Exploiting Row-Level Temporal Locality in DRAM to Reduce the Memory Access Latency

Hasan Hassan\textsuperscript{1,2,3} Gennady Pekhimenko\textsuperscript{4,2} Nandita Vijaykumar\textsuperscript{2} Vivek Seshadri\textsuperscript{5,2} Donghyuk Lee\textsuperscript{6,2} Oguz Ergin\textsuperscript{3} Onur Mutlu\textsuperscript{1,2}

\textsuperscript{1}ETH Zürich \textsuperscript{2}Carnegie Mellon University \textsuperscript{3}TOBB University of Economics & Technology \textsuperscript{4}University of Toronto \textsuperscript{5}Microsoft Research India \textsuperscript{6}NVIDIA Research

This paper summarizes the idea of ChargeCache, which was published in HPCA 2016 [51], and examines the work’s significance and future potential. DRAM latency continues to be a critical bottleneck for system performance. In this work, we develop a low-cost mechanism, called ChargeCache, that enables faster access to recently-accessed rows in DRAM, with no modifications to DRAM chips. Our mechanism is based on the key observation that a recently-accessed row has more charge and thus the following access to the same row can be performed faster. To exploit this observation, we propose to track the addresses of recently-accessed rows in a table in the memory controller. If a later DRAM request hits in that table, the memory controller uses lower timing parameters, leading to reduced DRAM latency. Row addresses are removed from the table after a specified duration to ensure rows that have leaked too much charge are not accessed with lower latency. We evaluate ChargeCache on a wide variety of workloads and show that it provides significant performance and energy benefits for both single-core and multi-core systems.

1. Problem: DRAM Latency

DRAM technology is commonly used as the main memory of modern computer systems. This is because DRAM is at a more favorable point in the trade-off spectrum of density (cost-per-bit) and access latency compared to other technologies like SRAM or flash. However, commodity DRAM devices are heavily optimized to maximize cost-per-bit. In fact, the latency of commodity DRAM has not reduced significantly in the past two decades [23, 25, 80, 83, 84, 108].

The latency of DRAM is heavily dependent on the design of the DRAM chip architecture, specifically the length of a wire called bitline. A DRAM chip consists of millions of DRAM cells. Each cell is composed of a transistor-capacitor pair. To access data from a cell, DRAM uses a component called sense amplifier. Each cell is connected to a sense amplifier using a bitline. To amortize the large cost of the sense amplifier, hundreds of DRAM cells are connected to the same bitline [84]. A longer bitline leads to higher resistance and parasitic capacitance on the path between a DRAM cell and the sense amplifier. As a result, longer bitlines result in higher DRAM access latency [80, 83, 84, 136].

To mitigate the negative effects of long DRAM access latency, existing systems rely on several major approaches. First, they employ large on-chip caches to exploit the temporal and spatial locality of memory accesses. However, cache capacity is limited by chip area. Even caches as large as tens of megabytes may not be effective for some applications due to very large working sets and memory access characteristics that are not amenable to caching [61, 90, 113, 117, 118]. Second, systems employ aggressive prefetching techniques to preload data from memory before it is needed [5, 28, 138]. However, prefetching is inefficient for many irregular access patterns and it increases the bandwidth requirements and interference in the memory system [36, 38, 39, 76, 131, 138]. Third, systems employ multithreading [86, 134, 145]. However, this approach increases contention in the memory system [32, 37, 98, 106] and does not aid single-thread performance [62, 144]. Fourth, systems exploit memory level parallelism [31, 47, 104, 106, 107]. The DRAM architecture provides various levels of parallelism that can be exploited to simultaneously process multiple memory requests generated by modern processor architectures [78, 107, 115, 146]. While prior works [31, 33, 60, 78, 106, 112] propose techniques to better utilize the available parallelism, the benefits of these techniques are limited due to 1) address dependencies between instructions in the programs [6, 40, 103], and 2) resource conflicts in the memory subsystem [73, 120]. Unfortunately, none of these four approaches fundamentally reduce memory latency at its source and the DRAM latency continues to be a performance bottleneck in many systems.

2. Existing Techniques That Reduce DRAM Latency

DRAM latency can be reduced using several techniques, all of which have their own specific shortcomings. One simple approach to reduce DRAM latency is to use shorter bitlines. In fact, some specialized DRAM chips [48, 96, 125] offer lower latency by using shorter bitlines compared to commodity DRAM chips. Unfortunately, such chips come at a significantly higher cost than chips that use long bitlines, as they reduce the overall density of the device because they require more sense amplifiers, which occupy significant area [84]. Therefore, such specialized chips are usually not desirable for systems that require high memory capacity [29]. Prior works have proposed several heterogeneous DRAM architectures (e.g., segmented bitlines [84], asymmetric bank organi-
zations [136], mechanisms that exploit the inherent latency variation across cells [25, 82]) that divide DRAM into two regions: one with low latency, and another with slightly higher latency. Such schemes propose to map frequently accessed data to the low-latency region, thereby achieving lower average memory access latency. However, such schemes might require 1) non-negligible changes to the cost-sensitive DRAM design, 2) techniques to create or identify low-latency regions in DRAM, and/or 3) mechanisms to identify, map, and migrate frequently-accessed data to low-latency regions. As a result, even though they reduce the latency for some portions of the DRAM chip, they may not be easy to adopt.

3. Key Observations

In our HPCA 2016 paper [51], we make two major observations that motivate a new mechanism for reducing DRAM latency.

**Charge Variation.** The amount of charge in the DRAM cells of a row determines the required latency for a DRAM access to that row. If the amount of charge in the cell is low, the sense amplifier completes its operation in longer time. Therefore, DRAM access latency increases. A DRAM cell loses its charge over time and the charge is replenished by a refresh operation or an access to the row. The access latency of a cell whose charge has been replenished recently can thus be significantly lower than the access latency of a cell that has less charge. Our SPICE simulations show that the first read/write command can be issued 44% faster to a highly-charged DRAM row compared to a row with less charge (see Section 6.2 and our HPCA 2016 paper [51]).

**Row-Level Temporal Locality.** We find that, mainly due to DRAM bank conflicts [73, 120], many applications tend to access rows that were recently closed (i.e., closed within a very short time interval). We refer to this form of temporal locality where certain rows are frequently closed and re-opened as Row-Level Temporal Locality (RLTL). An important outcome of this observation is that a DRAM row remains in a highly-charged state when accessed for the second time within a short interval after the prior access. This is because accessing the DRAM row inherently replenishes the charge within the DRAM cells (just like a refresh operation does) [26, 46, 87, 88, 109, 133].

We define t-RLTL of an application for a given time interval t as the fraction of row activations in which the activation occurs within the time interval t after a previous precharge to the same row. Figure 1 shows the average RLTL for single-core and eight-core workloads with five different time intervals (from 0.125 ms to 32 ms). Our detailed experimental methodology is described in Section 5 of our HPCA 2016 paper [51]. For single-core workloads, the average 1 ms-RLTL is 83%. In other words, 83% of all the row activations occur within 1 ms after the same row was previously precharged. Due to the additional bank conflicts incurred as the number of workloads executing increases, for eight-core workloads, the average 1 ms-RLTL is 89%, significantly higher than that for the single-core workloads. These results show that RLTL of both single-core and eight-core workloads is significantly high even for small values of t, motivating us to exploit RLTL (i.e., row-level temporal locality) to detect highly-charged DRAM rows.1

Note that a major reason for the high row-level temporal locality is the occurrence of bank conflicts in the DRAM subsystem. We find that, due to the bank conflicts, a row is likely to be requested again soon after it is precharged due to an intervening request to the same bank.

![Figure 1: Average row-level temporal locality (RLTL) for 22 single-core and 20 eight-core workloads.](image)

4. Our Goal

We observe that *many* applications exhibit high row-level temporal locality. In other words, for many applications, a significant fraction of the row activations occur within a small interval after the corresponding rows are precharged. As a result, such row activations can be served with lower activation latency than specified by the DRAM standard. Our goal in this work is to exploit this observation to reduce the effective DRAM access latency by tracking recently-accessed DRAM rows in the memory controller and reducing the latency for their next access(es). To this end, we propose an efficient mechanism, ChargeCache, which we describe in the next section.

5. Solution: ChargeCache

ChargeCache is based on three observations: 1) a row whose cells’ charge has been recently replenished can be accessed with lower activation latency, 2) activating a row replenishes the charge on the cells of that row and the cells start leaking only after the following precharge command, and 3) many applications exhibit high row-level temporal locality, i.e., recently-activated rows are more likely to be activated again. Based on these observations, ChargeCache tracks rows that are recently activated, and serves near-future activates to such rows with lower latency by lowering the DRAM timing parameters for such activations.

As we show in Figure 2, ChargeCache adds a small table (structured as a cache), called High-Charged Row Address Cache (HCRAC), to the memory controller that tracks the addresses of recently-accessed DRAM rows, i.e., highly-charged

---

1 For a more detailed study of row-level temporal locality, please see Section 3 of our HPCA 2016 paper [51].
Highly-Charged Row Address Cache (HCRAC) Invalidation Interval Counter (IIC) Entry Counter (EC) Invalidate 3 [ACT] Lookup

Per-Bank Row State

Invalidate Interval Counter (IIC)

Entry Counter (EC)

Per-Bank Timing State

Figure 2: Components of the ChargeCache Mechanism. Reproduced from [51].

We name our mechanism ChargeCache, as it provides a cache-like benefit, i.e., latency reduction based on a locality property (i.e., RLTL), and does so by taking advantage of the charge level stored in a recently-activated row. The mechanism could potentially be used with current and emerging DRAM-based memories where the stored charge level leads to different access latencies. We release the source code of ChargeCache for two different versions of Ramulator [74, 122, 123] to enable future research to build upon our ideas.

6. Experimental Evaluation

In this section, we first explain our experimental methodology. Later, we quantitatively analyze the system performance improvement and DRAM energy savings that ChargeCache provides.

6.1. Methodology

We use circuit-level SPICE simulations to evaluate the DRAM latency reduction that can be achieved when accessing a highly-charged DRAM row. In Section 6.2, we show the reduction in two DRAM timing parameters, \( t_{RCD} \) and \( t_{RAS} \), that are affected by high charge amount stored in a DRAM cell.

To evaluate the performance of ChargeCache, we use a cycle-accurate DRAM simulator, Ramulator [74,122], in CPU-trace-driven mode. CPU traces are collected using a Pin-tool [91]. Table 1 lists the configuration of the evaluated systems. We implement the HCRAC table, which ChargeCache uses to store the addresses of recently accessed DRAM rows, similarly to a 2-way associative cache that uses the LRU policy.

Table 1: Simulated system configuration. Reproduced from [51].

|               |                                                      |
|---------------|------------------------------------------------------|
| Processor     | 1-8 cores, 4GHz clock frequency, 3-wide issue, 8 MSHRs/core, 128-entry instruction window |
| Last-level    | 64B cache-line, 16-way associative, 4MB cache size   |
| Cache         | 64-entry read/write request queues, FR-FCFS scheduling policy [121, 153], open/closed row policy [71, 72] for single/multi core |
| DRAM          | DDR3-1600 [97], 800MHz bus frequency, 1/2 channels, 1 rank/channel, 8 banks/rank, 64K rows/bank, 8KB row-buffer size, \( t_{RCD}/t_{RAS} \) 11/28 cycles |
| ChargeCache   | 128-entry (672 bytes)/core, 2-way associativity, LRU replacement policy, 1ms caching duration, \( t_{RCD}/t_{RAS} \) reduction 4/8 cycles |

For area, power, and energy measurements, we modify McPAT [85] to implement ChargeCache using 22nm process technology. We use DRAMPower [22] to obtain power/energy results for the off-chip main memory subsystem. We feed DRAMPower with DRAM command traces obtained from our simulations using Ramulator.

We run 22 workloads from the SPEC CPU2006 [137], TPC [147], and STREAM [94] benchmark suites. We use SimPoint [50] to obtain traces from representative phases of each application. For single-core evaluations, unless stated otherwise, we run each workload for 1 billion instructions. For multi-core evaluations, we use 20 multiprogrammed workloads by assigning a randomly-chosen application to each core. We evaluate each configuration with its best-performing row-buffer management policy. Specifically, we use the open-row policy for single-core and closed-row policy for multi-core.
core configurations. We simulate the benchmarks until each core executes at least 1 billion instructions. For both single- and multi-core configurations, we first warm up the caches and ChargeCache by fast-forwarding 200 million cycles.

We measure performance improvement for single-core workloads using the Instructions per Cycle (IPC) metric. We measure multi-core performance using the weighted speedup [135] metric. Prior work has shown that weighted speedup is a measure of system-level job throughput [42].

6.2. Reduction in DRAM Timing Parameters

We evaluate the potential reduction in tRCD and tRAS for ChargeCache using circuit-level SPICE simulations. We implement the DRAM sense amplifier circuit using 55nm DDR3 model parameters [119] and PTM low-power transistor models [3, 152]. Figure 3 plots the variation in bitline voltage level during cell activation for different initial charge amounts of the cell.

![Figure 3: Effect of initial cell charge on bitline voltage. Reproduced from [51].](image)

Depending on the initial charge (i.e., voltage level) of the cell, the bitline voltage increases at different speeds. When the cell is fully-charged, the sense amplifier is able to drive the bitline voltage to the ready-to-access voltage level in only 10ns. However, a partially-charged cell (i.e., one that has not been accessed for 64ms) brings the bitline voltage up slower. Specifically, the bitline connected to such a partially-charged cell reaches the ready-to-access voltage level in 14.5ns. Since DRAM timing parameters are dictated by this worst-case partially-charged state right before the refresh interval, we can achieve a 4.5ns reduction in tRCD for a fully-charged cell. Similarly, the charge of the cell capacitor is restored at different times depending on the initial voltage of the cell. For a fully-charged cell, this results in a 9.6ns reduction in tRAS.

In practice, we expect DRAM manufacturers to identify the lowered timing constraints for different caching durations. Today, DRAM manufacturers test each DRAM chip to determine if it meets the timing specifications. Similarly, we expect the manufacturers would also test each chip to determine if it meets the ChargeCache timing constraints.

6.3. Results

We experimentally evaluate the following mechanisms: 1) ChargeCache, 2) NUAT [133], which accesses only rows that are recently-refreshed at lower latency than the DRAM standard, 3) ChargeCache + NUAT, which is a combination of ChargeCache and NUAT [133] mechanisms, and 4) Low-Latency DRAM (LL-DRAM) [96], which is an idealized comparison point where we assume all rows in DRAM can be accessed with low latency, compared to our baseline DDR3-1600 memory, at any time, irrespective of when they are accessed or refreshed.

We compare the performance of our mechanism against the most closely related previous work, NUAT [133]. The key idea of NUAT is to access recently-refreshed rows at low latency, because these rows are already highly-charged. Thus, NUAT does not use low latency for rows that are recently-accessed, and hence it does not exploit the RLTL (Row-Level Temporal Locality) present in many applications.

Figure 4 shows the performance of single-core and eight-core workloads. The figure also includes the number of row misses per kilo-cycles (RMPKC) to show row activation intensity, which provides insight into the RLTL of the workload.

**Single-Core Performance:** Figure 4a shows the performance improvement over the baseline system for single-core workloads. These workloads are sorted in ascending order of RMPKC. ChargeCache achieves up to 9.3% (an average of 2.1%) speedup. Our mechanism outperforms NUAT and achieves a speedup close to LL-DRAM with a few exceptions. Applications that have a wide gap in performance between ChargeCache and LL-DRAM (e.g., mcf, omnetpp) access a large number of DRAM rows and exhibit high row-reuse distance [63]. A high row-reuse distance indicates that there is a large number of accesses to other rows between two accesses to the same row. Due to this reason, ChargeCache cannot retain the addresses of highly-charged rows until the next access to that row. Increasing the number of ChargeCache entries or employing cache management policies aware of reuse distance or thrashing [35, 117, 130, 148] may improve the performance of ChargeCache for such applications. We leave the evaluation of these methods for future work. We conclude that ChargeCache significantly reduces execution time for most high-RMPKC workloads and outperforms NUAT for all but few workloads.

**Eight-Core Performance:** Figure 4b shows the speedup on eight-core multiprogrammed workloads. On average, ChargeCache and NUAT improve performance by 8.6% and 2.5%, respectively. Employing ChargeCache in combination with NUAT achieves a 9.6% speedup, which is only 3.8% less than the improvement obtained using LL-DRAM. Although the multiprogrammed workloads are composed of the same applications as in single-core evaluations, we observe much higher performance improvements for the eight-core workloads. The reason is twofold. First, since multiple cores share a limited capacity LLC, simultaneously running applications compete...
Overall, ChargeCache improves performance by up to 11.3% (8.1%) and on average 8.6% (2.1%) for eight-core (single-core) workloads. It outperforms NUAT for most of the applications. Using NUAT in combination with ChargeCache improves chsystem performance even further.

6.4. Impact on DRAM Energy

ChargeCache incurs negligible area and power overheads (see Section 6.5). Because it reduces execution time with negligible overhead, it leads to significant energy savings. Even though ChargeCache increases the energy efficiency of the entire system, we quantitatively evaluate the energy savings only for the DRAM subsystem since Ramulator [74] currently does not have a detailed CPU model. Figure 5 shows the average and maximum DRAM energy savings for single-core and eight-core workloads. ChargeCache reduces energy consumption by an average of 7.9% (1.8%), and by up to 14.1% (6.9%), for eight-core (single-core) workloads. We conclude that ChargeCache is effective at improving the energy efficiency of the DRAM subsystem, as well as the entire system.

6.5. Area and Power Consumption Overhead

HCRAC (Highly-Charged Row Address Cache) is the most area/power demanding component of ChargeCache. As we replicate HCRAC on a per-core and per-memory channel basis, the total area and power overhead ChargeCache introduces depends on the number of cores and memory channels. The total storage requirement is given by Equation 1, where \(C\) are \(MC\) are the number of cores and memory channels, respectively. \(LRU\text{bits}\) depends on HCRAC associativity. \(Entry\text{Size}\) is calculated using Equation 2, where \(R\), \(B\), and \(Ro\) are the number of ranks, banks, and rows in DRAM, respectively.

\[
\text{Storage\text{bits}} = C \times MC \times \text{Entries} \times (\text{Entry\text{Size}\text{bits}} + \text{LRU\text{bits}}) \quad (1)
\]

\[
\text{Entry\text{Size}\text{bits}} = \log_2(R) + \log_2(B) + \log_2(Ro) + 1 \quad (2)
\]

Area. Our eight-core configuration has two memory channels. This introduces a total of 5376 bytes in storage require-
ment for a 128-entry HCRAC, corresponding to an area of 0.022 mm². This overhead is only 0.24% of the 4MB LLC.

**Power Consumption.** HCRAC is accessed on every *activate* and *precharge* command issued by the memory controller. On an *activate* command, HCRAC is searched for the corresponding row address. On a *precharge* command, the address of the precharged row is inserted into HCRAC. HCRAC entries are periodically invalidated to ensure they do not exceed a specified *caching duration*. These three operations increase dynamic power consumption in the memory controller, and the HCRAC storage increases static power consumption. Our analysis indicates that ChargeCache consumes 0.149 mW on average. This is only 0.23% of the average power consumption of the entire 4MB LLC. Note that we include the effect of this additional power consumption in our DRAM energy evaluations in Section 6.4. We conclude that ChargeCache incurs almost negligible chip area and power consumption overheads.

### 6.6. Other Results

We also evaluate and assess the sensitivity of ChargeCache benefits to ChargeCache capacity, caching duration, and temperature in Sections 6.4 and 7.1 of our HPCA 2016 paper [51].

### 7. Related Work

To our knowledge, this paper is the first to (i) show that applications typically exhibit significant *Row-level Temporal Locality (RLTL)* and (ii) exploit this locality to improve system performance by reducing the latency of requests to recently-accessed memory rows.

We have already (in Section 6.3) qualitatively and quantitatively compared ChargeCache to NUAT [133], which reduces access latency to only recently-refreshed rows. We have also shown that ChargeCache provides significantly higher average latency reduction than NUAT because RLTL is usually high, whereas the fraction of accesses to rows that are recently-refreshed is typically low (see Section 3 in our HPCA 2016 paper [51]).

Other previous works propose techniques to reduce performance degradation caused by long DRAM latencies. They focus on 1) enhancing the DRAM, 2) exploiting variations in manufacturing process and operating conditions, 3) developing various memory scheduling policies. We briefly summarize how ChargeCache differs from these works.

**Enhancing DRAM Architecture.** Lee at al. propose Tiered-Latency DRAM (TL-DRAM) [84], which divides each subarray into near and far segments using isolation transistors. With TL-DRAM, the memory controller accesses the near segment with lower latency since the isolation transistor reduces the bitline capacitance in that segment. Our mechanism could be implemented on top of TL-DRAM to reduce the access latency for both the near and far segment. Kim et al. propose SALP, which unlocks parallelism between subarrays at low cost, by modifying the DRAM chip to enable pipelined access to subarrays [73]. The goal of SALP is to reduce the impact of bank conflicts by providing more parallelism and thereby reducing the latency of bank-conflict accesses. O et al. [110] propose a DRAM architecture where sense amplifiers are decoupled from bitlines to mitigate precharge latency. Choi et al. [30] propose to utilize multiple DRAM cells to store a single bit when sufficient DRAM capacity is available. By using multiple cells, they reduce activation, precharge and refresh latencies. Other works [24, 26, 49, 79, 126, 127, 128, 129, 136, 151] also propose new DRAM architectures to lower DRAM latency for various types of operations and accesses.

**Processing-in-memory (PIM) architectures** [1, 2, 8, 9, 34, 41, 44, 53, 54, 65, 69, 75, 111, 116, 127, 128, 129, 132, 139] using 3D-stacked memory [56, 59, 81, 89] reduce the *observed latency, from the perspective of the processor*, by moving some computation operations closer to DRAM. 3D-stacked memories are well suited for processing-in-memory due to their inclusion of a logic layer, which allows for the efficient implementation of CMOS logic in DRAM and offers high bandwidth to the DRAM layers. However, PIM architectures do not fundamentally reduce the access latency of the DRAM device, which ChargeCache does (for certain access patterns).

Unlike ChargeCache, a large number of these works require changes to the DRAM architecture itself. The approaches taken by these works are largely orthogonal to the ChargeCache approach and ChargeCache could be implemented together with any of these mechanisms to further reduce the DRAM latency.

**Exploiting Process and Operating Condition Variations.** Recent studies [21, 25, 27, 82, 83] propose methods to reduce the safety margins of the DRAM timing parameters when operating conditions are appropriate (i.e., not worst-case). Unlike these works, ChargeCache is largely independent of operating conditions like temperature, as discussed in Section 8.3, and is orthogonal to these latency reduction mechanisms.

**Memory Request Scheduling Policies.** Memory request scheduling policies (e.g., [4, 45, 55, 57, 66, 71, 72, 77, 99, 100, 105, 106, 121, 140, 141, 142, 143, 149, 153]) reduce the average DRAM access latency by improving DRAM parallelism, row buffer locality, and fairness in especially multi-core and heterogeneous systems. ChargeCache can be employed in conjunction with the scheduling policy that best suits the application and the underlying architecture.

### 8. Significance

Main memory latency has a critical impact on system performance [101]. Our work proposes a new low-cost mechanism to reduce DRAM latency, *without* any modifications to the existing DRAM chip architecture. In this section, we discuss the significance of our work by describing its novelty and expected long-term impact.
8.1. Novelty

ChargeCache reduces average DRAM latency by exploiting a type of DRAM access locality, Row-Level Temporal Locality (RLTL), that commonly exists in workloads due to the presence of DRAM bank conflicts. Our work is the first to observe and formally define RLTL and exploit it to reduce DRAM latency by designing a new mechanism that takes advantage of RLTL and the fact that a DRAM row gets inherently refreshed on access. Our mechanism does not require any changes to the existing DRAM array structure of the DRAM chips and can be easily implemented on top of any DRAM standard with negligible overhead in the memory controller logic.

8.2. Applicability to Emerging DRAM Standards

ChargeCache is applicable to any memory technology where cells are volatile (leak charge over time) and the charge variation due to charge leakage has impact on access latency. ChargeCache can be used with a large set of standards derived from DDR (DDRx, GDDRx, LPDDRx, etc.) [74] in a manner similar to the mechanism described in this work, without modifying the DRAM architecture. Using ChargeCache with 3D-stacked memories [81,89] such as Wide I/O, HBM, and HMC [74] is also straightforward. The difference is that, for the technologies that implement the memory controller in the logic layer, the DRAM controller, and hence ChargeCache, can be easily implemented in the logic layer of the 3D-stacked memory chip instead of the processor chip.

We also believe that the key idea of ChargeCache is not limited to DRAM, and can potentially be applied to other memory technologies that store information in form of electrical charge, such as NAND flash memory [10,11,12,13,14,15,16,17,18,19,92,93].

8.3. Long-Term Impact

8.3.1. Reducing DRAM Latency. During the last several decades, DRAM capacity increased significantly by shrinking the feature size of the transistors. Similarly, more efficient DRAM standards enabled memories with high bandwidth. The new 3D-stacking technology offers even higher bandwidth by incorporating DRAM and the logic layer on the same chip in a 3D-stacked manner. However, none of these advances lead to large improvements in the row access latency of the DRAM arrays. Hence, DRAM latency is already a critical bottleneck for system performance. Our work alleviates the DRAM latency problem with no overhead to the area-optimized DRAM chip, which is difficult to change, and with low overhead to the memory controller.

8.3.2. Row-Level Temporal Locality. Our paper is the first work to observe row-level temporal locality (RLTL). Note that RLTL is different from Row-Reuse Distance [63] that a prior work studies. Row-Reuse Distance is a metric indicating the number of accesses between two consecutive accesses to the same row. On the other hand, RLTL indicates the time between two consecutive accesses to the same row. A row locality metric that includes time is important since charge leakage in DRAM is a function of time. In this work, we exploit RLTL to reduce DRAM latency. However, RLTL can also potentially be used to discover new techniques to improve different aspects of DRAM, such as reliability [70,95,101,102] and bandwidth.

8.3.3. Importance for Future Systems. We believe the latency reduction mechanism of ChargeCache will become more important in future systems for four reasons. First, DRAM latency will become a much bigger bottleneck, as applications will become more data-intensive [101,108]. Higher demand for data will result in more bank conflicts, as the number of DRAM banks is not scaling as fast as data intensity. Such applications will also have fast data access requirements, which will increase their sensitivity to the memory access latency [43,64,101,108,150]. As bank conflicts increase and accesses become more latency-critical, the benefits of ChargeCache will increase, as there will be higher RLTL, which ChargeCache can exploit to provide higher performance improvement.

Second, ChargeCache is likely to remain much more competitive than other state-of-the-art latency reduction techniques for the 3D-stacked memories of the future. These memories will likely operate at higher temperatures compared to conventional DRAM chips. The charge leakage rate of DRAM cells approximately doubles for every $10^\circ C$ increase in temperature [67,83,87,114]. This observation can be exploited to lower the DRAM latency when operating at low temperatures. A previous study, Adaptive-Latency DRAM (AL-DRAM) [83], proposes a mechanism to improve system performance by reducing the DRAM timing parameters at low operating temperature. AL-DRAM is based on the premise that DRAM typically does not operate at temperatures close to the worst-case temperature (85\degree C) even when it is heavily accessed. However, new 3D-stacked DRAM technologies such as HMC, HBM, WideIO may operate at significantly higher temperatures due to tight integration of multiple stack layers [7]. Therefore, state-of-the-art and compelling dynamic latency scaling techniques such as AL-DRAM may be less useful in these scenarios. In contrast to AL-DRAM, ChargeCache is not based on the charge difference that occurs due to temperature dependence. Rather, we exploit the high level of charge in recently-precharged rows to reduce timing parameters during later accesses to such rows. After conducting tests to determine the possible latency reduction in accessing highly-charged rows (for ChargeCache hits) at worst-case temperatures, we show that ChargeCache can be employed independently of the operating temperature (see Section 7.1 in our HPCA 2016 paper [51]).

Third, ChargeCache is complementary to other temperature-based and structural DRAM latency reduction techniques [24,25,73,82,84,96,133,136]. ChargeCache can easily be used in conjunction with any of these techniques.
Fourth, ChargeCache is a low-cost mechanism, which does not require any changes to the existing DRAM chips, and requires only small changes to the memory controller. The low cost makes the adoption of ChargeCache more feasible in future systems than other proposed mechanisms, as these systems will be bottlenecked by power consumption, and thus by complexity [108].

Overall, we believe that ChargeCache will help to significantly reduce the memory access latency in future systems. To this end, to aid future research, we have released the source code of our ChargeCache simulator [123, 124] as part of our Ramulator releases [122, 123].

9. Conclusion

We introduce ChargeCache, a new, low-overhead mechanism that dynamically reduces the DRAM timing parameters for recently-accessed DRAM rows. ChargeCache exploits two key observations that we demonstrate in this work: 1) a recently-accessed DRAM row has cells with high amounts of charge and thus it can be accessed faster, and 2) many applications repeatedly access rows that are recently-accessed, due to bank conflicts.

Our extensive evaluations of ChargeCache on both single-core and multi-core systems show that it provides significant performance benefit and DRAM energy reduction at very modest hardware overhead. ChargeCache requires no modifications to the existing DRAM chips and occupies only a small area on the memory controller.

We conclude that ChargeCache is a simple yet efficient mechanism to dynamically reduce DRAM latency, which significantly improves both the performance and energy efficiency of modern systems. We hope that our observation of the phenomenon of row-level temporal locality and its simple exploitation to reduce DRAM latency inspires others to work to develop other new techniques to improve memory subsystem characteristics like performance, efficiency, and reliability.

Acknowledgments

We thank Saugata Ghose for his dedicated effort in the preparation of this article. We thank the reviewers and the SAFARI group members for their feedback. We acknowledge the generous support of Google, Intel, NVIDIA, Samsung, and VMware. This work is supported in part by NSF grants 1212962, 1320531, and 1409723, the Intel Science and Technology Center for Cloud Computing, and the Semiconductor Research Corporation.

References

[1] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Cho, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in ISCA, 2015.
[2] J. Ahn, S. Yoo, O. Mutlu, and K. Cho, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in ISCA, 2015.
[3] Arizona State Univ., NIMO Group, "Predictive Technology Model," http://ptm.asu.edu/.
[4] R. Ausavarungnirun, K. K-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems," in ISCA, 2012.
[5] J.-L. Baer and T.-F. Chen, "An Effective on-Chip Preloading Scheme to Reduce Data Access Penalty," in ICS, 1991.
[6] M. Bekerman, S. Jourdan, F. Harou, G. Kirshenbaum, L. Rappoport, A. Yoaz, and U. Weiser, "Correlated Load-Address Predictors," in ISCA, 1999.
[7] B. Black et al., "Die Stacking (3D) Microarchitecture," in MICRO, 2006.
[8] A. Boroumand et al., "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in ASPLOS, 2010.
[9] A. Boroumand, S. Ghose, B. Lucia, K. Hsieh, K. Malladi, H. Zheng, and O. Mutlu, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," IEEE CAL, 2017.
[10] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives," arXiv:1706.08642 [cs.AR], 2017.
[11] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," arXiv:1711.11427 [cs.AR], 2017.
[12] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in ISCA, 2017.
[13] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in DATE, 2012.
[14] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, "Neighboring Cell Assisted Error Correction in MLC NAND Flash Memories," in SIGMETRICS, 2014.
[15] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," Proc. IEEE, 2017.
[16] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture," in ISCA, 2005.
[17] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wein, and K. Goossens, "Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization," in DATE, 2014.
[18] K. Chandrasekar, C. Weis, B. Akesson, N. Wein, and K. Goossens, "Towards Variation-Aware System-Level Power Estimation of DRAMs: An Empirical Approach," in DAC, 2013.
[19] K. K. Chang, "Understanding and Improving the Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon Univ., 2017.
[20] K. K. Chang et al., "Low-Cost Inter-Linked Subarrays (LBA): Enabling Fast Inter-Subarray Data Movement in DRAM," in HPCA, 2016.
[21] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in SIGMETRICS, 2016.
[22] K. K. Chang, D. Lee, Z. Chihi, A. R. Alamdeeen, C. Wiltkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in HPCA, 2014.
[23] K. K. Chang, A. G. Yläölä, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O’Connor, H. Hassan, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," SIGMETRICS, 2017.
[24] M. J. Charney and T. R. Puzak, "Prefetching and Memory System Behavior of the SPEC95 Benchmark Suite," IBM JRD, 1997.
[25] M. Chaurasia and T. R. Pozuk, "Prefetching and Memory System Behavior of the SPEC95 Benchmark Suite," in IBM JRD, 1997.
[26] N. Chatterjee, M. Shevgoor, R. Balasubramonian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer, "Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access," in MICRO, 2012.
[27] J. Choi, W. Shin, J. Jang, S. Joo, Y. Kwon, Y. Moon, and L-S. Kim, "Multiple Clone Row DRAM: A Low Latency and Area Optimized DRAM," in ISCA, 2013.
[28] K. Choi, B. Fabi, and S. Abraham, "Microarchitecture Optimizations for Exploiting Memory-Level Parallelism," in ISCA, 2002.
[29] M. J. Charney and T. R. Puzak, "Prefetching and Memory System Behavior of the SPEC95 Benchmark Suite," IBM JRD, 1997.
[30] N. Chatterjee, M. Shevgoor, R. Balasubramonian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer, "Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access," in MICRO, 2012.
[31] J. Choi, W. Shin, J. Jang, S. Joo, Y. Kwon, Y. Moon, and L-S. Kim, "Multiple Clone Row DRAM: A Low Latency and Area Optimized DRAM," in ISCA, 2013.
[32] Y. Chou, B. Fabi, and S. Abraham, "Microarchitecture Optimizations for Exploiting Memory-Level Parallelism," in ISCA, 2004.
[33] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi, "Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems," in HPCA, 2013.
[34] W. Ding, D. Guttmann, and M. Kandemir, "Compiler Support for Optimizing Memory Bank-Level Parallelism," in MICRO, 2014.
[35] N. Draper et al., "Die Stacking (3D) Microarchitecture, " in ISCA, 2012.
[36] M. J. Charney and T. R. Pozuk, "Prefetching and Memory System Behavior of the SPEC95 Benchmark Suite," IBM JRD, 1997.
[37] N. Chatterjee, M. Shevgoor, R. Balasubramanian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer, "Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access," in MICRO, 2012.
[38] J. Choi, W. Shin, J. Jang, S. Joo, Y. Kwon, Y. Moon, and L-S. Kim, "Multiple Clone Row DRAM: A Low Latency and Area Optimized DRAM," in ISCA, 2013.
[39] Y. Chou, B. Fabi, and S. Abraham, "Microarchitecture Optimizations for Exploiting Memory-Level Parallelism," in ISCA, 2004.
