance degradation to a minimum. One factor in the number of dirty misses is how row buffers are architected. In this experiment, I have used a simple model with 8 chips, 8 banks, and quite large row buffers. One could imagine getting quite different results using other parameters and more advanced architectures, like something closer to what is implemented in DDR4 DRAM (with smaller row buffer sizes and bank groups among other differences).

5.3 Research Question 3: PCM Write Policy Energy Optimization

When going from the method using write-back but non-partial writes to the method using write-through at byte granularity we see in figure 6 that 9 out of 14 benchmarks’ write energy consumption goes down substantially and the geometrical mean is down to 29%. For the remaining five benchmarks, a likely explanation is that they keep the correct row open and can continually write to it before a row buffer miss thus resulting in a smaller number of bytes written to the PCM when using the row buffer write-back method compared to partial write-through on byte granularity.
From the results one can see than one should try being smarter than writing back the entire row buffer when only a few bytes are dirty. There is, however, a balance to be struck between energy efficiency and the peripheral circuity needing to be introduced, particularly if you want to write dirty bytes on byte granularity, as it requires a large number of dirty bits.

**Figure 6.**

![Energy cost of write-through at byte granularity normalized to energy cost of write-back with non-partial writes](image)

**Figure 7.**

![Energy cost of best PCM method normalized to energy cost of DRAM full row buffer write-back](image)

In figure 7 I compare the energy cost of the write method with the lowest energy cost (out of the two methods explored above) for that individual benchmark with DRAM full row buffer write-back (which is the only available write-back policy for DRAM). We see that
independent of the method chosen, PCM performs worse across the board except for with the 436.cactusADM benchmark, likely due to the fact that it has particularly high number of dirty row buffer misses and that it usually only writes a small number of bytes to each row in a row buffer before it is evicted. The geometrical mean shows us that on average the power consumption increases by a factor of 4.6 for PCM writes. Something that is important to note, however, is that we only look at the energy cost for writing to the memory. I have not taken into account the energy cost of DRAM refresh, which is something that would affect the memory system’s total energy consumption and is something that PCM completely lacks thanks to its non-volatile property. While non-volatility may currently not make up for PCM’s higher write costs, the fraction out of DRAM’s total energy costs used for refreshing its cells is likely to grow with an increasing size of DRAM chips [2], making it possible that this property might make up for PCM’s higher write costs in the future.

6. Related Work
There has already been a lot of work in exploring the architectural design space of systems incorporating emerging NVM technologies. In “A Study of Application Performance with Non-Volatile Main Memory” [40] Yiying Zhang and Steven Swanson look at the performance of storage applications such as a file server, a web server, a NoSQL database, a relational database, and Memcached with non-volatile main memory using a hardware non-volatile main memory emulator (in contrast to this thesis’ software simulator). They observe that application performance would increase significantly if NVMM replaced SSDs and HDDs even without any application changes. They have also looked into the performance impact of the projected lower bandwidth and higher latency of upcoming technologies. They found that the performance degradation was less than 10 % compared to DRAM for most of the tested applications. They also found that it can be costly to make data persistent since it entails flushing the data from the processor’s volatile caches into the non-volatile memory.

In “NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories” [41] Matt Poremba and Yuan Xie present NVMain, which is an architectural-level simulator that can model both NVM and DRAM as main memories. In their paper, they discuss the specifics of their simulator, which go into details such as timing, power consumption and endurance. The simulator can model different numbers of memory banks,
ranks and types of buses and types of memory controllers. In their paper they also show data to prove the correctness of their simulator.

There have been a few papers reviewing the state of emerging non-volatile technologies and two such reviews are An Chen’s “A Review of Non-Volatile Memory (NVM) Technologies and applications” [25] and Sparsh Mittal and Jeffrey S. Vetter’s “A survey of Software Technologies for Using Non-Volatile Memories for Storage and Main Memory Systems” [5]. Both of these surveys have a broader view of the potential applications of NVM compared to this paper, but both look into the possibilities of main memory application. Both papers also identify the most important technologies as PCM, STT-RAM, and ReRAM, although Chen also mentions FeFet and Mittal and Vetter mentions Flash. Apart from describing the different technologies structure and characteristics, Chen also look at the performance of available test chips and compares the main advantages and challenges of each technology. In addition, Chen considers possible applications of these emerging NVMs. The main possibilities discussed are replacing existing memories, simplifying the memory-storage hierarchy, and enabling storage-class memory as well as applications in low-power computing and non-von-Neumann architectures. Chen also considers some more exotic applications outside memory space such as true random number generators and brain-inspired computing, among other things. Mittal and Vetter’s survey focuses on software techniques improving NVM performance when used as secondary storage and main memory. They put forth the belief that current NVM technologies and DRAM are not sufficiently high performing and reliable to be viable in a future where data will be ever more abundant. In order to make NVM technologies ready for such a future they say that innovations need to be made on multiple levels of abstraction, from device level to system level. On the device level they, among other things, mention 3D design as a something that might lead to smaller form factors and lower latencies. They also bring up software schemes that possibly could hide the larger write overhead of NVM technologies. While these NVM technologies are unlikely to be affected by soft errors caused by radiation, they may have soft errors caused by thermal noise and resistance drift, and this is something Mittal and Vetter point to as an area in need of further research. The lower endurance of NVM technologies need to be dealt with at the architectural level with write-minimization, wear-leveling and fault-tolerance. Additionally, at the system level they suggest, among other things, a unification of the address space as a possible way to further reduce overhead. Due to the wide scope of the survey there are many other techniques and ideas discussed as well.
7. Conclusions and Future Work
In this thesis I have explored the current state of emerging NVM technologies, especially with respect to how they relate to main memory. I have in more detail discussed STT-RAM, ReRAM and PCM which are three front runners in this group of technologies. Then I have answered three research questions relating to PCM.

The first question I answer is whether in the case of horizontally integrated PCM and DRAM it is useful to place the stack of an application on DRAM and the heap on PCM to improve energy costs and performance. I find that in the simulations performed, almost all of the cache misses are to heap addresses, due to most of the stack being kept in the cache. This result leads me to conclude that this mapping policy is not useful in cases where most of the stack is able to be kept in the cache. If one still insists on using this policy, the result is similar to just replacing DRAM with PCM, which leads to a decrease in performance and energy costs.

The second question I answer is what the effects on read latency are when replacing DRAM with PCM. I find an average increase in read latency of about 48 % but I note that this does not mean a 48 % increase in runtime, since read latency is only a subset of total runtime.

In the third and last question I compare the energy costs of two different write policies in PCM. The first one uses write-through for dirty bytes on byte granularity and the second one uses write-back of the full row buffer. I find that on average the first policy’s energy costs are less than a third of the second one’s. In five benchmarks, where the same row is written to repeatedly before being closed, the second policy had lower energy costs. I also compared the method with best results for each benchmark with DRAM using full row buffer write-back and find that on average PCM has about 4.6 times higher energy costs for write operations.

In further research, it could be interesting to explore less simplistic types of memory mapping for horizontally integrated PCM and DRAM. With regards to read latency it would be valuable to explore more ways in which to vary how row buffers are architected to minimize the probability of dirty row buffer misses, and doing so with more advanced simulators to achieve higher validity compared to this study. Looking into the energy costs for the entire system with a more sophisticated simulator, one that takes DRAM refresh and other more advanced memory features into account, would also be interesting.
8. References


