Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture

Donghyuk Lee Yoongu Kim Vivek Seshadri Jamie Liu Lavanya Subramanian Onur Mutlu
Carnegie Mellon University

Abstract

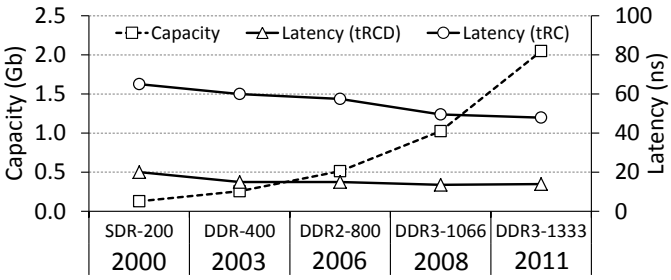
The capacity and cost-per-bit of DRAM have historically scaled to satisfy the needs of increasingly large and complex computer systems. However, DRAM latency has remained almost constant, making memory latency the performance bottleneck in today's systems. We observe that the high access latency is not intrinsic to DRAM, but a trade-off made to decrease cost-per-bit. To mitigate the high area overhead of DRAM sensing structures, commodity DRAMs connect many DRAM cells to each sense-amplifier through a wire called a bitline. These bitlines have a high parasitic capacitance due to their long length, and this bitline capacitance is the dominant source of DRAM latency. Specialized low-latency DRAMs use shorter bitlines with fewer cells, but have a higher cost-per-bit due to greater sense-amplifier area overhead. In this work, we introduce Tiered-Latency DRAM (TL-DRAM), which achieves both low latency and low cost-per-bit. In TL-DRAM, each long bitline is split into two shorter segments by an isolation transistor, allowing one segment to be accessed with the latency of a short-bitline DRAM without incurring high cost-per-bit. We propose mechanisms that use the low-latency segment as a hardware-managed or software-managed cache. Evaluations show that our proposed mechanisms improve both performance and energy-efficiency for both single-core and multi-programmed workloads.

1 Introduction

Primarily due to its low cost-per-bit, DRAM has long been the choice substrate for architecting main memory subsystems. In fact, DRAM's cost-per-bit has been decreasing at a rapid rate as DRAM process technology scales to integrate ever more DRAM cells into the same die area. As a result, each successive generation of DRAM has enabled increasingly large-capacity main memory subsystems at low cost.

In stark contrast to the continued scaling of cost-per-bit, the *latency* of DRAM has remained almost constant. During the same 11-year interval in which DRAM's cost-per-bit decreased by a factor of 16, DRAM latency (as measured by the t_{RCD} and t_{RC} timing constraints[1]) decreased by only 30.5% and 26.3%, as shown in Fig 1. From the perspective of the processor, an access to DRAM takes hundreds of cycles – time during which the processor may be stalled, waiting for DRAM. Such wasted time leads to large performance degradations commonly referred to as the "memory wall" [58] or the "memory gap" [56].

However, the high latency of commodity DRAM chips is in fact a *deliberate* trade-off made by DRAM manufacturers. While process technology scaling has enabled DRAM designs with both lower cost-per-bit *and* lower latency [16], DRAM manufacturers have usually sacrificed the latency benefits of scaling in order to achieve even lower cost-per-bit, as we explain below. Hence, while low-latency DRAM chips exist [25, 27, 45], their higher cost-per-bit relegates them to specialized applications such as high-end networking equipment that require very low latency even at a very high cost [6].

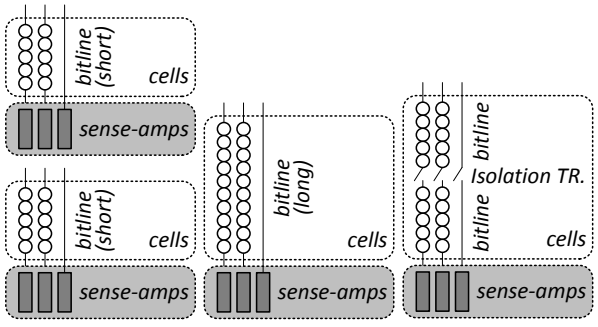


† We refer to the dominant DRAM chips during the period of time [5, 19].

Figure 1. DRAM Capacity & Latency Over Time [5, 19, 35, 43]

In DRAM, each bit is represented by electrical charge on a capacitor-based *cell*. The small size of this capacitor necessitates the use of an auxiliary structure, called a *sense-amplifier*, to sense the small amount of charge held by the cell and amplify it to a full digital logic value. A sense-amplifier is up to three orders of magnitude larger than a cell [39]. To mitigate the large size of sense-amplifiers, each sense-amplifier is connected to many DRAM cells through a wire called a *bitline*.

DRAM manufacturers trade latency for cost-per-bit by adjusting the length of these bitlines. Shorter bitlines (fewer cells connected to the bitline) constitute a smaller electrical load on the bitline, resulting in decreased latency, but require a larger number of sense-amplifiers for a given DRAM capacity (Fig 2a), resulting in higher cost-per-bit. Longer bitlines (more cells connected to the bitline) require fewer sense-amplifiers for a given DRAM capacity (Fig 2b), reducing cost-per-bit, but impose a higher electrical load on the bitline, increasing latency. As a result, neither of these two approaches can optimize for both cost-per-bit and latency.



(a) Latency Opt. (b) Cost Opt. (c) Our Proposal

Figure 2. DRAM: Latency vs. Cost Optimized, Our Proposal

Our goal is to design a new DRAM architecture that provides low latency for the common case while still retaining a low cost-per-bit overall. Our proposal, which we call *Tiered-Latency DRAM*, is based on the key observation that *long bitlines are the dominant source of DRAM latency* [47].

Our key idea is to adopt long bitlines to achieve low cost-per-bit, while allowing their lengths to appear shorter in order to achieve low latency. In our mechanism (*Tiered-Latency DRAM*), each long bitline is split into two shorter segments using an *isolation transistor*, as shown in Fig 2c: the segment that

[1]The overall DRAM access latency can be decomposed into individual DRAM *timing constraints* (Sec 2.2). Two of the most important timing constraints are t_{RCD} (row-to-column delay) and t_{RC} ("row-conflict" latency). When DRAM is lightly loaded, t_{RCD} is often the bottleneck, whereas when DRAM is heavily loaded, t_{RC} is often the bottleneck (Secs 2.2 and 2.3).

is connected directly to the sense-amplifier is called the *near segment*, whereas the other is called the *far segment*.

Table 1 summarizes the latency and die-size (i.e., cost-per-bit[2]) characteristics of Tiered-Latency DRAM and other DRAMs (refer to Sec 3 for details). Compared to commodity DRAM (long bitlines) which incurs high access latency for all cells, Tiered-Latency DRAM offers significantly reduced access latency for cells in the near segment, while not always increasing the access latency for cells in the far segment. In fact, for cells in the far segment, although t_{RC} is increased, t_{RCD} is actually *decreased.*

Table 1. Latency & Die-Size Comparison of DRAMs (Sec 3)

		Short Bitline (Fig 2a)	Long Bitline (Fig 2b)	Segmented Bitline (Fig 2c)	
		Unsegmented	Unsegmented	Near	Far
Length (Cells)		32	512	32	480
Latency	t_{RCD}	Lowest (8.2ns)	High (15ns)	Lowest (8.2ns)	**Low** (12.1ns)
	t_{RC}	Lowest (23.1ns)	High (52.5ns)	Lowest (23.1ns)	Higher (65.8ns)
Normalized Die-Size (Cost)		Highest (3.76)	Lowest (1.00)	Low (1.03)	

To access a cell in the near segment, the isolation transistor is turned off, so that the cell and the sense-amplifier see[3] only the portion of the bitline corresponding to the near segment (i.e., reduced electrical load). Therefore, the near segment can be accessed quickly. On the other hand, to access a cell in the far segment, the isolation transistor is turned on to connect the entire length of the bitline to the sense-amplifier. In this case, however, the cell and the sense-amplifier see the full electrical load of the bitline in addition to the extra load of the isolation transistor. Due to the increased total load, t_{RC} of the far segment is indeed higher than that of a long unsegmented bitline (Table 1). However, counter-intuitively, t_{RCD} of the far segment is actually *lower* than that of a long unsegmented bitline (highlighted in Table 1). As we will explain in Section 4.1, this is because the isolation transistor's resistance decouples the two segments to a certain degree.

We describe two different approaches to exploit the asymmetric latency characteristics of the near and the far segments. In the first approach, the near segment capacity is not exposed to the operating system (OS). Rather, the memory controller uses the near segment as a hardware-managed cache for the far segment. We propose three different policies to manage the near segment cache. The three policies differ in when a row in the far segment is cached into the near segment and when it is evicted from the near segment. In the second approach, the near segment capacity is exposed to the OS, enabling the OS to use the full available DRAM capacity. We propose two mechanisms, one where the memory controller uses an additional layer of indirection to map frequently accessed pages to the near segment, and another where the OS uses static/dynamic profiling to directly map frequently accessed pages to the near segment. In both approaches, the accesses to pages that are mapped to the near segment are served faster and with lower power than in conventional DRAM, resulting in improved system performance and energy efficiency.

[2]For the same number of bits, we assume that a DRAM chip's cost-per-bit is linearly correlated with the die-size.

[3]In a closed electrical circuit that connects multiple electrical components to each other, one component is said to "see" (i.e., experience) the effective electrical load of another component, and vice versa.

This paper makes the following contributions.

- Based on the observation that long internal wires (bitlines) are the dominant source of DRAM latency, we propose a new DRAM architecture, Tiered-Latency DRAM. To our knowledge, this is the first work that enables low-latency DRAM without significantly increasing cost-per-bit.
- We quantitatively evaluate the latency, area, and power characteristics of Tiered-Latency DRAM through circuit simulations based on a publicly available 55nm DRAM process technology [39].
- We describe two major ways of leveraging TL-DRAM: 1) by not exposing the near segment capacity to the OS and using it as a hardware-managed cache, and 2) by exposing the near segment capacity to the OS and using hardware/software to map frequently accessed pages to the near segment. We propose two new policies to manage the near segment cache that specifically exploit the asymmetric latency characteristics of TL-DRAM.
- We evaluate our proposed memory management policies on top of the Tiered-Latency DRAM substrate and show that they improve both system performance and energy efficiency for both single-program and multi-programmed workloads.

2 DRAM Background

A DRAM chip consists of numerous cells that are grouped at different granularities to form a hierarchical organization as shown in Fig 3. First, a *DRAM chip* is the set of all cells on the same silicon die, along with the I/O circuitry which allows the cells to be accessed from outside the chip. Second, a DRAM chip is divided into multiple banks (e.g., eight for DDR3), where a *bank* is a set of cells that share peripheral circuitry such as the address decoders (row and column). Third, a bank is further sub-divided into tens of subarrays, where a *subarray* [24] is a set of cells that share bitlines and sense-amplifiers.

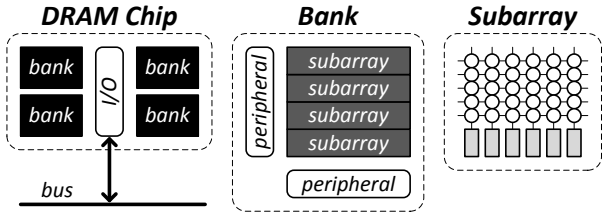
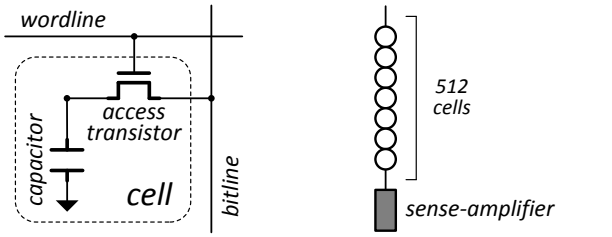


Figure 3. DRAM Organization

In this work, we are predominantly concerned with the operation and latency of the bitlines, which are entirely encapsulated at the subarray-level scope. While I/O and peripherals at the chip-level and bank-level also contribute to the DRAM access latency, to a certain extent, this latency is overlapped with the large latency at the subarray, as we will explain in Sec 2.2.

2.1 Subarray Organization

A DRAM subarray is a two-dimensional array of elementary units called *cells*. As shown in Fig 4a, a cell consists of two components: *i)* a capacitor that represents binary data in the form of stored electrical charge and *ii)* an access transistor that is switched on/off to connect/disconnect the capacitor to a wire called the *bitline*. As shown in Fig 4b, there are approximately 512 cells in the vertical direction (a "column" of cells), all of which share the same bitline. For each bitline, there is a *sense-amplifier* whose main purpose is to read from a cell by reliably detecting the very small amount of electrical charge stored in the cell. When writing to a cell, on the other hand, the sense-amplifier acts as an electrical driver and programs the cell by filling or depleting its stored charge.



(a) Cell (b) Bitline & Sense-Amplifier

Figure 4. DRAM Elementary Components

Numerous bitlines (and their associated sense-amplifiers) are laid side-by-side in parallel to compose a subarray (Fig 3). All cells in the horizontal direction (a "row" of cells) have their access-transistors controlled by a shared wire called the *wordline*. When the wordline voltage is raised to V_{DD}, all cells of a row are connected to their respective bitlines and sensed in lockstep by the sense-amplifiers. This is why the set of all sense-amplifiers in a subarray is also called a *row-buffer*. At any given time, at most one wordline in a subarray is ever raised (i.e., at most one cell per column is connected to the bitline) – otherwise, cells in the same column would corrupt each other's data.

2.2 Three Phases of DRAM Access

As the timelines in Fig 5 show, a DRAM chip access can be broken down into three distinct *phases*: *i)* activation, *ii)* I/O, and *iii)* precharging. Activation and precharging occur entirely within the subarray, whereas I/O occurs in the peripherals and I/O circuitry. First, during the *activation* phase, a wordline within a subarray is raised, connecting a row of cells to the bitlines. Soon thereafter, the data in the row of cells is copied to the sense-amplifiers of that subarray. Second, during the *I/O* phase, the sense-amplifiers transfer the data through the peripherals to the DRAM chip's I/O circuitry. From there, the data leaves the DRAM chip and is sent to the processor over the memory bus. As Fig 5 shows, the I/O phase's latency is overlapped with the latency of the activation phase. Third, during the *precharging* phase, the raised wordline in the subarray is lowered, disconnecting the row of cells from the bitlines. Also, the subarray's sense-amplifiers and bitlines are initialized (i.e., cleared of their data) to prepare for the next access to a new row of cells.

Three DRAM Commands. The DRAM controller (typically residing on the processor die) issues *commands* to the DRAM chip to initiate the three phases listed above. As shown in Fig 5, there are three commands, one for each phase. In their respective order, they are: ACTIVATE (ACT), READ/WRITE, and PRECHARGE (PRE). Among the commands, ACTIVATE and PRECHARGE are subarray-related commands since they directly operate on the subarray, whereas READ and WRITE are I/O-related commands.

Timing Constraints. After the DRAM controller issues a command to initiate a phase, it must wait for a sufficient amount of time before issuing the next command. Such restrictions imposed between the issuing of commands are called *timing constraints*. DRAM timing constraints are visualized in Fig 5 and summarized in Table 2. Two of the most important timing constraints are t_{RCD} (*row-to-column delay*) and t_{RC} (*row-cycle time*). Every time a new row of cells is accessed, the subarray incurs t_{RCD} (15ns; ACTIVATE→READ/WRITE) to copy the row into the sense-amplifiers. On the other hand, when there are multiple accesses to different rows in the same subarray, an earlier access delays all later accesses by t_{RC} (52.5ns; ACTIVATE→ACTIVATE). This is because the subarray needs time to complete the activation phase (t_{RAS}) and the precharging phase (t_{RP}) for the earlier access, whose sum is defined as t_{RC} ($= t_{RAS} + t_{RP}$), as shown in Fig 5.

Access Latency. Fig 5 illustrates how the DRAM access latency can be decomposed into individual DRAM timing constraints. Specifically, the figure shows the latencies of two read accesses (to different rows in the same subarray) that are served one after the other. From the perspective of the first access, DRAM is "unloaded" (i.e., no prior timing constraints are in effect), so the DRAM controller immediately issues an ACTIVATE on its behalf. After waiting for t_{RCD}, the controller issues a READ, at which point the data leaves the subarray and incurs additional latencies of t_{CL} (peripherals and I/O circuitry) and t_{BL} (bus) before it reaches the processor. Therefore, the latency of the first access is 37.5ns ($t_{RCD} + t_{CL} + t_{BL}$). On the other hand, the second access is delayed by the timing constraint that is in effect due to the first access (t_{RC}) and experiences a large "loaded" latency of 90ns ($t_{RC}+t_{RCD}+t_{CL}+t_{BL}$).

2.3 Subarray Operation: A Detailed Look

Subarray-related timing constraints (t_{RCD} and t_{RC}) constitute a significant portion of the unloaded and loaded DRAM access latencies: 40% of 37.5ns and 75% of 90ns, respectively. Since t_{RCD} and t_{RC} exist only to safeguard the timely operation of the underlying subarray, in order to understand why their values are so large, we must first understand how the subarray operates during the activation and the precharging phases. (As previously explained, the I/O phase does not occur within the subarray and its latency is overlapped with the activation phase.) Specifically, we will show how the bitline plays a crucial role in both activation and precharging, such that it heavily influences both t_{RCD} and t_{RC} (Fig 6).

- **Activation (Charge Sharing).** Before it is accessed, the cell is initially in the quiescent state (State ①, Fig 6). An access to the cell begins when its access-transistor is turned on by an ACTIVATE, connecting its capacitor to the bitline. Slowly, the charge in the cell capacitor (or the lack thereof) is shared with the *bitline parasitic capacitor*, thereby perturbing the bitline voltage away from its quiescent value ($0.5V_{DD}$) in the positive (or negative) direction (State ②). During charge-sharing, note that the cell's charge is modified (i.e., data is lost) because it is shared with the bitline. But this is only temporary since the cell's charge is restored as part of the next step, as described below.

- **Activation (Sensing & Amplification).** After allowing sufficient time for the charge sharing to occur, the sense-amplifier is turned on. Immediately, the sense-amplifier "senses" (i.e., observes) the polarity of the perturbation on the bitline voltage. Then the sense-amplifier "amplifies" the perturbation by injecting (or withdrawing) charge into (or from) both the cell capacitor and the bitline parasitic capacitor. After a latency of t_{RCD}, midway through amplification, enough charge has been injected (or withdrawn) such that the bitline voltage reaches a *threshold* state of $0.75V_{DD}$ (or $0.25V_{DD}$). At this point, data is considered to have been "copied" from the cell to the sense-amplifier (State ③). In other words, the bitline voltage is now close enough to V_{DD} (or *0*) for the sense-amplifier to detect a binary data value of '1' (or '0') and transfer the data to the I/O circuitry, allowing READ and WRITE commands to be issued. Eventually, the voltage of the bitline and the cell are fully amplified to V_{DD} or *0* (State ④). Only at this point is the charge in the cell fully restored to its original value. The latency to reach this *restored* state (State ④) is t_{RAS} (which is one component of t_{RC}).

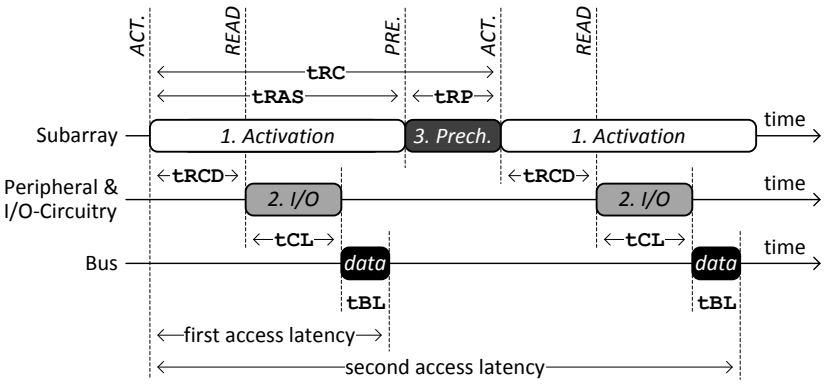


Figure 5. Three Phases of DRAM Access

Table 2. Timing Constraints (DDR3-1066) [43]

Phase	Commands	Name	Value
1	ACT → READ ACT → WRITE	tRCD	15ns
	ACT → PRE	tRAS	37.5ns
2	READ → *data* WRITE → *data*	tCL tCWL	15ns 11.25ns
	data burst	tBL	7.5ns
3	PRE → ACT	tRP	15ns
1 & 3	ACT → ACT	tRC (tRAS+tRP)	52.5ns

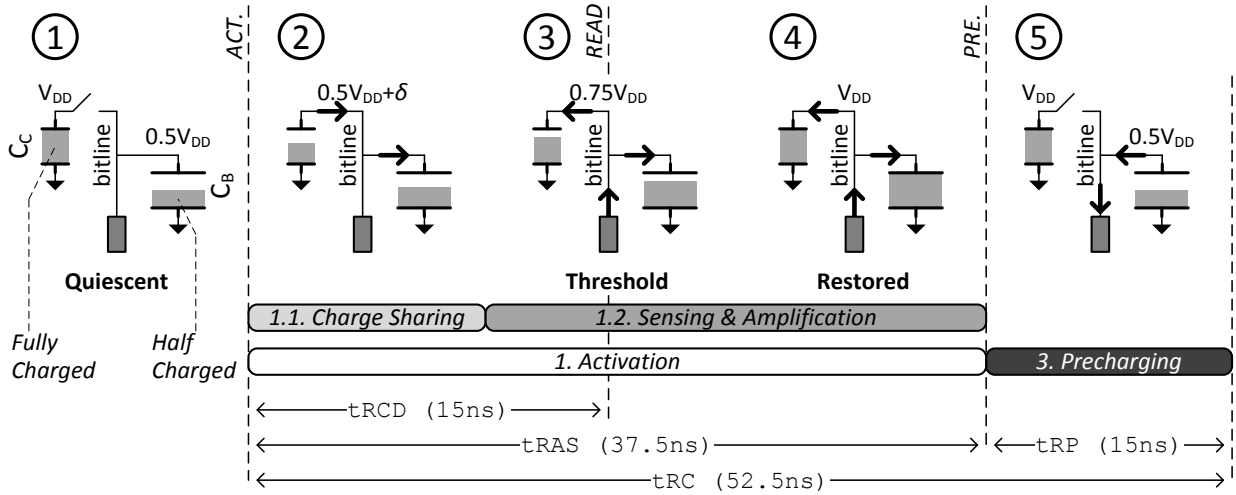


Figure 6. Charge Flow Between the Cell Capacitor (C_C), Bitline Parasitic Capacitor (C_B), and the Sense-Amplifier ($C_B \approx 3.5C_C$ [39])

- **Precharging**. An access to a cell is terminated by turning off its access-transistor by a PRECHARGE. By doing so, the cell becomes decoupled from the bitline and is not affected by any future changes in the bitline voltage. The sense-amplifier then withdraws (or injects) charge from the bitline parasitic capacitor such that the bitline voltage reaches the quiescent value of $0.5V_{DD}$ (State ⑤). Precharging is required to ensure that the next accessed cell can perturb the bitline voltage in either direction (towards V_{DD} or towards *0*). This would not be possible if the bitline is left unprecharged at V_{DD} or *0*. The latency to precharge the bitline is t_{RP} (which is the other component of t_{RC}).

Summary. Through our discussion, we have established a relationship between the subarray timing constraints (t_{RCD} and t_{RC}) and the bitline parasitic capacitance. To summarize, t_{RCD} and t_{RC} are determined by how quickly the bitline voltage can be driven – for t_{RCD}, from $0.5V_{DD}$ to $0.75V_{DD}$ (*threshold*); for t_{RC}, from $0.5V_{DD}$ to V_{DD} (*restored*) and back again to $0.5V_{DD}$. In turn, the drivability of the bitline is determined by the bitline parasitic capacitance, whose value is a function of the bitline length, as we discuss next.

3 Motivation: Short vs. Long Bitlines

The key parameter in the design of a DRAM subarray is the number of DRAM cells connected to each bitline (cells-per-bitline) – i.e., the number of DRAM rows in a subarray. This number directly affects the length of the bitline, which in turn affects both the access latency and the area of the subarray. As we describe in this section, the choice of the number of cells-per-bitline presents a crucial trade-off between the DRAM access latency and the DRAM die-size.

3.1 Latency Impact of Cells-per-Bitline

Every bitline has an associated parasitic capacitance whose value is proportional to the length of the bitline. This parasitic capacitance increases the subarray operation latencies: *i)* charge sharing, *ii)* sensing & amplification, and *iii)* precharging, which we discussed in Fig 6.

First, the bitline capacitance determines the bitline voltage after charge sharing. The larger the bitline capacitance, the closer its voltage will be to $0.5V_{DD}$ after charge sharing. Although this does not significantly impact the latency of charge sharing, this causes the sense-amplifier to take longer to amplify the voltage to the final restored value (V_{DD} or *0*).

Second, in order to amplify the voltage perturbation on the bitline, the sense-amplifier injects (or withdraws) charge into (or from) both the cell and the bitline. Since the sense-amplifier can do so only at a fixed rate, the aggregate capacitance of the cell and the bitline determine how fast the bitline voltage reaches the *threshold* and the *restored* states (States ③ and ④, Fig 6). A long bitline, which has a large parasitic capacitance, slows down the bitline voltage from reaching these states, thereby lengthening both t_{RCD} and t_{RAS}, respectively.

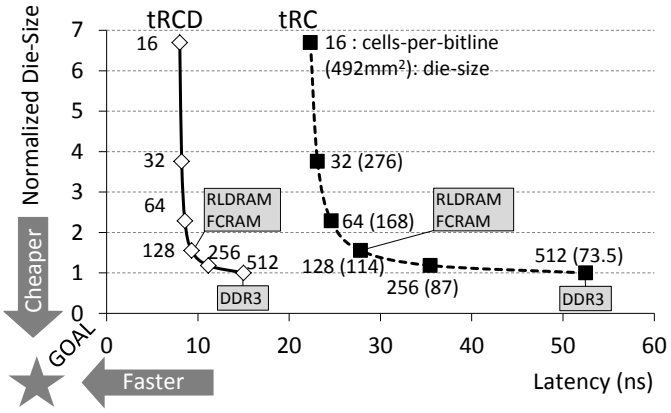
Third, to precharge the bitline, the sense-amplifier drives the bitline voltage to the quiescent value of $0.5V_{DD}$. Again, a long bitline with a large capacitance is driven more slowly and hence has a large t_{RP}.

3.2 Die-Size Impact of Cells-per-Bitline

Since each cell on a bitline belongs to a row of cells (spanning horizontally across multiple bitlines), the number of cells-per-bitline in a subarray is equal to the number of rows-per-subarray. Therefore, for a DRAM chip with a given capacity (i.e., fixed total number of rows), one can either have many subarrays with short bitlines (Fig 2a) or few subarrays with long bitlines (Fig 2b). However, since each subarray requires its own set of sense-amplifiers, the size of the DRAM chip increases along with the number of subarrays. As a result, for a given DRAM capacity, its die-size is inversely proportional to the number of cells-per-bitline (as a first-order approximation).

3.3 Trade-Off: Latency vs. Die-Size

From the above discussion, it is clear that a short bitline (fewer cells-per-bitline) has the benefit of lower subarray latency, but incurs a large die-size overhead due to additional sense-amplifiers. On the other hand, a long bitline (more cells-per-bitline), as a result of the large bitline capacitance, incurs high subarray latency, but has the benefit of reduced die-size overhead. To study this trade-off quantitatively, we ran transistor-level circuit simulations based on a publicly available 55nm DRAM process technology [39]. Fig 7 shows the results of these simulations. Specifically, the figure shows the latency (t_{RCD} and t_{RC}) and the die-size for different values of cells-per-bitline. The figure clearly shows the above described trade-off between DRAM access latency and DRAM die-size.



† RLDRAM bitline length is estimated from its latency and die-size [21, 27].
† The reference DRAM is 55nm 2GB DDR3 [39].

Figure 7. Bitline Length: Latency vs. Die-Size

As Fig 7 shows, existing DRAM architectures are either optimized for die-size (commodity DDR3 [43, 30]) and are thus low cost but high latency, or optimized for latency (RLDRAM [27], FCRAM [45]) and are thus low latency but high cost. **Our goal** is to design a DRAM architecture that achieves the best of both worlds – i.e., low access latency and low cost.

4 Tiered-Latency DRAM (TL-DRAM)

To obtain both the latency advantages of short bitlines and the cost advantages of long bitlines, we propose the *Tiered-Latency DRAM* (TL-DRAM) architecture, as shown in Fig 8. The key idea of TL-DRAM is to introduce an *isolation transistor* that divides a long bitline into two segments: the *near segment*, connected directly to the sense-amplifier, and the *far segment*, connected through the isolation transistor. Unless otherwise stated, throughout the following discussion, we assume, without loss of generality, that the isolation transistor divides the bitline such that length of the near and far segments is 128 cells and 384 cells (=512-128), respectively. (Sec 4.1 discusses the latency sensitivity to the segment lengths.)

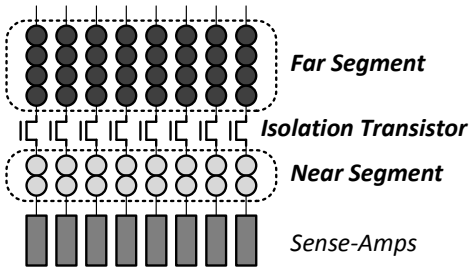


Figure 8. TL-DRAM: Near vs. Far Segments

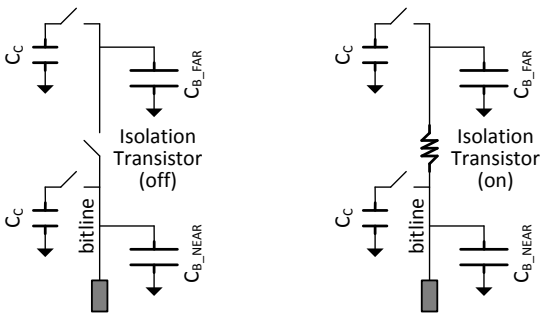
4.1 Latency Analysis (Overview)

The primary role of the isolation transistor is to electrically decouple the two segments from each other. As a result, the effective bitline length (and also the effective bitline capacitance) as seen by the cell and sense-amplifier is changed. Correspondingly, the latency to access a cell is also changed – albeit differently depending on whether the cell is in the near or the far segment (Table 3), as will be explained next.[4]

Table 3. Segmented Bitline: Effect on Latency

	Near Segment (128 cells)	Far Segment (384 cells)
t_{RCD}	**Reduced** (15ns → 9.3ns)	**Reduced** (15ns → 13.2ns)
t_{RC}	**Reduced** (52.5ns → 27.8ns)	Increased (52.5ns → 64.1ns)

Near Segment. When accessing a cell in the near segment, the isolation transistor is turned off, disconnecting the far segment (Fig 9a). Since the cell and the sense-amplifier see only the reduced bitline capacitance of the shortened near segment, they can drive the bitline voltage more easily. In other words, for the same amount of charge that the cell or the sense-amplifier injects into the bitline, the bitline voltage is higher for the shortened near segment compared to a long bitline. As a result, the bitline voltage reaches the *threshold* and *restored* states (Fig 6, Sec 2.3) more quickly, such that t_{RCD} and t_{RAS} for the near segment is significantly reduced. Similarly, the bitline can be precharged to $0.5V_{DD}$ more quickly, leading to a reduced t_{RP}. Since t_{RC} is defined as the sum of t_{RAS} and t_{RP} (Sec 2.2), t_{RC} is reduced as well.

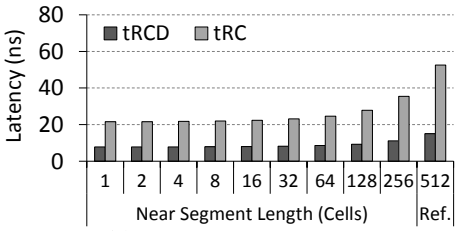


(a) Near Segment Access **(b) Far Segment Access**

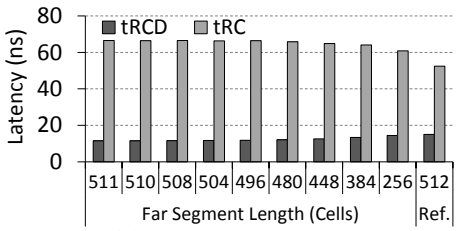
Figure 9. Circuit Model of Segmented Bitline

Far Segment. On the other hand, when accessing a cell in the far segment, the isolation transistor is turned on to connect the entire length of the bitline to the sense-amplifier. In this case, however, the isolation transistor acts like a resistor inserted between the two segments (Fig 9b).

[4] Note that the latencies shown in Table 3 differ from those shown in Table 1 due to differing near segment lengths.

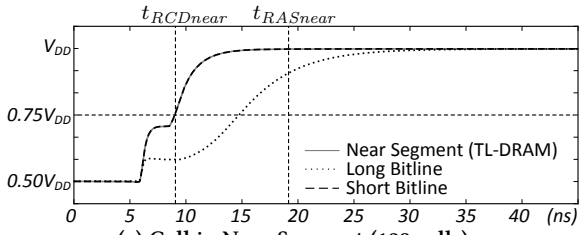


(a) Cell in Near Segment

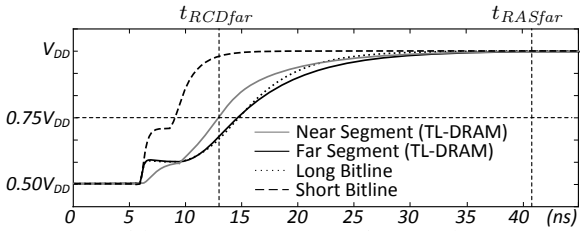


(b) Cell in Far Segment

Figure 10. Latency Analysis

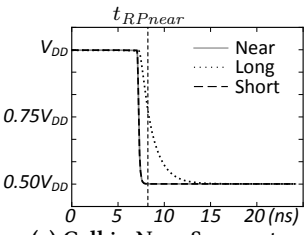


(a) Cell in Near Segment (128 cells)

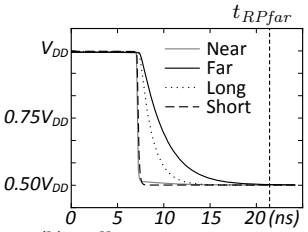


(b) Cell in Far Segment (384 cells)

Figure 11. Activation: Bitline Voltage



(a) Cell in Near Segment



(b) Cell in Far Segment

Figure 12. Precharging

When the sense-amplifier is turned on during activation (as explained in Sec 2.3), it begins to drive charge onto the bitline. In commodity DRAM, this charge is spread across the large capacitance of the entire long bitline. However, in TL-DRAM, the resistance of the isolation transistor limits how quickly charge flows to the far segment, such that the reduced capacitance of the shortened near segment is charged more quickly than that of the far segment. This has two key consequences. First, because the near segment capacitance is charged quickly, the near segment voltage rises more quickly than the bitline voltage in commodity DRAM. As a result, the near segment voltage more quickly reaches $0.75V_{DD}$ (*threshold* state, Sec 2.3) and, correspondingly, the sense-amplifier more quickly detects the binary data value of '1' that was stored in the far segment cell. That is why t_{RCD} is lower in TL-DRAM than in commodity DRAM even for the far segment. Second, because the far segment capacitance is charged more slowly, it takes *longer* for the far segment voltage — and hence the cell voltage — to be *restored* to V_{DD} or *0*. Since t_{RAS} is the latency to reach the *restored* state (Sec 2.3), t_{RAS} is increased for cells in the far segment. Similarly, during precharging, the far segment voltage reaches $0.5V_{DD}$ more slowly, for an increased t_{RP}. Since t_{RAS} and t_{RP} both increase, their sum t_{RC} also increases.

Sensitivity to Segment Length. The lengths of the two segments are determined by where the isolation transistor is placed on the bitline. Assuming that the number of cells per bitline is fixed at 512 cells, the near segment length can range from as short as a single cell to as long as 511 cells. Based on our circuit simulations, Fig 10a and Fig 10b plot the latencies of the near and far segments as a function of their length, respectively. For reference, the rightmost bars in each figure are the latencies of an unsegmented long bitline whose length is 512 cells. From these figures, we draw three conclusions. First, the shorter the near segment, the lower its latencies (t_{RCD} and t_{RC}). This is expected since a shorter near segment has a lower effective bitline capacitance, allowing it to be driven to target voltages more quickly. Second, the longer the far segment, the lower the far segment's t_{RCD}. Recall from our previous discussion that the far segment's t_{RCD} depends on how quickly the near segment (not the far segment) can be driven. A longer far segment implies a shorter near segment (lower capacitance), and that is why t_{RCD} of the far segment decreases. Third, the shorter the far segment, the smaller its t_{RC}. The far segment's t_{RC} is determined by how quickly it reaches the full voltage (V_{DD} or *0*). Regardless of the length of the far segment, the current that trickles into it through the isolation transistor does not change significantly. Therefore, a shorter far segment (lower capacitance) reaches the full voltage more quickly.

4.2 Latency Analysis (Circuit Evaluation)

We model TL-DRAM in detail using SPICE simulations. Simulation parameters are mostly derived from a publicly available 55nm DDR3 2Gb process technology file [39] which includes information such as cell and bitline capacitances and resistances, physical floorplanning, and transistor dimensions. Transistor device characteristics were derived from [33] and scaled to agree with [39].

Fig 11 and Fig 12 show the bitline voltages during activation and precharging respectively. The x-axis origin (time 0) in the two figures correspond to when the subarray receives the ACTIVATE or the PRECHARGE command, respectively. In addition to the voltages of the segmented bitline (near and far segments), the figures also show the voltages of two unsegmented bitlines (short and long) for reference.

Activation (Fig 11). First, during an access to a cell in the near segment (Fig 11a), the far segment is disconnected and is floating (hence its voltage is not shown). Due to the reduced bitline capacitance of the near segment, its voltage increases almost as quickly as the voltage of a short bitline (the two curves are overlapped) during both charge sharing and sensing & amplification. Since the near segment voltage reaches $0.75V_{DD}$ and V_{DD} (the *threshold* and *restored* states) quickly, its t_{RCD} and t_{RAS}, respectively, are significantly reduced compared to a long bitline. Second, during an access to a cell in the far segment (Fig 11b), we can indeed verify that the voltages of the near and the far segments increase at different rates due to the resistance of the isolation transistor, as previously explained. Compared to a long bitline, while the near segment voltage reaches $0.75V_{DD}$ more quickly, the far segment voltage reaches V_{DD} more slowly. As a result, t_{RCD} of the far segment is reduced while its t_{RAS} is increased.

Precharging (Fig 12). While precharging the bitline after accessing a cell in the near segment (Fig 12a), the near segment reaches $0.5V_{DD}$ quickly due to the smaller capacitance, almost as quickly as the short bitline (the two curves are overlapped). On the other hand, precharging the bitline after accessing a cell in the far segment (Fig 12b) takes longer compared to the long bitline baseline. As a result, t_{RP} is reduced for the near segment and increased for the far segment.

4.3 Die-Size Analysis

Adding an isolation transistor to the bitline increases only the height of the subarray and not the width (Fig 8). Without the isolation transistor, the height of a baseline subarray is equal to the sum of height of the cells and the sense-amplifier. In the following analysis, we use values from the Rambus power model [39].[5] The sense amplifier and the isolation transistor are respectively 115.2x and 11.5x taller than an individual cell. For a subarray with 512 cells, the overhead of adding a single isolation transistor is $\frac{11.5}{115.2+512}$ = 1.83%.

Until now we have assumed that all cells of a DRAM row are connected to the same row of sense-amplifiers. However, the sense-amplifiers are twice as wide as an individual cell. Therefore, in practice, only every other bitline (cell) is connected to a bottom row of sense-amplifiers. The remaining bitlines are connected to the another row of sense-amplifiers of the vertically adjacent (top) subarray. This allows for tighter packing of DRAM cells within a subarray. As a result, each subarray requires two sets of isolation transistors. Therefore, the increase in subarray area due to the two isolation transistors is 3.66%. Once we include the area of the peripheral and I/O circuitry which does not change due to the addition of the isolation transistors, the resulting DRAM die-size area overhead is 3.15%.

4.4 Enabling Inter-Segment Data Transfer

One way of exploiting TL-DRAM's asymmetric latencies is to use the near segment as a cache to the far segment. Compared to the far segment, the near segment is smaller and faster. Therefore, frequently-used or latency-critical data can benefit significantly from being placed in the near segment as opposed to the far segment. The problem lies in enabling an efficient way of transferring (copying or migrating) data between the two segments. Unfortunately, in existing DRAM chips, even to transfer data from one row to another row within the same subarray, the data must be read out externally to the DRAM controller and then written back to the chip. This wastes significant amounts of power and bandwidth on the memory bus (that connects the DRAM chip to the DRAM controller) and incurs large latency.

In TL-DRAM, data transfer between the two segments occurs entirely within DRAM without involving the external memory bus. TL-DRAM leverages the fact that the bitline itself is essentially a bus that connects to all cells in both the near and far segments. As an example, let us assume that we are accessing a cell in the far segment (transfer source). When the bitline has reached the *restored* state (Sec 2.3), the data in the cell is fully copied onto the bitline. Normally, at this point, the DRAM controller would issue a PRECHARGE to clear the bitline voltage to $0.5V_{DD}$. Instead, TL-DRAM allows the DRAM controller to issue another ACTIVATE, this time to a cell in the near segment (transfer destination). Since the bitline is at a full voltage, the bitline drives the near segment cell so that data is copied into it. According to our simulations, writing into the destination cell takes about 4ns. In fact, the destination cell can be connected to the bitline even before the source cell has reached the *restored* state, thereby overlapping the copying latency with the t_{RAS} latency. More generally, the source and the destination can be any two cells connected to the same bitline, regardless of which segment they lie on.

5 Leveraging the TL-DRAM Substrate

One simple way of leveraging the TL-DRAM substrate is to use the near segment as a hardware-managed cache for the far segment. In this approach, the memory controller does not expose the near segment capacity to the operating system (OS). While this approach reduces the overall available memory capacity, it keeps both the hardware and the software design simple. Another alternative approach is to expose the near segment capacity to the OS. As we will describe in Section 5.2, effectively exploiting the TL-DRAM substrate using this alternate approach may slightly increase the complexity of the hardware or the software. We now describe our different mechanisms to leverage the TL-DRAM substrate.

5.1 Near Segment as an OS-Transparent Hardware-Managed Cache

We describe three different mechanisms that use the near segment as a hardware-managed cache to the far segment. In all three mechanisms, the memory controller tracks the rows in the far segment that are cached in the near segment (for each subarray). The three mechanisms differ in 1) when they cache a far-segment row in the near segment, and 2) when they evict a row from the near segment.

Mechanism 1: Simple Caching (SC). Our first mechanism, *Simple Caching* (SC), utilizes the near segment as an LRU cache to the far segment. Under this mechanism, the DRAM controller categorizes a DRAM access into one of three cases: *i)* sense-amplifier hit: the corresponding row is already activated; *ii)* near segment hit: the row is already cached in the near segment; and *iii)* near segment miss: the row is not cached in the near segment. In the first case, sense-amplifier hit (alternatively, row-buffer hit), the access is served directly from the row-buffer. Meanwhile, the LRU-ordering of the rows cached in the near segment remains unaffected. In the second case, near segment hit, the DRAM controller quickly activates the row in the near segment, while also updating it as the MRU row. In the third case, near segment miss, the DRAM controller checks whether the LRU row (eviction candidate) in the near segment is dirty. If so, the LRU row must first be copied (or written back) to the far segment using the transfer mechanism described in Section 4.4. Otherwise, the DRAM controller directly activates the far segment row (that needs to be accessed) and copies (or caches) it into the near segment and updates it as the MRU row.

Mechanism 2: Wait-Minimized Caching (WMC). When two accesses to two different rows of a subarray arrive almost simultaneously, the first access delays the second access by a large amount, t_{RC}. Since the first access causes the second access to *wait* for a long time, we refer to the first access as a *wait-inducing access*. Assuming both rows are in the far segment, the latency at the subarray experienced by the second access is $t_{RCfar} + t_{RCDfar}$ (77.3ns). Such a large latency is mostly due to the wait caused by the first access, t_{RCfar} (64.1ns). Hence, it is important for the second access that the wait is minimized, which can be achieved by caching the *first* accessed data in the near segment. By doing so, the wait is significantly reduced from t_{RCfar} (64.1ns) to t_{RCnear} (27.8ns). In contrast, caching the *second* row is not as useful, since it yields only a small latency reduction from t_{RCDfar} (13.2ns) to $t_{RCDnear}$ (9.3ns).

Our second mechanism, *Wait-Minimized Caching* (WMC), caches only *wait-inducing rows*. These are rows that, while they are accessed, cause a large wait (t_{RCfar}) for the next access to a different row. More specifically, a row in the far segment is classified as wait-inducing if the next access to a different row arrives while the row is still being activated. WMC operates similarly to our SC mechanism except for the following differences. First, WMC copies a row from the far segment to the near segment *only if the row is wait-inducing*. Second, instead of evicting the LRU row from the near segment, WMC evicts the *least-recently wait-inducing* row. Third, when

[5] We expect the values to be of similar orders of magnitude for other designs.

a row is accessed from the near segment, it is updated as the *most-recently wait-inducing* row only if the access would have caused the next access to wait had the row been in the far segment. The memory controller is augmented with appropriate structures to keep track of the necessary information to identify wait-inducing rows (details omitted due to space constraints).

Mechanism 3: Benefit-Based Caching (BBC). Accessing a row in the near segment provides two benefits compared to accessing a row in the far segment: 1) reduced t_{RCD} (faster access) and 2) reduced t_{RC} (lower wait time for the subsequent access). Simple Caching (SC) and Wait-Minimized Caching (WMC) take into account only one of the two benefits. Our third mechanism, *Benefit-Based Caching* (BBC) explicitly takes into account both benefits of caching a row in the near segment. More specifically, the memory controller keeps track of a *benefit* value for each row in the near segment. When a near segment row is accessed, its benefit is incremented by the number of DRAM cycles saved due to reduced access latency and reduced wait time for the subsequent access. When a far-segment row is accessed, it is immediately promoted to the near segment, replacing the near-segment row with the least benefit. To prevent benefit values from becoming stale, on every eviction, the benefit for every row is halved. (Implementation details are omitted due to space constraints.)

5.2 Exposing Near Segment Capacity to the OS

Our second approach to leverage the TL-DRAM substrate is to expose the near segment capacity to the operating system. Note that simply replacing the conventional DRAM with our proposed TL-DRAM can potentially improve system performance due to the reduced $t_{RCDnear}$, t_{RCDfar}, and t_{RCnear}, while not reducing the available memory capacity. Although this mechanism incurs no additional complexity at the memory controller or the operating system, we find that the overall performance improvement due to this mechanism is low. To better exploit the benefits of the low access latency of the near segment, frequently accessed pages should be mapped to the near segment. This can be done by the hardware or by the OS. To this end, we describe two different mechanisms.

Exclusive Cache. In this mechanism, we use the near segment as an *exclusive* cache to the rows in the far segment. The memory controller uses one of the three mechanisms proposed in Section 5.1 to determine caching and eviction candidates. To cache a particular row, the data of that row is *swapped* with the data of the row to be evicted from the near segment. For this purpose, each subarray requires a *dummy-row* (D-row). To swap the data of the to-be-cached row (C-row) and to-be-evicted row (E-row), the memory controller simply performs the following three migrations:

C-row $\rightarrow$ D-row E-row $\rightarrow$ C-row D-row $\rightarrow$ E-row

The exclusive cache mechanism provides almost full main memory capacity (except one dummy row per subarray, $< 0.2\%$ loss in capacity) at the expense of two overheads. First, since row swapping changes the mappings of rows in both the near and the far segment, the memory controller must maintain the mapping of rows in both segments. Second, each swapping requires three migrations, which increases the latency of caching.

Profile-Based Page Mapping. In this mechanism, the OS controls the virtual-to-physical mapping to map frequently accessed pages to the near segment. The OS needs to be informed of the bits in the physical address that control the near-segment/far-segment mapping. Information about frequently accessed pages can either be obtained statically using compiler-based profiling, or dynamically using hardware-based profiling. In our evaluations, we show the potential performance improvement due to this mechanism using hardware-based profiling. Compared to the exclusive caching mechanism, this approach requires much lower hardware storage overhead.

6 Implementation Details & Further Analysis

6.1 Near Segment Row Decoder Wiring

To avoid the need for a large, monolithic row address decoder at each subarray, DRAM makes use of *predecoding*. Outside the subarray, the row address is divided into M sets of bits. Each set, N bits, is decoded into 2^N wires, referred to as $N:2^N$ predecoding.[6] The input to each row's wordline driver is then simply the logical AND of the M wires that correspond to the row's address. This allows the per-subarray row decoding logic to be simple and compact, at the cost of increasing the wiring overhead associated with row decoding at each subarray. As a result, wiring overhead dominates the cost of row decoding.

The inter-segment data transfer mechanism described in Sec 4.4 requires up to two rows to be activated in the same subarray at once, necessitating a second row decoder. However, since one of the two activated rows is always in the near segment, this second row decoder only needs to address rows in the near segment. For a near segment with 32 rows, a scheme that splits the 5 ($\log_2 32$) near segment address bits into 3-bit and 2-bit sets requires 12 additional wires to be routed to each subarray (3:8 predecoding + 2:4 predecoding). The corresponding die-size penalty is 0.33%, calculated based on the total die-size and wire-pitch derived from Vogelsang [39, 54].

6.2 Additional Storage in DRAM Controller

The memory controller requires additional storage to keep track of the rows that are cached in the near segment. In the inclusive caching mechanisms, each subarray contains a near segment of length N, serving as an N-way (fully-associative) cache of the far segment of length F. Therefore, each near-segment row requires a $\lceil \log_2 F \rceil$-bit tag. Hence, each subarray requires $N\lceil \log_2 F \rceil$ bits for tag storage, and a system with S subarrays requires $SN\lceil \log_2 F \rceil$ bits for tags. In the exclusive caching mechanism, row swapping can lead to any physical page within a subarray getting mapped to any row within the subarray. Hence, each row within the subarray requires the tag of the physical page whose data is stored in that row. Thus, each row in a subarray requires a $\lceil \log_2(N+F) \rceil$-bit tag, and a system with S subarrays requires $S(N+F)\lceil \log_2(N+F) \rceil$ bits for tags. In the system configuration we evaluate (described in Sec 7), $(N, F, S) = (32, 480, 256)$, the tag storage overhead is 9 KB for inclusive caching and 144 KB for exclusive caching.

SC and WMC additionally require $\log_2 N$ bits per near segment row for replacement information ($SN \log_2 N$ bits total), while our implementation of BBC uses an 8-bit benefit field per near segment row ($8SN$ bits total). For our evaluated system, these are overheads of 5 KB and 8 KB respectively.

6.3 Power Analysis

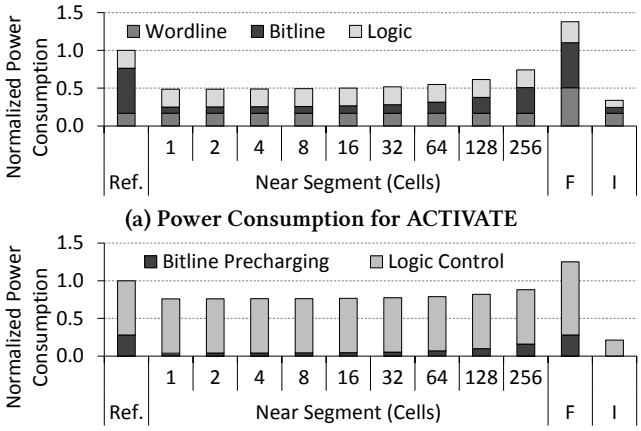
The non-I/O power consumption of the DRAM device can be broken into three dominant components: *i)* raising and lowering the wordline during ACTIVATE and PRECHARGE, *ii)* driving the bitline during ACTIVATE, and *iii)* transferring data from the sense-amplifiers to the peripherals. The first two of these components differ between a conventional DRAM and TL-DRAM, for two reasons:

Reduced Power Due to Reduced Bitline Capacitance in Near Segment. The energy required to restore a bitline is proportional to the bitline's capacitance. In TL-DRAM, the near segment has a lower capacitance than that of a conventional DRAM's bitline, resulting in decreased power consumption.

[6] N may differ between sets.

Additional Power Due to Isolation Transistors. The additional power required to control the isolation transistors when accessing the far segment is approximately twice that of raising the wordline, since raising the wordline requires driving one access transistor per bitline, while accessing the far segment requires driving two isolation transistors per bitline (Sec 4.3).

Using DRAM power models from Rambus and Micron [29, 39, 54], we estimate power consumption of TL-DRAM and conventional DRAM in Figure 13. Note that, while the near segment's power consumption increases with the near segment length, the far segment's power does not change as long as the total bitline length is constant. Our evaluations in Sec 8 take these differences in power consumption into account.



(a) Power Consumption for ACTIVATE

(b) Power Consumption for PRECHARGE

† Ref.: Long Bitline, F: Far Segment, I: Inter-Segment Data Transfer

Figure 13. Power Consumption Analysis

6.4 More Tiers

Although we have described TL-DRAM with two segments per subarray, one can imagine adding more isolation transistors to divide a subarray into more tiers, each tier with its own latency and power consumption characteristics. As a case study, we evaluated the latency characteristics of TL-DRAM with three tiers per subarray (near/middle/far with 32/224/256 cells). The normalized t_{RCD} and t_{RC} of the near/middle/far segments are 54.8%/70.7%/104.1% and 44.0%/77.8%/156.9%. While adding more tiers can enable more fine-grained caching and partitioning mechanisms, they come with the additional area cost of 3.15% per additional tier, additional power consumption to control the multiple isolation transistors, and logic complexity in DRAM and the DRAM controller.

7 Evaluation Methodology

Simulators. We developed a cycle-accurate DDR3-SDRAM simulator that we validated against Micron's Verilog behavioral model [28] and DRAMSim2 [42]. We use this memory simulator as part of a cycle-level in-house x86 multi-core simulator, whose front-end is based on Pin [26].

System Configuration. The evaluated system is configured as shown in Table 4. Unless otherwise stated, the evaluated TL-DRAM has a near segment size of 32 rows.

Parameters. DRAM latency is as calculated in Sec 4.2. DRAM dynamic energy consumption is evaluated by associating an energy cost with each DRAM command, derived using the tools [29, 39, 54] and the methodology given in Sec 6.3.

Benchmarks. We use 32 benchmarks from SPEC CPU2006, TPC [52], STREAM [1] and a *random* microbenchmark similar in behavior to GUPS [14]. We classify benchmarks whose performance is significantly affected by near segment size as *sensitive*, and all other benchmarks as *non-sensitive*. For single-core sensitivity studies, we report results that are averaged across all 32 benchmarks. We also present multi-programmed multi-core evaluations in Sec 8.4. For each multi-core workload group, we report results averaged across 16 workloads, generated by randomly selecting from specific benchmark groups (sensitive or non-sensitive).

Table 4. Evaluated System Configuration

Processor	5.3 GHz, 3-wide issue, 8 MSHRs/core, 128-entry instruction window
Last-Level Cache	64B cache line, 16-way associative, 512kB private cache slice/core
Memory Controller	64/64-entry read/write request queue, row-interleaved mapping, closed-page policy, FR-FCFS scheduling [41]
DRAM	2GB DDR3-1066, 1/2/4 channel (@1-core/2-core/4-core), 1 rank/channel, 8 banks/rank, 32 subarrays/bank, 512 rows/bitline t_{RCD} (unsegmented): 15.0ns, t_{RC} (unsegmented): 52.5ns
TL-DRAM	32 rows/near segment, 480 rows/far segment t_{RCD} (near/far): 8.2/12.1ns, t_{RC} (near/far): 23.1/65.8ns

Simulation and Evaluation. We simulate each benchmark for 100 million instructions. For multi-core evaluations, we ensure that even the slowest core executes 100 million instructions, while other cores continue to exert pressure on the memory subsystem. To measure performance, we use instruction throughput (IPC) for single-core systems and *weighted speedup* [49] for multi-core systems.

8 Results

8.1 Single-Core Results: Inclusive Cache

Fig 14 compares our proposed TL-DRAM based mechanisms (SC, WMC, and BBC) to the baseline with conventional DRAM. For each benchmark, the figure plots four metrics: *i)* performance improvement of the TL-DRAM based mechanisms compared to the baseline, *ii)* the number of misses per instruction in the last-level cache, *iii)* the fraction of accesses that are served at the row buffer, near segment and the far segment using each of the three proposed mechanisms, and *iv)* the power consumption of the TL-DRAM based mechanisms relative to the baseline. We draw three conclusions from the figure.

First, for most benchmarks, all of our proposed mechanisms improve performance significantly compared to the baseline. On average, SC, WMC and BBC improve performance by 12.3%, 11.3% and 12.8%, respectively, compared to the baseline. As expected, performance benefits are highest for benchmarks with high memory intensity (MPKI: Last-Level Cache Misses-Per-Kilo-Instruction) and high near-segment hit rate.

Second, all three of our proposed mechanisms perform similarly for most benchmarks. However, there are a few benchmarks where WMC significantly degrades performance compared to SC. This is because WMC only caches wait-inducing rows, ignoring rows with high reuse that cause few conflicts. BBC, which takes both reuse and wait into account, outperforms both SC and WMC. For example, BBC significantly improves performance compared to SC ($> 8\%$ for *omnetpp*, $> 5\%$ for *xalancbmk*) by reducing wait and compared to WMC ($> 9\%$ for *omnetpp*, $> 2\%$ for *xalancbmk*) by providing more reuse.

Third, BBC degrades performance only for the microbenchmark *random*. This benchmark has high memory intensity (MPKI = 40) and very low reuse (near-segment hit rate $< 10\%$). These two factors together result in frequent bank conflicts in the far segment. As a result, most requests experience the full

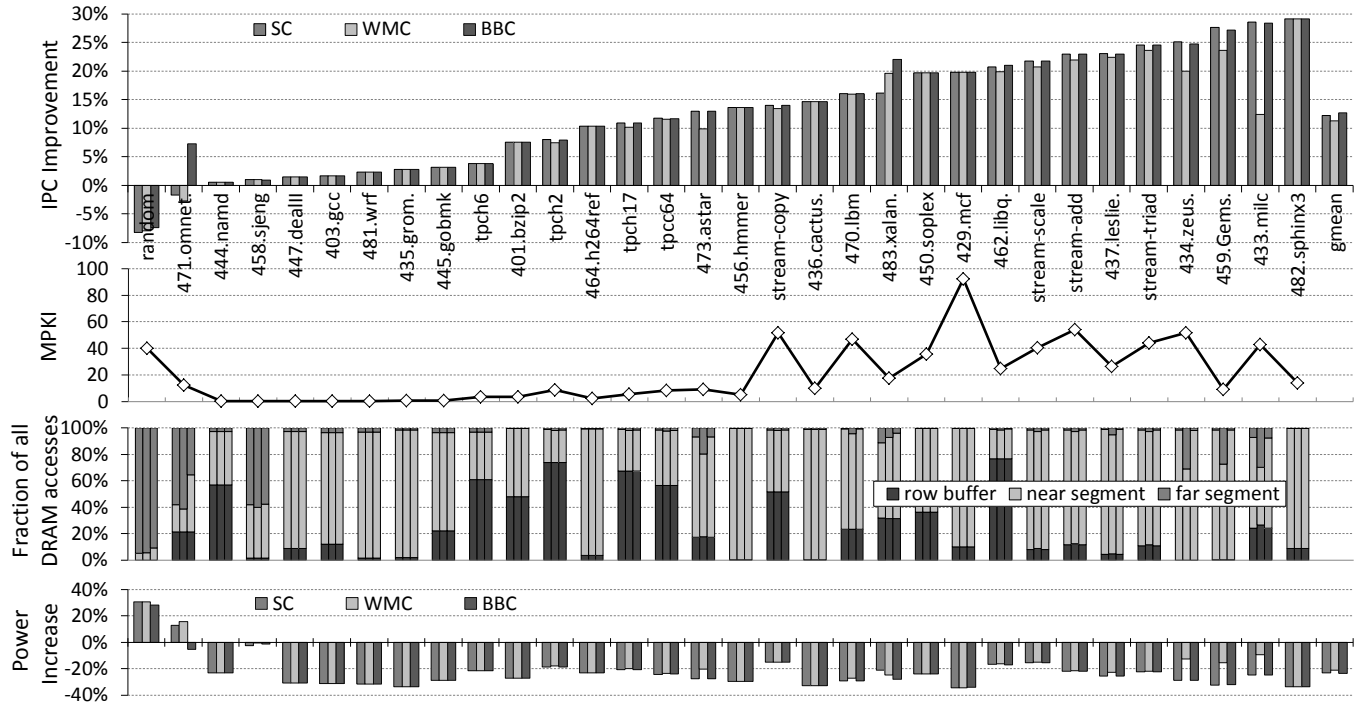


Figure 14. Single-core: IPC improvement, LLC MPKI, Fraction of accesses serviced at row buffer/near segment/far segment, Power consumption

penalty of the far segment's longer t_{RC}. We analyze this microbenchmark in detail in Sec 8.2.

Power Analysis. As described in Sec 6.3, power consumption is significantly lower for near-segment accesses, but higher for far-segment accesses, compared to accesses in a conventional DRAM. As a result, TL-DRAM achieves significant power savings when most accesses are to the near segment. The bottom-most plot of Fig 14 compares the power consumption of our TL-DRAM-based mechanisms to that of the baseline. Our mechanisms produce significant power savings (23.6% for BBC vs. baseline) on average, since over 90% of accesses hit in the row buffer or near segment for most benchmarks.

8.2 Effect of Far Segment Latency: Inclusive Cache

As we describe in Sec 4.1, the t_{RCD} of the far segment is *lower* than the t_{RCD} of conventional DRAM. Therefore, even if most of the accesses are to the far segment, if the accesses are sufficiently far apart such that t_{RC} is *not* a bottleneck, TL-DRAM can still improve performance. We study this effect using our *random* microbenchmark, which has very little data reuse and hence usually accesses the far segment. Fig 15 shows the performance improvement of our proposed mechanisms compared to the baseline with varying memory intensity (MPKI) of *random*. As the figure shows, when the memory intensity is low (< 2), the reduced t_{RCD} of the far segment dominates, and our mechanisms improve performance by up to 5%. However, further increasing the intensity of *random* makes t_{RC} the main bottleneck as evidenced by the degraded performance due to our proposed mechanisms.

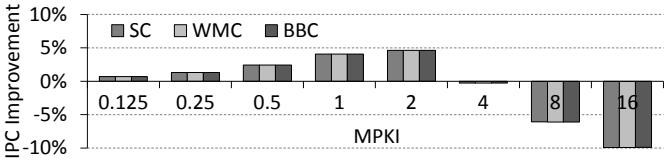


Figure 15. TL-DRAM Performance on Benchmark *random*

8.3 Sensitivity Results: Inclusive Cache

Sensitivity to Near Segment Capacity. The number of rows in each near segment presents a trade-off, as increasing the near segment's size increases its capacity but also increases its access latency. Fig 16 shows the average performance improvement of our proposed mechanisms over the baseline as we vary the near segment size. As expected, performance initially improves as the number of rows in the near segment is increased due to increased near segment capacity. However, increasing the number of rows per near segment beyond 32 reduces the performance benefits due to the increased near segment access latency.

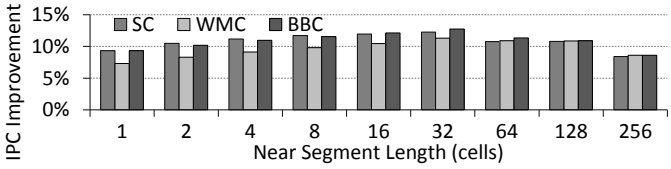
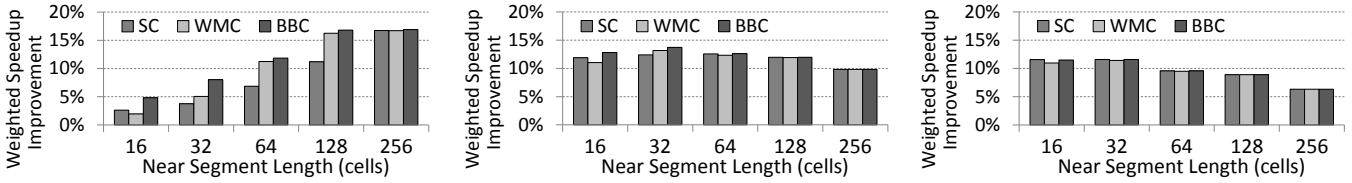


Figure 16. Varying Near Segment Capacity (Inclusive Cache)

Sensitivity to Channel Count. For 1-core systems with 1, 2 and 4 memory channels, TL-DRAM provides 12.8%, 13.8% and 14.2% performance improvement compared to the baseline. The performance improvement of TL-DRAM increases with increasing number of channels. This is because, with more channels, the negative impact of channel conflicts reduces and bank access latency becomes the primary bottleneck. Therefore, TL-DRAM, which reduces the average bank access latency, provides better performance with more channels.

8.4 Multi-Core Results: Inclusive Cache

Fig 17 shows the system performance improvement of our proposed mechanisms compared to baseline for three different workload categories. As expected, when both benchmarks are sensitive to near segment capacity (Fig 17a), the improvement in weighted speedup increases with increasing near segment capacity; when neither benchmark is sensitive to the near segment capacity, the weighted speedup improvement decreases



(a) Sensitive - Sensitive (b) Sensitive - Non-Sensitive (c) Non-sensitive - Non-Sensitive

Figure 17. System Performance: 2-core, Inclusive Cache

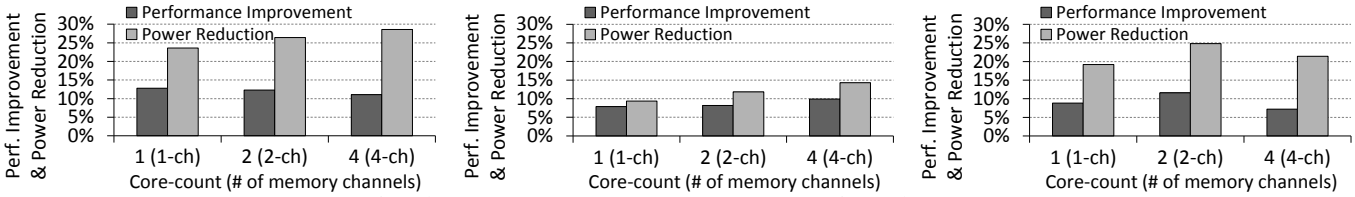


Figure 18. Inclusive Cache Analysis (BBC) **Figure 19. Exclusive Cache Analysis (WMC)** **Figure 20. Profile-Based Page Mapping**

with increasing near segment capacity due to the increased near segment access latency (Fig 17c). For almost all workload categories and near segment capacities, BBC performs comparably to or significantly better than SC, emphasizing the advantages of our benefit-based near segment management policy. As shown in Fig 18, on average, BBC improves weighted speedup by 12.3% and reduces power consumption by 26.4% compared to the baseline. We observe similar trends on 4-core systems, where BBC improves weighted speedup by 11.0% and reduces power consumption by 28.6% on average.

8.5 Exclusive Cache

Fig 19 shows the performance improvement of the TL-DRAM with the exclusive caching mechanism (Section 5.2) and 32 rows in the near segment over the baseline. For 1-/2-/4-core systems, TL-DRAM with exclusive caching improves performance by 7.9%/8.2%/9.9% and reduces power consumption by 9.4%/11.8%/14.3%. The performance improvement due to exclusive caching is lower compared to that of inclusive caching due to the increased caching latency, as explained in Section 5.2. Unlike inclusive caching, where BBC outperforms SC and WMC, WMC performs the best for exclusive caching. This is because unlike BBC and SC that cache any row that is accessed in the far segment, WMC caches a row only if it is wait-inducing. As a result, WMC reduces bank unavailability due to the increased caching latency.

8.6 Profile-Based Page Mapping

Fig 20 shows the performance improvement of TL-DRAM with profile-based page mapping over the baseline. The evaluated memory subsystem has 64 rows in the near segment. Therefore, the top 64 most frequently accessed rows in each subarray, as determined by a profiling run, are allocated in the near segment. For 1-/2-/4-core systems, TL-DRAM with profile-based page mapping improves performance by 8.9%/11.6%/7.2% and reduces power consumption by 19.2%/24.8%/21.4%. These results indicate a significant potential for such a profiling based mapping mechanism. We leave a more rigorous evaluation of such a profiling based mapping mechanism to future work.

9 Related Work

Short Bitlines. Some specialized DRAMs reduce access latency by reducing the number of cells-per-bitline, e.g. Micron's RLDRAM [27] and Fujitsu's FCRAM [45]. However, as discussed in Secs 1 and 3, this requires the addition of more sense-amplifiers, incurring an area overhead of 30–80% [21, 45]. This results in significantly higher cost-per-bit.

Cached DRAM. Cached DRAM [10, 12, 13, 15, 20, 34, 44, 57, 59] adds an SRAM cache to DRAM chips. However, SRAM-cached DRAM approaches have two major limitations. First, an SRAM cache incurs significant area overhead; using CACTI-D [51], we estimate that an SRAM-cached DRAM with equivalent capacity to a TL-DRAM with 32 rows/near segment would incur 145.3% area overhead. Second, transferring data between the DRAM array and the SRAM cache requires use of the relatively narrow global I/O bus within the DRAM chip leading to high caching latency. In contrast, TL-DRAM incurs minimal area overhead of 3.15% (Sec 4.3) and facilitates fast in-DRAM transfer of data between segments (Sec 4.4). With the same amount of cache capacity, the performance improvement of Cached DRAM (8.3%) is less compared to that of TL-DRAM (12.8%) for 1-core systems. This is primarily due to the large caching latency incurred by Cached DRAM.

Increased DRAM Parallelism. Kim et al. [24] propose schemes to parallelize accesses to different subarrays within a bank, thereby overlapping their latencies. Multiple works [2, 55, 60] have proposed partitioning a DRAM rank into multiple independent rank subsets that can be accessed in parallel [3]. All of these proposals reduce the frequency of row-buffer conflicts, but not their latency, and can be applied in conjunction with our TL-DRAM substrate.

DRAM Controller Optimizations. Sudan et al. [50] propose a mechanism to co-locate heavily reused data in the same row, with the goal of improving row-buffer locality. A large body of prior work has explored DRAM access scheduling in the controller (e.g. [4, 9, 23, 22, 32, 31]), to mitigate inter-application interference and queueing latencies in multi-core systems. Our TL-DRAM substrate is orthogonal to all of these approaches.

Segmented Bitlines in Other Technologies. Prior works [11, 40, 48] have proposed the use of segmented bitlines in other memory technologies, like SRAM caches and flash memory. In contrast, this is, to our knowledge, the first work to propose the use of segmented bitlines in DRAM. Our approach and the resulting tradeoffs are different as we take into account characteristics and operation that are unique to DRAM, such as DRAM-specific subarray organization, sensing mechanisms, and timing constraints.

Caching and Paging Techniques. We leverage our TL-DRAM substrate to cache data in the near segment. Many previous works (e.g., [8, 17, 18, 36, 37, 38, 46]) have proposed sophisticated cache management policies in the context of processor SRAM caches. These techniques are potentially applicable to TL-DRAM to manage which rows get cached in the near segment. The TL-DRAM substrate also allows the operating system to exploit the asymmetric latencies of the near and far segments in software through intelligent page placement and migration techniques [53, 7]. We leave the investigation of such caching and paging techniques on TL-DRAM to future work.

10 Conclusion

Existing DRAM architectures present a trade-off between cost-per-bit and access latency. One can either achieve low cost-per-bit using long bitlines or low access latency using short bitlines, but not both. We introduced Tiered-Latency DRAM (TL-DRAM), a DRAM architecture that provides both low latency (in the common case) and low cost-per-bit. The key idea behind TL-DRAM is to segment a long bitline using an isolation transistor, creating a segment of rows with low access latency while keeping cost-per-bit on par with commodity DRAM.

We presented mechanisms that take advantage of our TL-DRAM substrate by using its low-latency segment as a hardware-managed cache. Our most sophisticated cache management algorithm, Benefit-Based Caching (BBC), selects rows to cache that maximize access latency savings. We show that our proposed techniques significantly improve both system performance and energy efficiency across a variety of systems and workloads. We conclude that TL-DRAM provides a promising low-latency and low-cost substrate for building main memories, on top of which existing and new caching and page allocation mechanisms can be implemented to provide even higher performance and higher energy efficiency.

Acknowledgments

Many thanks to Uksong Kang, Hak-soo Yu, Churoo Park, Jung-Bae Lee, and Joo Sun Choi from Samsung, and Brian Hirano from Oracle for their helpful comments. We thank the reviewers for their feedback. We thank the SAFARI group members for the feedback and stimulating research environment they provide. We acknowledge the support of our industrial partners: AMD, HP Labs, IBM, Intel, Oracle, Qualcomm, and Samsung. This research was also partially supported by grants from NSF (Awards CCF-0953246 and CCF-1212962), GSRC, and the Intel URO Memory Hierarchy Program. Donghyuk Lee is supported in part by a Ph.D. scholarship from Samsung.

References

[1] STREAM Benchmark. http://www.streambench.org/.
[2] J. H. Ahn et al. Multicore DIMM: an energy efficient memory module with independently controlled DRAMs. *IEEE CAL*, January 2009.
[3] J. H. Ahn et al. Improving system energy efficiency with memory rank subsetting. *ACM TACO*, Mar. 2012.
[4] R. Ausavarungnirun et al. Staged memory scheduling: achieving high performance and scalability in heterogeneous systems. In *ISCA*, 2012.
[5] S. Borkar and A. A. Chien. The future of microprocessors. In *CACM*, 2011.
[6] D. Burns et al. An RLDRAM II Implementation of a 10Gbps Shared Packet Buffer for Network Processing. In *AHS*, 2007.
[7] R. Chandra et al. Scheduling and page migration for multiprocessor compute servers. In *ASPLOS*, 1994.
[8] J. Collins and D. M. Tullsen. Hardware identification of cache conflict misses. In *MICRO*, 1999.
[9] E. Ebrahimi et al. Parallel application memory scheduling. In *MICRO*, 2011.
[10] Enhanced Memory Systems. Enhanced SDRAM SM2604, 2002.
[11] K. Ghose and M. B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *ISLPED*, 1999.
[12] C. A. Hart. CDRAM in a unified memory architecture. In *Compcon Spring '94, Digest of Papers*, 1994.
[13] H. Hidaka et al. The cache DRAM architecture: A DRAM with an on-chip cache memory. *IEEE Micro*, March 1990.
[14] HPC Challenge. GUPS. http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/.
[15] W.-C. Hsu and J. E. Smith. Performance of cached DRAM organizations in vector supercomputers. In *ISCA*, 1993.
[16] ITRS. International Technology Roadmap for Semiconductors: Process Integration, Devices, and Structures. http://www.itrs.net/Links/2007ITRS/Home2007.htm, 2007.
[17] A. Jaleel et al. High performance cache replacement using re-reference interval prediction. In *ISCA*, 2010.
[18] T. Johnson et al. Run-time cache bypassing. *IEEE TC*, 1999.
[19] T. S. Jung. Memory technology and solutions roadmap. http://www.sec.co.kr/images/corp/ir/irevent/techforum_01.pdf, 2005.
[20] G. Kedem and R. P. Koganti. WCDRAM: A fully associative integrated cached-DRAM with wide cache lines. *Duke*, (CS-1997-03), 1997.
[21] B. Keeth et al. *DRAM Circuit Design. Fundamental and High-Speed Topics*. Wiley-IEEE Press, 2007.
[22] Y. Kim et al. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *HPCA*, 2010.
[23] Y. Kim et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *MICRO*, 2010.
[24] Y. Kim et al. A case for exploiting subarray-level parallelism (SALP) in DRAM. In *ISCA*, 2012.
[25] T. Kimura et al. 64Mb 6.8ns random row access DRAM macro for ASICs. In *ISSCC*, 1999.
[26] C.-K. Luk et al. Pin: building customized program analysis tools with dynamic instrumentation. In *PLDI*, 2005.
[27] Micron. RLDRAM 2 and 3 Specifications. http://www.micron.com/products/dram/rldram-memory.
[28] Micron. Verilog: DDR3 SDRAM Verilog model. http://www.micron.com/get-document/?documentId=808.
[29] Micron. DDR3 SDRAM System-Power Calculator, 2010.
[30] Y. Moon et al. 1.2V 1.6Gb/s 56nm 6F2 4Gb DDR3 SDRAM with hybrid-I/O sense amplifier and segmented sub-array architecture. *ISSCC*, 2009.
[31] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *MICRO*, 2007.
[32] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *ISCA*, 2008.
[33] S. Narasimha et al. High performance 45-nm SOI technology with enhanced strain, porous low-k BEOL, and immersion lithography. In *IEDM*, 2006.
[34] NEC. Virtual Channel SDRAM uPD4565421, 1999.
[35] D. A. Patterson. Latency lags bandwith. *Commun. ACM*, Oct. 2004.
[36] T. Piquet et al. Exploiting single-usage for effective memory management. In *ACSAC*, 2007.
[37] M. K. Qureshi et al. Adaptive insertion policies for high performance caching. In *ISCA*, 2007.
[38] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO*, 2006.
[39] Rambus. DRAM Power Model. http://www.rambus.com/energy, 2010.
[40] R. Rao et al. Exploiting non-uniform memory access patterns through bitline segmentation. In *WMPI*, 2006.
[41] S. Rixner et al. Memory access scheduling. In *ISCA*, 2000.
[42] P. Rosenfeld et al. DRAMSim2: A cycle accurate memory system simulator. *IEEE CAL*, January 2011.
[43] Samsung. DRAM Data Sheet. http://www.samsung.com/global/business/semiconductor/product.
[44] R. H. Sartore et al. Enhanced DRAM with embedded registers. U.S. patent, 1999. Patent number 5887272.
[45] Y. Sato et al. Fast Cycle RAM (FCRAM); a 20-ns random row access, pipe-lined operating DRAM. In *Symposium on VLSI Circuits*, 1998.
[46] V. Seshadri et al. The Evicted-Address Filter: A unified mechanism to address both cache pollution and thrashing. In *PACT*, 2012.
[47] S. M. Sharroush et al. Dynamic random-access memories without sense amplifiers. In *Elektrotechnik & Informationstechnik*, 2012.
[48] J. M. Sibigtroth et al. Memory bit line segment isolation. U.S. patent, 2006. Patent number 7042765.
[49] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *ASPLOS*, 2000.
[50] K. Sudan et al. Micro-pages: Increasing DRAM efficiency with locality-aware data placement. In *ASPLOS*, 2010.
[51] S. Thoziyoor et al. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *ISCA*, 2008.
[52] Transaction Processing Performance Council. http://www.tpc.org/.
[53] B. Verghese et al. Operating system support for improving data locality on cc-numa compute servers. In *ASPLOS*, 1996.
[54] T. Vogelsang. Understanding the energy consumption of dynamic random access memories. In *MICRO*, 2010.
[55] F. Ware and C. Hampel. Improving power and data efficiency with threaded memory modules. In *ICCD*, 2006.
[56] M. V. Wilkes. The memory gap and the future of high performance memories. *SIGARCH Comput. Archit. News*, March 2001.
[57] W. A. Wong and J.-L. Baer. DRAM caching. Technical Report UW-CSE-97-03-04, 1997.
[58] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, March 1995.
[59] Z. Zhang et al. Cached DRAM for ILP processor memory access latency reduction. *IEEE Micro*, July 2001.
[60] H. Zheng et al. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *MICRO*, 2008.