# 1 Review “Architecting Phase Change Memory as a Scalable DRAM Alternative” (1)

## Summary

This paper proposes to use phase change memory (PCM) instead of DRAM. The main claims are that PCM is more scalable then DRAM. The paper proposes an architecture that reduces the performance difference of energy and delay between DRAM and PCM from 1.6x and 2.2x to 1.2x and 1.0x. The paper does this with the following steps:

- The paper outlines the issue of DRAM scaling. Specifically that with smaller technology nodes the capacity of the cells decreases and their leakage current increases.
- The paper claims that with some simple architectural changes PCM can achieve a similar performance to DRAM while being more scalable as its memory is non-volatile.
- The paper explains the functioning of a PCM cell. The basic logic is that a cell is comprised of two electrodes separated by a resistor and phase change material. By injecting a current into the phase change material a phase change is caused which is the stored data.
- The paper discusses the issue of high delay time of PCM which boil down to approximately 4.4x DRAM read delay and 12.0x DRAM write delay. It also acknowledges that PCM incurs higher energy costs than DRAM. It proposes a novel buffer organization whereby using narrower but more buffers these shortfalls can be compensated.
- The paper provides simulated test results that show that PCM works at roughly the same delay times and energy consumption as DRAM. They also calculate hypothetical endurance values of the PCM.

## Strengths

The paper identifies the important issue of DRAM scaling and proposes a solution. With PCM it provides a possible alternative to DRAM and it shows that under certain conditions it can perform on a similar level to DRAM

## Weaknesses

The paper only provides simulated results which are way weaker than results from physical implementation. The paper estimates PCM lifetime with unverified back of the envelope calculations. These cannot be taken seriously as they are provided. Even in the best case PCM is barely as good as common, not high performance, DRAM.

## My POV

Time has clearly shown that PCM is not a widescale-viable replacement to DRAM. DRAM, while not perfect, proved surprisingly scalable compared to the predictions in this paper. The values provided in the paper seem somewhat detached from reality and it seems as if parameters and configurations were specifically chosen to present PCM in the best possible light.

## Takeaways

- PCM is a technology which can act as a replacement to DRAM but is not wide-scale viable.

## 2 Review "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives" (2)

### Summary

This paper provides a broad overview of errors in flash-memory-based solid-state drives(SSDs) as well as mitigation techniques. It also discusses the effect of future technology on these errors and their prevalence. The paper starts off by describing the state-of-the-art SSD architecture and describing how flash memory works to keep this review short, the description of the functioning of an SSD and flash memory will be omitted in the summary. The digression into similar errors in other memory technologies is also omitted for the same reason.

#### Error Characterization

- **P/E cycling errors:** Errors that occur due to failiure in erase operation or the writing operation to a cell. These errors increase over time and are a major symptom of wearout. These errors occur significantly more often in multi-level cells(MLCs) and triple-level cells(TLCs).
- **Program Errors:** Errors that occur when the flash array being read contains errors. And these values are then used to program new data. These stem mainly from two-step or foggy-fine programming, and copyback operations.
- **Cell-to-Cell Program Interference Errors:** Errors that occur due to parasitic capacitance coupling between nearby cells. This effect happens mainly if a neighbouring cell is written to.
- **Data Retention Errors:** Errors that occur due to charge leakage over time after a flash cell is programmed. These are the dominant source of flash memory errors.
- **Read Disturb Errors:** Errors that occur when reading of a cell causes the threshold voltages of other nearby cells to shift to a higher value.

#### Error Mitigation

**Shadow Program Sequencing** Shadow program sequencing tries to ensure that a fully-programmed wordline experiences interference minimally from cell-to-cell program interference. Shadow program sequencing assigns a unique page number to each page within a block. This allows two-step programming to write to pages in increasing order inside a block, such that a fully-programmed wordline experiences interference only from the most-significant bit(MSB) page programming of the wordline directly above it.

**Neighbour-Cell Assisted Error Correction** This mitigation technique works by using the data values stored in the adjacent wordline to determine a better set of read reference voltages for the wordline being read. This leads to a more accurate identification of the logical data value, as the data in the adjacent wordline was partially responsible for shifting the threshold voltage of the cells in the wordline being read.

#### Refresh Mechanisms

- **Remapping-Based Refresh:** This refresh works by periodically reading data from each valid flash block, correcting any errors, and remapping the data to a different physical location in order to prevent the data from accumulating too many retention errors.
- **In-Place Refresh:** Remapping incurs a cost due to the additional writes it causes and therefore the additional wearout. In-place refresh incrementally replenishes the lost charge of each page at its current location without remapping.
- **Read Reclaim to Reduce Read Disturb Errors:** This refresh works by remapping data in a block that has experienced a high number of reads to a new block.
- **Adaptive Refresh and Read Reclaim Mechanisms:** For the above mentioned refresh mechanisms the controller can either invoke them at fixed regular intervals or adapt the rate at which it invokes the mechanisms. By adapting the mechanisms based on the current conditions of the SSD, the controller can reduce the overhead of performing refresh or read reclaim.

**Read-Retry** This mitigation allows the read reference voltages to dynamically adjust to changes in distributions. This allows to adapt to changing threshold voltage distributions.

**Voltage Optimization** Many raw bit errors in NAND flash memory are affected by the various voltages used within the memory to enable reading of values. As a result, optimizing these voltages such that they minimize the total number of errors can greatly mitigate error counts. For this there are multiple approaches.

- Optimizing Read Reference Voltages Using Disparity Based Approximation and Sampling.
- Optimizing Pass-Through Voltage to Reduce Read Disturb Errors.

**Hot Data Management** Frequently-written pages are called write-hot pages. Analogously pages which are frequently-read are called read-hot pages. On the contrary pages with low access frequencies are called write-/read-cold pages. hot and cold pages have different refresh requirements and different error profiles. Therefore they need to be managed differently.

**Adaptive Error Mitigation Mechanisms** Adaptive error mitigation mechanisms are capable of adapting error tolerance capability to the error rate. This provides stronger error tolerance capability when the error rate is higher which improves flash lifetime.

- **Multi-Rate ECC:** This mechanism uses a weaker codeword - a codeword that uses fewer bits - when the SSD is relatively new and has less raw bit errors.
- **Dynamic Cell Levels:** A major error source in NAND flash memory is the overlapping of neighbouring voltage distributions. This issue is more prevalent in TLC than in MLC and again more prevelant in MLC than in single-level cells(SLCs). By dynamically adjusting the number of bits stored in the cells these errors can be mitigated.

### Error Correction and Data Recovery Techniques

**Error Correction Flow with Bose-Chaudhuri-Hocquenghem(BCH) Codes** The data is first corrected by the error-correction-code(ECC) engine by using BCH decoding on the raw data. This reports the total number of bit errors in the data. If the data cannot be corrected by the implemented BCH codes, many controllers invoke read-retry or read reference voltages to lower the raw bit error rate. BCH decoding is hard decoding, where the ECC engine can only use the hard bit value information.

**Error Correction Flow with LDPC Codes** LDPC decoding consists of three major steps. First, the SSD controller performs LDPC hard decoding, with the optimal read reference voltages. Second, if LDPC hard decoding cannot correct all of the errors, the controller uses LDPC soft decoding. Third, if LDPC soft decoding also cannot correct all of the errors, the controller invokes superpage-level parity.

**SSD Data Recovery** When the afformentioned correction mechanisms fail, then data loss can occur and the SSD is considered to have reached the end of its lifetime. To recover this data from the SSD, the controller can employ *Retention Failure Recovery* and *Read Disturb Recovery.*

### Emerging Reliability Issues for 3D NAND Flash

The paper states that current issues of 3D NAND flash is different from. But the paper states that with increasing scaling of 3D NAND flash memory it will encounter the same issue as planar 2D NAND flash memory. But issues like cell-to-cell interference will worsen as in a 3D layout each cell has more neighbours.

## Strengths

The paper provides a thorough review of errors in flash-based SSDs as well as the mitigation techniques thereof. The paper also provides a good introduction into the underlying concepts as well as the high level consequences of these issues.

## Weaknesses

I cannot list any concrete weaknesses of the paper.

## My POV

The paper is a very informative summary of prevalent issues in NAND flash SSDs. It really helped me understand these topics. While it does not present a large amount of own research, this is fine as that clearly is not the intention of the paper. The paper rather is a strong educational piece about NAND flash-based SSDs.

## Takeaways

- NAND flash-based SSDs are a very strong technology which comes with its own challenges and error sources. It is necessary to address these error sources to provide functioning computer systems.
- Future technology scaling and 3D stacked NAND flash systems will encounter the same issues and increase the prevalence of these error sources.

# 3 Review "Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning" (3)

## Summary

The paper addresses the issue of memory prefetching. The basic idea of memory prefetching is to load memory into a cache before the CPU needs it, thus reducing cache miss rates and therefore reducing the compute time by masking the high DRAM access latency. The paper proposes a reinforcement-learning(RL) based prefetcher called Pythia. The paper explains its functionality, assesses its performance on a set of recognized benchmarks, and compares it to other prefetchers. The paper does this in the following steps.

- The paper introduces Pythia as an RL-based prefetcher. The key idea is that RL allows the prefetcher to efficiently learn memory access patterns. Further RL enables the Pythia to learn online meaning that its policies can adapt to the workload independently.
- To train the RL model Pythia has a set of five rewards which it uses to learn any memory access patterns that might occur. These as well as the hyperparameters were tuned by successive experiments. The five rewards are:
  1. Accurate and Timely
  2. Accurate but Late
  3. Loss of Coverage
  4. Inaccurate
  5. No-Prefetch
- The paper claims that Pythia has a space and energy cost of 1.03% and 0.37% on a 4-core desktop class processor. This claim is based on an HDL implementation of Pythia compiled with Synopsys. The paper also evaluates Pythia with ChampSim and compares it to other state-of-the-art prefetchers. The paper shows that Pythia outperforms the other prefetchers on most of the traces and in the geometric mean performance over all traces.

## Strengths

The idea to use machine learning for prefetching is not new but the efficient RL implementation warrants acknowledgement. Pythia shows very strong best case performance and strong average performance.

## Weaknesses

Pythia shows clear performance degradation for some traces. These examples were not highlighted in the paper but my own investigation of Pythia has led me to this conclusion. The area and energy cost of 1.03% and 0.37% are significant as these values are not comparisons to other prefetchers but to the overall processor. This issue was insufficiently discussed in the paper.

## My POV

Pythia provides clear advantages. Mainly its high peak performance seems very promising. But the significant area and energy cost which are related to its complexity might prohibit implementation in a real setting, as Pythia does not provide constant performance. This is due to performance degradation for certain applications.

## Takeaways

- RL is a powerful tool that can lead to significant speedup through prefetching. But a more complex implementation always comes at an energy and area cost.
- Pythia shows promise but is insufficient to be used in an actual implementation as it does show performance degradation in some application traces.

# 4 Review “Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers” (4)

## Summary

The paper highlights the issue that more aggressive prefetchers achieve a higher peak performance in certain traces but have also very high performance loss in other traces while moderate prefetchers show a more balanced performance profile. The paper proposes a system to dynamically tune the prefetcher to exploit both the high peak performance of aggressive prefetchers while minimising performance loss for other traces. The paper does this in the following steps:

- The paper shows that aggressive prefetchers, i.e. prefetchers that prefetch a lot of memory, achieve way higher peak performance especially in applications that align with their prefetching strategy. But these same prefetchers lose a lot of performance in other applications as they clog the memory bandwidth and pollute the cache.
- The paper proposes to scale the prefetchers “aggressiveness” with the two parameters `prefetch distance` and `prefetch degree`. By tuning these parameters the amount of prefetched memory can be controlled. In other implementations these are hardcoded values.
- The paper introduces three metrics to evaluate the performance of the prefetcher.
  1. Prefetch Accuracy: This is the amount of useful prefetches per prefetch.
  2. Prefetch Lateness: This is the amount of late prefetches, prefetches that arrive after they are needed and thus still cause a cache miss, per prefetch.
  3. Prefetcher-Generated Cache Pollution: This is the amount of cache misses caused by the prefetcher per cache miss.
- The paper proposes a system where for a set time interval, in the implementations this is counted in cache accesses, these metrics are evaluated and based on the performance of the prefetcher the `prefetch distance` and `prefetch degree` get tuned dynamically.
- The paper also proposes an extension to the system where depending on the cache pollution the insertion in the LRU order of cache is dynamically tuned.
- The paper evaluates an implementation of the system in a stream prefetcher, a global-history-buffer-based delta correlation prefetcher, and a PC-based stride prefetcher. For all three implementations the paper demonstrates clear performance improvements over conservative and moderate versions over all traces. For the very aggressive strategies there are some cases where the dynamic aggression prefetcher performs worse but on average it also outperforms the aggressive prefetchers.

## Strengths

The paper provides a deep-dive into the concept of dynamic aggresion tuning in prefetchers. In the course of this it outlines strong fundamental concepts that allow prefetcher performance improvement without having to trade off the high peak performance of aggressive prefetching strategies vs. the good median performance of conservative prefetchers.

## Weaknesses

I do not see any clear weaknesses in this paper. The only complaint I have is the need for further research.

## My POV

The paper introduces clear and effective concepts to dynamically tune prefetcher performance. I do not have any concrete criticisms for this paper. While the strategies proposed in the paper aren’t state of the art anymore and surely can be improved through further research it nonetheless laid an important foundation.

## Takeaways

- Dynamic prefetcher tuning is essential. Modern prefetchers clearly have evolved beyond these basic metrics but the concepts introduced in this paper are still visible in modern systems.

## 5 Review “Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads” + Retrospecitve (5)

### Summary

#### Paper

The paper discusses the issue of cache miss mitigation via runahead execution. Runahead execution is a process where a runahead processor executes a part of an application before the CPU does to predict memory accesses and preload them into the cache to reduce cache misses. The paper shows that the runahead efficiency is heavily dependent on the runahead window. To increase runahead performance it proposes continuous runahead. The paper compares the results with other runahead implementations as well as other prefetchers. It does this in the following steps:

- The paper states that runahead in contrary to prefetchers have a very high accuracy. It also highlights that previous runahead implementations only run for short window lengths which decreases their effectiveness.
- The paper proposes the continuous runahead. The basic logic is to have a runahead processor which starts to operate on a cache miss, and runs for either a fixed but long window length or until another miss occurs. On a miss the relevant registers and instructions are transferred to the runahead processor. The runahead processor determines the dependence chain for the memory and repeatedly executes it to prefetch the memory.
- The paper discusses different dependence chain selection policies:
  1. PC-Based Policy
  2. Maximum-Misses Policy
  3. Stall Policy
- The paper provides an implementation of a continuous runahead engine(CRE) which is implemented as a CPU adjacent unit which is connected to the CPU’s registers.
- The paper evaluates the continuous runahead on both a single-core and quad-core system. It shows that for the single-core system it outperforms all prefetchers in the geometric mean but performs worse on specific applications. The paper also provides statistics for prefetcher + CRE systems which show the best performance. On the quad-core system the performance decreases and is heavily dependent on the policy to select which core to runahead on.
- The paper shows that a Round Robin policy shows the best performance. for the quad-core system. It also again shows that the prefetcher + CER systems give the best performance.

#### Retrospective

The retrospective gives a basic explanation of the algorithm as well as a lot of references to related work. It ends in a criticim of current academia and paper review standards.

### Strengths

The paper provides important insight into runahead computation. It clearly outlines the issues of small runahead windows and shows the benefits of increasing them. The paper gives a strong and performant implementation of a CRE which is even more performant when combined with a GHB prefetcher.

### Weaknesses

The paper understates the importance of the area and energy cost incurred by the CRE. The stated 10% of a core are significant and cannot be ignored. The paper also states that this is less than a GHB prefetcher but the discussion on area and power usage falls short. It would be nice to see a clear comparison of the different systems (prefetcher, CRE, prefetcher + CRE), with their respective area and energy costs in both the single-core and the quad-core system. Without that data I cannot fairly assess the viability of most efficient provided solution which combines both a CRE and a GHB prefetcher.

## My POV

Runahead execution is a very elegant solution to the memory prefetching problem. The combination of CRE and GHB provides a very performant solution. Due to the omission of a detailed area and power analysis in the paper I cannot draw a clear conclusion on the viability of the system as proposed in the paper. The paper makes a strong case on a theoretical level by introducing and showing the functioning of continuous runahead which inspired further research.

## Takeaways

- Continuous Runahead is a strong solution to the memory prefetching problem but it also comes at significant area and energy cost.