

# A Comprehensive Analytical Performance Model of DRAM Caches

Nagendra Gulur,  
Mahesh Mehendale  
Texas Instruments  
Bangalore, India  
nagendra@ti.com,  
m-mehendale@ti.com

Ramaswamy Govindarajan  
Indian Institute of Science  
Bangalore, India  
govind@serc.iisc.ernet.in

## ABSTRACT

Stacked DRAM promises to offer unprecedented capacity, and bandwidth to multi-core processors at moderately lower latency than off-chip DRAMs. A typical use of this abundant DRAM is as a large last level cache. Prior research works are divided on how to organize this cache and the proposed organizations fall into one of two categories: (i) as a *Tags-In-DRAM* organization with the cache organized as small blocks (typically 64B) and metadata (tags, valid, dirty, recency and coherence bits) stored in DRAM, and (ii) as a *Tags-In-SRAM* organization with the cache organized as larger blocks (typically 512B or larger) and metadata stored on SRAM. *Tags-In-DRAM* organizations tend to incur higher latency but conserve off-chip bandwidth while the *Tags-In-SRAM* organizations incur lower latency at some additional bandwidth. In this work, we develop a unified performance model of the DRAM-Cache that models these different organizational styles. The model is validated against detailed architecture simulations and shown to have latency estimation errors of 10.7% and 8.8% on average in 4-core and 8-core processors respectively. We also explore two insights from the model: (i) the need for achieving very high hit rates in the metadata cache/predictor (commonly employed in the *Tags-In-DRAM* designs) in reducing latency, and (ii) opportunities for reducing latency by load-balancing the DRAM Cache and main memory.

## 1. INTRODUCTION

Stacked DRAMs offer unprecedented capacity and bandwidth by allowing many layers of DRAM storage to be vertically stacked up and enabling access to these cells via high bandwidth channels [25]. The on-chip integration also reduces latency compared to off-chip DRAM storage. While the capacity of stacked DRAM (100s of MB) is far higher than SRAM (a few MB), the DRAM storage is still not sufficient to hold the entire working set of multi-core workloads. Thus the stacked DRAM is typically employed as a large last-level cache, backed by main memory. The main memory could be made up of DRAM or non-volatile technologies such as PCM [24] or STT-MRAM [14]. In this scenario, the DRAM Cache organization plays a pivotal role in processor performance since it

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
*ICPE'15*, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.  
Copyright © 2015 ACM 978-1-4503-3248-4/15/01 ...\$15.00.  
<http://dx.doi.org/10.1145/2668930.2688044>.

services the bulk of the memory requests coming from the last-level SRAM cache (abbreviated *LLSC* in the rest of this paper).

Due to their large capacity, DRAM Caches require a large amount of metadata (tags, valid, dirty, recency, coherence bits). The size of this metadata can run into megabytes (for example, a 256MB Cache with 64B blocks and 4B metadata per block needs 16MB for metadata). Two different organizational styles have been proposed to manage metadata: *Tags-In-SRAM* and *Tags-In-DRAM*. *Tags-In-SRAM* organizations [12, 13] reduce the metadata overhead by using larger cache blocks (typically 512B or larger). The (small-sized) metadata is maintained on SRAM. However, the larger block size may waste off-chip bandwidth by fetching un-used data. In *Tags-In-DRAM* organizations [11, 16, 19], the metadata is stored alongside data in DRAM Cache rows (pages). This organization incurs a higher access latency since both tags and data have to be accessed from DRAM. Thus a small metadata cache/predictor is typically employed to reduce or avoid DRAM Cache accesses for tags.

In this work, we develop a unified analytical performance model of the DRAM Cache that spans both these styles of organizations taking into account key parameters such as cache block size, tag cache/predictor hit rate, DRAM Cache timing values, off-chip memory timing values and salient workload characteristics. The model estimates average miss penalty and bandwidth seen by the *LLSC*. Through detailed simulation studies, we validate this model for accuracy, with resulting latency estimation errors of 10.7% and 8.8% on average in 4-core and 8-core workloads respectively. Using the model we draw two interesting and useful insights:

**Role of Tag Cache/Predictor:** We show that the hit rate of the auxiliary tag cache/predictor (abbreviated *Tag-Pred*) is crucial to overall latency reduction. We also show that this tag cache (*Tag-Pred*) needs to achieve very high hit rate in order for the *Tags-In-DRAM* designs to out-perform *Tags-In-SRAM* designs.

**Tapping Main Memory Bandwidth:** Counter to intuition, we show that performance can be improved by sacrificing DRAM Cache hits to a certain extent. Sacrificing the hits allow balancing the utilization of on-chip (DRAM Cache) and off-chip memory bandwidth which can lead to overall latency reduction.

## 2. BACKGROUND

### 2.1 DRAM Cache Overview

By virtue of stacking and the inherent density of DRAM, a DRAM cache provides a large capacity (typically 64MB to even gigabytes) offering an unprecedented opportunity to hold critical workload

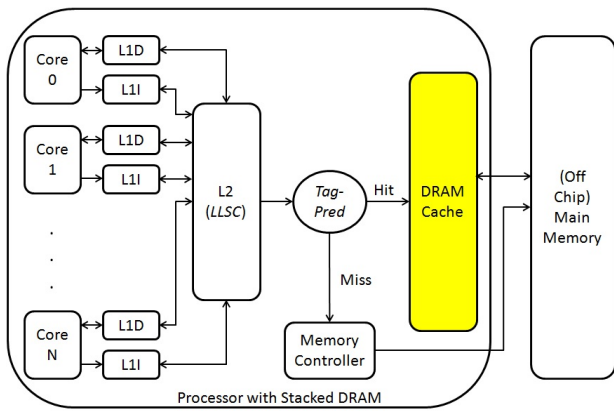


Figure 1: Overview of a Multi-core Processor with DRAM Cache

data on chip. The DRAM cache is typically organized as a last level shared cache behind a hierarchy of SRAM caches. A DRAM cache offers such large capacity caches at lower power unlike L1, L2 caches that are implemented using SRAM. However DRAM cache design requires careful attention to access latency since a typical DRAM access requires activating a row of cells, sensing the stored charge on these capacitors and finally transmitting the sensed data over a bus. Since row activation has drained the corresponding capacitors, a *precharge* operation is required to restore the charge back on these capacitors.

Figure 1 provides an organizational overview of a multi-core processor comprising 2 levels of SRAM caches, followed by a DRAM Cache and off-chip main memory. The DRAM Cache services misses and write-backs from the last level SRAM Cache (LLSC). A *Tag-Pred* is typically employed to make a quick decision of whether to access the DRAM Cache (if the data is predicted to be present in it) or go directly to main memory.

The logical organization and functionality of DRAM caches is similar to traditional SRAM caches. For the purposes of this work, we assume that it is organized as a  $N$  way set-associative cache with block size that is  $B_s$  times the size of the CPU LLC block size. Note that  $B_s = 1$  models *Tags-In-DRAM* organizations while larger values (typically  $B_s = 8$  or 16) model *Tags-In-SRAM* organizations. DRAM Cache misses fetch the cache block from main memory. We assume a writeback cache and that only dirty sub-blocks are written back upon eviction. This assumption reflects the real-world implementation that DRAM Caches use - namely, when a dirty block is evicted, only the portions that are actually dirty (modified) are written to main memory. For example, a cache organized at 512B block size ( $B_s = 8$ ) will write back only those 64B sub-blocks that are modified. This is done to conserve bandwidth and energy in the main memory system.

Since the data is on DRAM the access incurs high latency (comprising row activation, column access, and finally precharge) and thus several data layout organizations have been evaluated [11–13, 16, 19]. We summarize the key design considerations below that we need to take into account in the model.

**Metadata Storage:** The large capacities offered by DRAM caches incur high metadata (tags, valid, dirty bits, recency, coherence bits etc) storage requirements which can run into multiple megabytes. Obviously committing this much storage in SRAM is costly and energy expensive<sup>1</sup>. *Tags-In-DRAM* organizations [11, 16, 19] propose

<sup>1</sup>The tag storage overhead may even exceed the total size of the last-level SRAM cache.

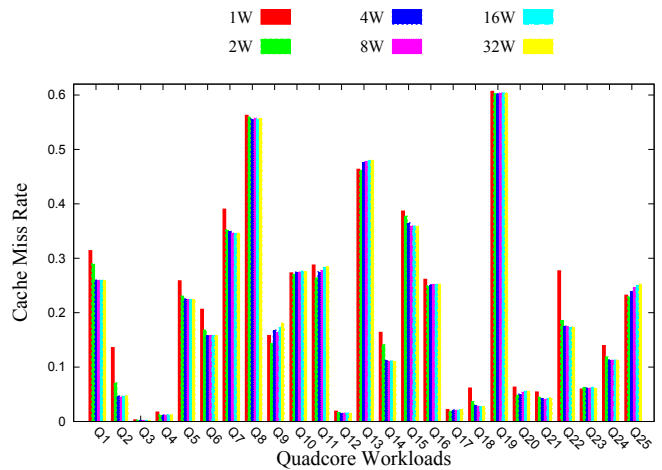


Figure 2: Impact of Set Associativity on DRAM Cache Hit Rate

to store the set metadata in the same DRAM rows as data thereby ensuring that one DRAM row activation is sufficient to retrieve both tags and data. In case of a DRAM cache miss, the latency of access is a sum of the DRAM cache tag look up time followed by the main memory access time. Since this degrades the access latency of cache misses, nearly all of these organizations propose the use of a *tag cache/predictor* structure (*Tag-Pred*) in SRAM to quickly evaluate if the access is a hit in the cache.

In the *Tags-In-SRAM* organizations [12, 13] the metadata is held in SRAM and provides faster tag lookup. Typically, these organizations employ large block sizes (typically 1KB or larger) to reduce storage overhead.

**Cache Block Size:** Typically upper-level caches (L1, L2) employ a small block size ( $\approx 64B$  or  $128B$ ) to capture spatial locality as well as to ensure low cost of a line fill. With large DRAM caches however larger block sizes may be gainfully employed to reduce metadata storage overhead (especially when it is on SRAM) as well as to leverage inherent spatial locality in workloads.

**Set Associativity:** A  $k$ -way mapped cache organizes each set to hold  $k$  blocks of data. Larger values of  $k$  reduce potential conflicts caused by addresses mapping to the same sets. However, in the context of DRAM caches, we find that associativity does not have a significant bearing on hit rate – an observation that other researchers have also made [10, 19]. Figure 2 plots the cache miss rates achieved at six different associativities (1-way, 2-way, 4-way, 8-way, 16-way, and 32-way) for several quad-core workloads in a 128MB cache. Except for workloads Q2 and Q22 that show noticeable miss rate reduction from 1-way to 2-way, all the others show no significant reduction with higher associativity (average reduction in miss rate in the 32-way organization over the direct-mapped organization is 4.3%). Thus, in our model we assume that the hit rate is independent of set associativity.

**Row-Buffer Hit Rate:** If accesses map to currently open pages in DRAM cache banks, then they can be serviced quickly (termed *row-buffer hits*). Row-buffer hit rate is governed by spatial locality in the access stream as well as by how adjacent cache blocks are mapped to DRAM cache sets and pages. Exploiting locality plays an important role in overall latency reduction.

The above design aspects interact to influence arrival rate, cache hit rate, DRAM row-buffer hit rate and thus the resulting latencies at the cache and main memory. Next we present an overview of the underlying DRAM performance model that we use in this work.

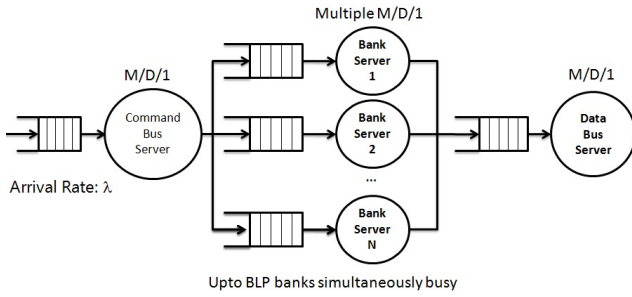


Figure 3: Queuing Model of Memory Controller as a 3-Stage Network of Queues

## 2.2 ANATOMY Overview

Our proposed performance model of the DRAM Cache is based on *ANATOMY* - an analytical model of memory performance in [7]. *ANATOMY* is the substrate for the DRAM Cache model that we develop in this work. *ANATOMY* has two key components that work together:

- A queuing model of memory that models in detail the key technological characteristics and design choices. The service times in this queuing model are parameterized by workload characteristics.
- A trace based analytical model that estimates key workload characteristics, namely arrival rate, row-buffer hit rate (*RBH*), bank-level parallelism (*BLP*) and request spread (*S*), that are used as inputs to the queuing model to estimate memory performance.

The queuing model of *ANATOMY* considers a memory system  $\mathcal{M}$  with  $M$  memory controllers. Each controller has a single channel and manages a memory system consisting of  $D$  DIMMs, each consisting of  $R$  ranks, and each rank having  $B$  banks. In this work, we do not model the rank parameters in detail<sup>2</sup> and treat that there is a total of  $N = D \times R \times B$  banks. Each channel has a command bus and data bus.

We summarize the actions from the time a request reaches the memory controller to the time the required data is sent back. The controller selects one of several queued requests based on a scheduling policy. For simplicity, we assume the *FCFS* scheduling policy here but as demonstrated in [7], other schedulers can be incorporated into the model. The controller has to issue a series of commands to the memory to perform access (read/write). Thus the command bus is a “server” that each request uses for one or more cycles. Once a command is issued, the memory bank to which that command was addressed performs the requested operation (precharge, row activation or column access). During this time the memory bank is busy and can not service other commands. Each bank is thus modeled as a server. Since there are  $N$  banks in the memory system, we model them as  $N$  servers. Finally, once the memory bank has put the data on the memory bus, the burst of data reaches the memory controller taking a few bus clock cycles. This final step of data transfer is modeled as the third stage server. An overview of this 3-stage network-of-queues model is presented in Figure 3. A system with multiple memory controllers is modeled using a 3-stage network for each controller.

### 2.2.1 3-stage Network of Queues Model

<sup>2</sup>We observe that the rank parameters, such as the rank-to-rank switching delay do not significantly affect the memory performance.

Each server is modeled as an  $M/D/1$  server, where the inter-arrival times are assumed to be exponentially distributed and the service time is deterministic<sup>3</sup>.

### 2.2.2 Stage 1: Command Bus

The command bus server captures issuing of necessary commands to the memory banks. We assume that the inter-arrival times of memory requests are exponentially distributed with a mean  $\frac{1}{\lambda}$ . As we consider multi-programmed workloads (details in Section 4), considering the interleaved nature of memory requests from the various programs, the assumption that the arrival process is Markovian is a reasonable approximation. The arrival rate  $\lambda$  is a characteristic of the application/workload and is estimated from a trace of memory requests issued by the *LLSC*. As explained earlier, based on whether an access turns into a row-buffer hit or a miss, the command bus issues either one (column access) or three (precharge, row-activate and column access) commands respectively to the corresponding DRAM bank. The time required to send any one command is fixed, and equal to one cycle of memory clock ( $t_{CK}$ ). Hence the average service time at the command bus can be approximated as a function of Row-buffer Hit rate (*RBH*), the fraction of requests that experience a row-buffer hit. *RBH* is primarily a workload characteristic with some design parameters like page size affecting it. For a given *RBH* value of  $R$ , the average service time required by the command bus is  $(R \times 1 + (1 - R) \times 3) \times t_{CK}$ . Since *RBH* is a workload specific constant, the average service time required can be treated as fixed.

Using the queuing theory result for the  $M/D/1$  queue [17], the queue delay at the command bus is given by:

$$QD_{Cmd\_Bus} = \frac{1}{2\mu_{cmd}} \frac{\rho_{cmd}}{(1 - \rho_{cmd})} \quad (1)$$

where  $\mu_{cmd} = \frac{1}{(R \times 1 + (1 - R) \times 3) \times t_{CK}}$  and  $\rho_{cmd} = \frac{\lambda}{\mu_{cmd}}$ .

### 2.2.3 Stage 2: Memory Banks

The bank servers take into account the key memory technology-specific timing parameters as well as the inherent parallelism present in a multi-bank memory.

In real memory systems, the number of banks that operate in parallel depends to a great extent on the amount of parallelism found in the memory accesses made by the workload. This workload characteristic is commonly referred to as Bank Level Parallelism (*BLP*)<sup>4</sup>. Note that in real memory, the requests are queued at the memory controller in bank-specific queues [20] until the bank becomes available. The functioning of each bank is assumed to be independent of other banks<sup>5</sup>. Thus we treat this stage as a collection of  $M/D/1$  queues operating in parallel (rather than as a single  $M/D/N$  queue). Multiple  $M/D/1$  servers enable modeling the concurrency of many banks simultaneously servicing requests.

As the service time of stage 1, the command bus server, is really small, we make the simplifying assumption that the input process at the second stage is also Markovian [17], with the same mean  $\lambda$ . Since the memory system typically has more banks ( $N$ ) than currently active ( $B$ ), a fraction of the incoming requests may go to idle banks. Such requests do not have to queue up as their banks are idle. This fraction of requests that go to idle banks is called the *Request Spread* (denoted  $S$ ). Thus only the rest of the requests

<sup>3</sup>While the output of an  $M/D/1$  process is not Markovian, in practice this approximation works well (see [17])

<sup>4</sup>Some of the memory design choices also have an impact on *BLP*.

<sup>5</sup>Except for peak-power limiting timing constraints such as  $T_{FAW}$ , the banks operate pretty much independently.

(fraction  $(1 - S)$ ) incur queueing delays. Thus the average arrival rate to each busy bank is modeled as:  $\lambda_{busy\_bank} = \frac{((1-S)*\lambda)}{B}$ .

Next, we consider the average service time of a request in one of the banks. The service time depends on whether the actual request turns into a row-buffer hit or miss. In the case of a row-buffer hit, the time required is the column access latency ( $t_{CL}$ ). If the access turns into a row-buffer miss, then the time required is a sum of the time required to complete precharge ( $t_{PRE}$ ), activate ( $t_{ACT}$ ) and column access ( $t_{CL}$ ). Hence with the application locality being characterized by a RBH of  $R$ , the average service time for a request is  $(t_{CL} \times R + (t_{PRE} + t_{ACT} + t_{CL}) \times (1 - R))$ .

The important thing to note regarding the service time computations is that the values for  $t_{CL}$ ,  $t_{PRE}$  and  $t_{ACT}$  are technology specific. Hence choice with respect to technology (like DDR3, DDR4, PCM or STT-MRAM) can be captured here by choosing appropriate latencies for the various actions. Thus our DRAM Cache model supports different types of main memory models - not just DRAM.

The queue delay at each bank is given by:

$$QD_{Bank} = \frac{1}{2\mu_{bank}} \frac{\rho_{bank}}{(1 - \rho_{bank})} \quad (2)$$

where  $\mu_{bank} = \frac{1}{(R \times t_{CL} + (1-R) \times (t_{PRE} + t_{ACT} + t_{CL}))}$  and  $\rho_{bank} = \frac{\lambda_{busy\_bank}}{\mu_{bank}}$ .

### 2.2.4 Stage 3: Data Bus

Data is transferred to/from the memory in a burst. A data burst leverages the open row-buffer and a large burst amortizes the cost of the row activation. The size of the burst (measured in number of clock cycles) is denoted  $BL$ . This data transfer takes a fixed time of  $BL \times t_{CK}$ .

The queue delay at this stage is given by:

$$QD_{Data} = \frac{1}{2\mu_{data}} \frac{\rho_{data}}{(1 - \rho_{data})} \quad (3)$$

where  $\mu_{data} = \frac{1}{BL \times t_{CK}}$  and  $\rho_{data} = \frac{\lambda}{\mu_{data}}$ .

### 2.2.5 Estimation of Workload Characteristics

The four workload characteristics ( $\lambda$ ,  $R$ ,  $B$ ,  $S$ ) are estimated from a time-annotated trace of memory requests issued by the last-level cache ( $LLSC$  in our case). Details of these estimations and their accuracy are presented in [7].

### 2.2.6 ANATOMY Summary

The *ANATOMY* performance model provides an estimate of average latency and peak bandwidth as a function of the memory technology, memory organization and workload characteristics. The average latency is given by:

$$\begin{aligned} Lat_{Avg} = & \frac{1}{\mu_{Cmd\_Bus}} + QD_{Cmd\_Bus} \\ & + \frac{1}{\mu_{bank}} + QD_{Bank} \\ & + \frac{1}{\mu_{data}} + QD_{Data} \end{aligned} \quad (4)$$

The peak bandwidth achievable by the memory system (per controller) is limited by the smallest of the three service stages and is given by:

$$Peak\_BW = \min(\mu_{Cmd\_Bus}, N * \mu_{bank}, \mu_{data}) \quad (5)$$

The model was validated against detailed simulations of 4-, 8- and 16-core workloads with average errors of 8.1%, 4.1%, 9.7% respectively.

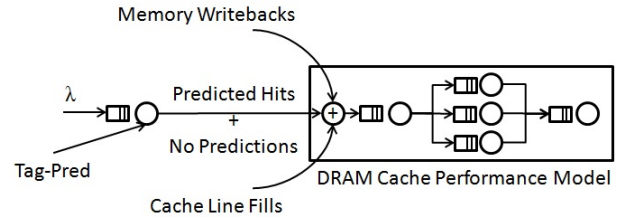


Figure 4: *ANATOMY* augmented with the Tag Cache/Predictor (*Tag-Pred*) Server

## 3. MODEL

Our model is based on the *ANATOMY* performance model. We first present the construction of the DRAM Cache model that provides an analytical estimation of latency seen at the *LLSC*. This model covers both the *Tags-In-SRAM* and the *Tags-In-DRAM* organizations. Next, we extend the model to achieve optimal latency by bypassing a fraction of DRAM Cache accesses.

### 3.1 DRAM Cache Model

The DRAM Cache is modeled as an instance of the *ANATOMY* model since the cache is a DRAM comprised of channels, ranks, banks and rows/columns of cells. Thus all the ingredients of the 3-stage network-of-queues formulation are applicable to the cache model. We further specialize the *ANATOMY* model by taking into account the following design considerations:

#### 3.1.1 Metadata Storage and Hit/Miss Determination:

We model the *Tag-Pred* in front of the DRAM Cache as an M/D/1 server with service rate  $\mu_{pred} = \frac{1}{t_{pred}}$ . We extend *ANATOMY* by modeling this server in front of the 3-stage DRAM queuing model as shown in Figure 4. Since  $t_{pred}$  is typically very small, the output of this server can be approximated as Markovian [17]. The total latency of this server is given by:  $L_{Pred} = t_{pred} + QD_{pred}$

With probability  $h_{pred}$  this server makes a prediction (cache hit/miss). We assume that the predictor does not make false-positive or false-negative predictions. Thus a fraction  $(1 - h_{pred})$  of the requests do not get predicted. For such requests, the DRAM Cache is first looked up, and if it is a cache miss, only then the main memory is accessed. This assumption reflects real-world implementations since the cost of a wrong prediction is expensive<sup>6</sup>.

Observe that this model includes both the *Tags-In-SRAM* as well as the *Tags-In-DRAM* organizations. For the *Tags-In-SRAM* organizations, since tags are stored on SRAM the tag look up is like a perfect predictor with  $h_{pred} = 100\%$  and  $t_{pred}$  depends on the tag store size. For the *Tags-In-DRAM* organizations, the predictor's  $h_{pred}$  and  $t_{pred}$  parameters can be set to the values proposed in literature [11, 16, 19].

As we show in Section 5.3, the performance of the *Tag-Pred* plays a critical role in overall hit latency and is an important parameter governing the decision on metadata storage location.

#### 3.1.2 Modeling the Effects of DRAM Cache Block Size:

The DRAM Cache performance is influenced by a block size factor  $B_s$  which denotes the number of *LLSC* blocks that corresponds to one DRAM Cache block. Recall that DRAM Cache blocks may be larger than CPU L1, L2 cache blocks and thus we let  $B_s$  denote the ratio of DRAM Cache block size to CPU L2 Cache block size.

<sup>6</sup>In case of accesses to dirty cache blocks, an incorrect prediction can lead to incorrect program execution.

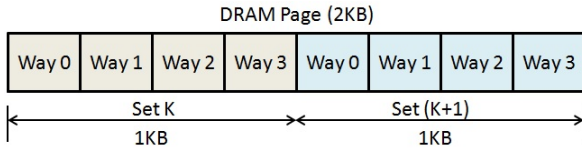


Figure 6: Set Data Layout in DRAM Pages

For instance, with a 64B line size in the L2 cache and a 512B line size in the DRAM Cache, we have  $B_s = 8$ . Note that  $B_s$  is always a power of 2.

**Block Size and Cache Hit Rate:** Larger block sizes improve cache hit rate if the workload exhibits sufficient spatial locality to utilize larger contiguous chunks of data. In a bandwidth-neutral model, *every doubling of block size halves the DRAM Cache miss rate*. If this condition holds, then block size does not influence bandwidth. In such a model knowing the DRAM Cache hit rate at the CPU cache block size organization is sufficient to estimate the hit rates at other block sizes. Specifically, if  $h_1$  is the cache hit rate when the cache is organized at the *LLSC* block size (i.e.,  $B_s = 1$ ) then the cache hit rate  $h_2$  at twice the block size (i.e.,  $B_s = 2$ ) is given by:  $h_2 = h_1 + \frac{(1-h_1)}{2}$ . Generalizing, at a block size factor  $B_s$  times the *LLSC* block size, the cache hit rate under the bandwidth-neutral model is given by:

$$h_{b_s} = 1 - \frac{(1 - h_1)}{B_s} \quad (6)$$

This model provides a useful way to understand the role of DRAM Cache block size. If miss rates do indeed halve with doubling of block size, then the cache should be organized at larger block sizes to leverage efficient line fills and lower metadata overhead. On the other hand, if miss rates did not halve with doubling of block size, then larger block sizes are wasteful of bandwidth and cache space.

Figures 5a through 5d plot the observed miss rates (bars) as well as the “ideal” (labeled “theoretical” in the figure) miss rates under the bandwidth-neutral model (lines) for 4 quad-core workloads. It shows that workloads *Q5* and *Q10* follow the empirical miss-rate-halving rule quite closely while workloads *Q7* and *Q22* tend to deviate from this model (their miss rates fall by less than half with doubling block size).

On average, this rule resulted in average cache hit rate estimation errors of 16.2% and 14.5% in 4- and 8-core workloads respectively. Thus the rule provides a reasonable approximation for cache hit rate estimation at different block sizes. It also helps to determine whether a workload is bandwidth efficient at larger block sizes or not by comparing the workloads’s actual miss rate at a large block size with the estimated miss rate.

**Block Size and RBH:** Larger block sizes improve *RBH* in general by capturing spatio-temporal locality in the workloads’ accesses. In order to understand this interaction, we first present the data layout of cache blocks in DRAM Cache pages.

The data blocks corresponding to the ways of a set are mapped to contiguous locations in the same DRAM page. This is done in order to ensure that the cache controller can access the correct DRAM row as soon as the set index has been identified. This reduces access latency as well as keeps the set index to DRAM page mapping quite simple. Depending on the associativity of the cache, one or more sets could be mapped to the same page. For example, Figure 6 shows a 2KB DRAM page holding data for 2 sets, in a 4-way set-associative cache organized at 256B block size. The 4 ways of set  $K$  occupy the first 1KB of the DRAM page while the 4 ways of the set  $(K + 1)$  occupy the next 1KB of the page.

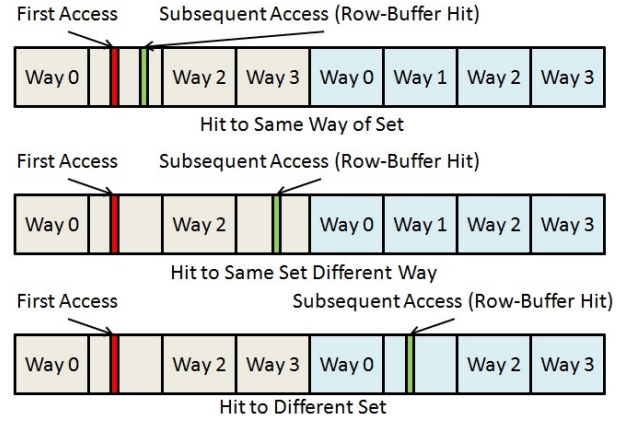


Figure 7: Row-Buffer Hit Scenarios in DRAM Caches

In this mapping, a cache hit is also a row-buffer hit (denoted  $RBH_{hit}$ ) under the following scenarios:

- R.1** A subsequent access to the DRAM bank maps to the same DRAM cache way that initially opened that page.
- R.2** A subsequent access to the DRAM bank maps to the same cache set (but a different cache way) that initially opened that page.
- R.3** A subsequent access to the DRAM bank maps to a different cache set that is mapped to the same page.

These scenarios are illustrated in Figure 7. Condition [R.1] occurs whenever the DRAM cache block size is larger than the *LLSC* block size and the CPU accesses possess spatial locality. For example, with an *LLSC* block size of 64B, a single DRAM cache block of 512B may get multiple accesses successively.

Since the physical addresses of the different ways in a set are not contiguous (contiguous addresses at cache block size granularity map to contiguous sets), the probability of getting a row-buffer hit due to condition [R.2] is quite low. This follows because each DRAM bank holds data for several millions of cache blocks and the probability of getting a subsequent access to a different block of the *same* is set is very small.

Condition [R.3] can again arise from spatial locality since consecutive cache block addresses map to consecutive sets and thus a neighboring cache block may be resident in the same page at the time the page was opened.

Thus the row-buffer hits in a DRAM cache page are limited by the spatial locality that a page can capture. Hence we estimate  $RBH_{hit}$  using a smaller “effective” page size  $E_P$  that is computed as  $E_P = B \times S$  where  $B$  is the cache block size and  $S$  is the number of sets per DRAM Cache page. In the above example,  $E_P = 256 \times 2 = 512B$ . This effective page size is used to estimate  $RBH_{hit}$  using the reuse distance methodology from *ANATOMY*. Figure 8 shows the actual and estimated  $RBH_{hit}$  in several quad-core workloads for a 128MB 2-way associative 256B block-size cache with an underlying DRAM page size of 2KB. The average error in  $RBH_{hit}$  estimation is  $< 3\%$  across 4- and 8-core workloads indicating that the modeling of an “effective page size” is a reliable method for DRAM Cache  $RBH_{hit}$  estimation.

### 3.1.3 Row Buffer Hits during Cache Line Fills:

In Section 3.1.2 we provided an estimate of the  $RBH_{hit}$  observed in the DRAM Cache due to cache hits. Here, we account

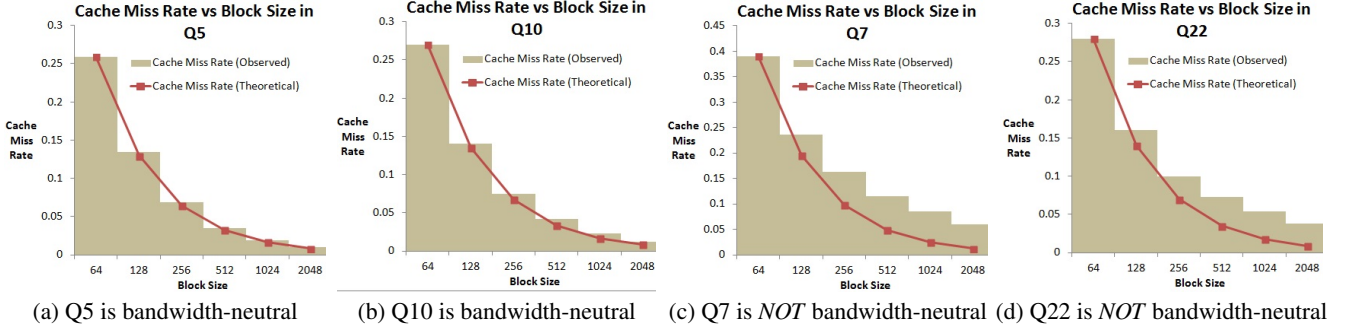


Figure 5: Miss Rate versus Block Size in a 128MB DRAM Cache

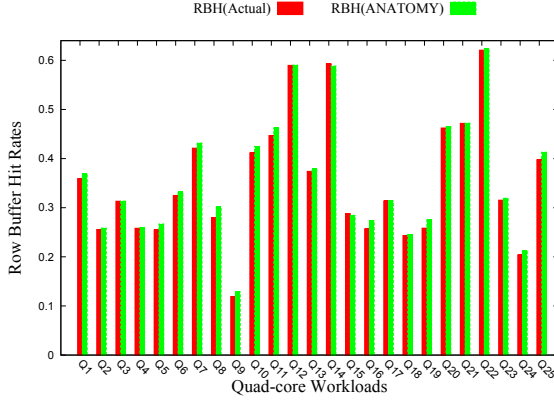


Figure 8: Row-Buffer Estimation Accuracy

for the row buffer hits seen during line fills. Note that we modeled a line fill of a DRAM cache block in terms of  $B_s$  *LLSC* block sized accesses. These accesses have high spatial locality and thus have a very high row-buffer hit rate since they are typically serviced back to back. At each cache miss, we assume that the first access is a row-buffer miss (mildly pessimistic) followed by  $(B_s - 1)$  row-buffer hits. This results in an  $RBH_{miss}$  of  $\frac{(B_s-1)}{B_s}$  for cache misses.

Combining with the  $RBH$  estimate obtained in Section 3.1.2 for cache hits, we can obtain the overall  $RBH$  as:

$$RBH_{cache} = RBH_{hit} * h_{cache} + RBH_{miss} * (1 - h_{cache}) \quad (7)$$

### 3.1.4 DRAM Cache Access Rate:

The access rate seen at the DRAM Cache is a sum of several streams of accesses (see Figure 4). In the below discussion, we assume that the arrival rate from the *LLSC* is  $\lambda$  (comprising both misses and writebacks).

**Predicted DRAM Cache Hits:** With a predictor hit rate of  $h_{pred}$  and a DRAM cache hit rate of  $h_{cache}$ , the cache sees an average arrival rate  $\lambda * h_{cache} * h_{pred}$  of incoming requests.

**No Predictions (Tag Look-Up)** A fraction  $(1 - h_{pred})$  of requests are not predicted and go to the DRAM Cache. This contributes an arrival rate of  $\lambda * (1 - h_{pred})$  to the cache. These requests cause tag accesses on the DRAM cache for hit/miss evaluation and thus need to be counted in the overall traffic to the cache.

**DRAM Cache Line Fills:** At a rate  $\lambda * (1 - h_{cache})$ , cache misses cause line fills into the DRAM Cache. Each line fill brings  $B_s$  times the *LLSC* cache block size worth of data. We model this by scaling the cache access rate by  $B_s$ . This leads to an additional arrival rate  $\lambda * (1 - h_{cache}) * B_s$  to the cache<sup>7</sup>.

**DRAM Cache Writebacks:** A fraction  $w$  of the DRAM Cache misses result in dirty writebacks from the DRAM Cache. These writebacks also contend for the same DRAM Cache resources - namely the command bus, banks and data bus. Thus this adds another arrival rate term  $\lambda * (1 - h_{cache}) * w$ . Note that  $w$  is measured in *LLSC* cache block granularity to model real world implementations wherein only the dirty sub-blocks of large blocks are written back to main memory.

These interactions effectively cause the total arrival rate at the DRAM Cache to be:

$$\lambda_{cache} = \lambda * [h_{cache} * h_{pred} + (1 - h_{pred}) + (1 - h_{cache}) * B_s + (1 - h_{cache}) * w] \quad (8)$$

It is worth highlighting that it is important to model all the above components of the arrivals seen at the DRAM Cache for a correct modeling of its latency. In particular, cache line fills constitute a significant additional traffic to the cache (particularly when the cache miss rate is high and  $B_s$  is large). We discuss this further in the Results section (Section 5).

### 3.1.5 Summing Up the Model

In summary, the model simply adds a predictor server at the head of the *ANATOMY* model to account for the tag hit/miss prediction time. It also adjusts the arrival rate seen at the DRAM Cache to take into account the additional traffic caused by tag lookups, line fills and writebacks as shown in Equation 8. The model requires  $t_{pred}$ ,  $h_{pred}$ ,  $h_{cache}$ ,  $B_s$  and  $w$  as inputs. Table 2 summarizes how these parameters are obtained.

Among the workload characteristics that *ANATOMY* requires,  $\lambda_{cache}$  and  $RBH_{cache}$  are computed as presented earlier. The estimation of the other inputs (BLP  $B$ , and Request Spread  $S$ ) is not affected by use of the DRAM as a cache.

The model estimates the average cache access latency using  $\lambda_{cache}$  (from Equation 8) and  $RBH_{cache}$  (from Equation 7) along with the rest of the *ANATOMY* workload inputs  $B$ ,  $S$ .

<sup>7</sup>Alternately, we could model this fill-stream as a different class of requests requiring a larger service time.

$$\begin{aligned}
L_{Dram\_Cache} = & \frac{1}{\mu_{Cmd\_Bus}} + QD_{Cmd\_Bus} \\
+ & \frac{1}{\mu_{bank}} + QD_{Bank} \\
+ & \frac{1}{\mu_{data}} + QD_{Data} \quad (9)
\end{aligned}$$

### 3.1.6 Extension to Main Memory

A similar analysis holds for the main memory model. The main memory sees traffic contributed by:

**Cache Misses:** A fraction  $\lambda * (1 - h_{cache})$  of incoming requests cause cache line fills. These are read requests on the main memory. Each line fill brings  $B_s$  times the *LLSC* block size worth of data. We model this by scaling the memory access rate by  $B_s$ . This leads to an additional arrival rate  $\lambda * (1 - h_{cache}) * B_s$  to the memory.

**Cache Writebacks:** A fraction  $w$  of the DRAM Cache misses result in dirty writebacks from the DRAM Cache. These writebacks create an additional arrival rate term  $\lambda * (1 - h_{cache}) * w$  to the memory.

It may be observed that the main memory arrival rate is not affected by the tag/predictor hit rate since the predictor's role is only to avoid DRAM Cache lookups for tags and not to eliminate any accesses to the main memory. The total arrival rate at memory is given by:

$$\begin{aligned}
\lambda_{mem} = & \lambda * (1 - h_{cache}) * B_s \\
+ & \lambda * (1 - h_{cache}) * w \quad (10)
\end{aligned}$$

We can derive the latency estimate  $L_{Mem}$  of main memory similar to Equation 9. *RBH*, *BLP*, and Request Spread  $S$  of the memory are estimated from a trace of accesses issued to it by the *Tag-Pred* and the DRAM-Cache. In Section 5.2 we validate the accuracy of these estimates.

### 3.1.7 Average *LLSC* Miss Penalty

The average latency seen by *LLSC* misses can now be estimated as a sum of the following weighted contributions made by  $L_{Dram\_Cache}$  and  $L_{Mem}$ :

- **Predicted Hits:** A fraction  $(h_{pred} * h_{cache})$  of all the arrivals from the *LLSC* are sent to the DRAM Cache for cache hit processing. The weighted average latency of such requests is:

$$L_{Dram\_Cache}^{Pred\_Hits} = (h_{pred} * h_{cache}) * L_{Dram\_Cache} \quad (11)$$

- **Predicted Misses:** A fraction  $(h_{pred} * (1 - h_{cache}))$  of all the arrivals from the *LLSC* are sent to memory for fetching full DRAM Cache blocks for filling them into the DRAM Cache as line fills. The weighted average latency of such requests is:

$$L_{Mem}^{Pred\_Misses} = (h_{pred} * (1 - h_{cache})) * L_{Mem} \quad (12)$$

- **Un-Predicted Hits:** A fraction  $((1 - h_{pred}) * h_{cache})$  of all the arrivals from the *LLSC* are unpredicted and turn out to be hits in the DRAM Cache. The weighted average latency of such requests is:

$$L_{Dram\_Cache}^{Unpred\_Hits} = ((1 - h_{pred}) * h_{cache}) * L_{Dram\_Cache} \quad (13)$$

- **Un-Predicted Misses:** A fraction  $((1 - h_{pred}) * (1 - h_{cache}))$  of all the arrivals from the *LLSC* are unpredicted and turn out to be misses in the DRAM Cache. This requires two rounds of accesses. The first access to the DRAM Cache to resolve the request as a miss, followed by access to the main memory. The weighted average latency of such requests is:

$$\begin{aligned}
L_{Dram\_Cache\_Then\_Mem}^{Unpred\_Misses} = & ((1 - h_{pred}) * (1 - h_{cache})) \\
& * [L_{Dram\_Cache} + L_{Mem}] \quad (14)
\end{aligned}$$

The overall average latency  $L_{LLSC}$  is given by:

$$\begin{aligned}
L_{LLSC} = & L_{Dram\_Cache}^{Pred\_Hits} + L_{Mem}^{Pred\_Misses} \\
& + L_{Dram\_Cache}^{Unpred\_Hits} + L_{Dram\_Cache\_Then\_Mem}^{Unpred\_Misses} \\
& + L_{pred} \quad (15)
\end{aligned}$$

Note that the last term accounts for the *Tag-Pred* lookup latency.

## 3.2 Load-Balancing the Cache with Main Memory

The above model reveals the opportunity for an architecture optimization - balancing the load at the DRAM Cache and main memory. At high DRAM Cache hit rates, the cache gets almost all the traffic from the *LLSC*. This causes significant contention at the DRAM Cache incurring queuing delays and increased waiting times. In such a scenario, a better use of the idle main memory could be made by diverting some of the cache traffic to it<sup>8</sup>. Thus we present a *load balancing* model that minimizes the average latency by identifying the optimal fraction of requests to divert to main memory.

For purposes of the model, we assume that the *Tag-Pred* is enhanced to divert a fraction  $f_{mem}$  of the predicted hits and misses to the main memory. In case of cache misses, the diverted accesses are like *cache bypasses* - they directly return the fetched data to the *LLSC* without performing cache line fills into the DRAM Cache. We further assume that all the diverted requests are only for *clean* cache blocks.

We can now estimate the new arrival rates and average latencies of the cache and memory as with the standalone model.

### 3.2.1 Traffic to the DRAM Cache

The arrival rate at the cache is now a sum of predicted hits sent to the cache (fraction:  $h_{pred} * h_{cache} * (1 - f_{mem})$ ), un-predicted accesses (fraction:  $(1 - h_{pred})$ ), cache line fills (fraction:  $h_{pred} * (1 - h_{cache}) * (1 - f_{mem}) * B_s + (1 - h_{pred}) * (1 - h_{cache}) * B_s$ ), and write-backs (fraction:  $(h_{pred} * (1 - h_{cache}) * (1 - f_{mem}) + (1 - h_{pred}) * (1 - h_{cache})) * w$ ). The resulting arrival rate  $\lambda_{cache}$  is:

$$\begin{aligned}
\lambda_{cache} = & \lambda * [h_{pred} * h_{cache} * (1 - f_{mem}) \\
& + (1 - h_{pred}) \\
& + h_{pred} * (1 - h_{cache}) * (1 - f_{mem}) * B_s \\
& + (1 - h_{pred}) * (1 - h_{cache}) * B_s \\
& + h_{pred} * (1 - h_{cache}) * (1 - f_{mem}) * w \\
& + (1 - h_{pred}) * (1 - h_{cache}) * w] \quad (16)
\end{aligned}$$

The average DRAM Cache access latency  $L_{Dram\_Cache}$  can be obtained from Equation 9.

<sup>8</sup>This is valid only if the cache did not hold a more recent copy of the data.

### 3.2.2 Traffic to the Main Memory

The arrival rate at the main memory is now a sum of predicted hits sent to the memory (fraction:  $h_{pred} * h_{cache} * f_{mem}$ ), memory reads for cache line fills (fraction:  $h_{pred} * (1 - h_{cache}) * (1 - f_{mem}) * B_s + (1 - h_{pred}) * (1 - h_{cache}) * B_s$ ), misses that bypass the cache (fraction:  $h_{pred} * (1 - h_{cache}) * f_{mem}$ ), and write-backs (fraction:  $(h_{pred} * (1 - h_{cache}) * (1 - f_{mem}) + (1 - h_{pred}) * (1 - h_{cache})) * w$ ). The resulting arrival rate  $\lambda_{mem}$  is estimated similar to Equation 10.

The average main memory access latency  $L_{Mem}$  can be obtained similar to Equation 9.

### 3.2.3 Optimal $f_{mem}$

In order to estimate the average latency, we first estimate the latency of each type of request processing:

- **Predicted Hits sent to the DRAM Cache:** A fraction ( $h_{pred} * h_{cache} * (1 - f_{mem})$ ) of all the arrivals from the *LLSC* are sent to the DRAM Cache for cache hit processing. The weighted average latency of such requests is:

$$L_{DRAM\_Cache}^{Pred\_Hits} = (h_{pred} * h_{cache} * (1 - f_{mem})) * L_{DRAM\_Cache} \quad (17)$$

- **Predicted Hits diverted to the main memory:** A fraction ( $h_{pred} * h_{cache} * f_{mem}$ ) of all the arrivals from the *LLSC* are diverted to memory for hit processing. The weighted average latency of such requests is:

$$L_{Mem}^{Pred\_Hits} = (h_{pred} * h_{cache} * f_{mem}) * L_{Mem} \quad (18)$$

- **Predicted Misses that bypass the cache:** A fraction ( $h_{pred} * (1 - h_{cache}) * f_{mem}$ ) of all the arrivals from the *LLSC* are sent to memory for fetching the *LLSC* requested block. These requests do not initiate DRAM Cache fills. The weighted average latency of such requests is:

$$L_{Mem}^{Pred\_Misses\_Bypassed} = (h_{pred} * (1 - h_{cache}) * f_{mem}) * L_{Mem} \quad (19)$$

- **Predicted Misses that are cached:** A fraction ( $h_{pred} * (1 - h_{cache}) * (1 - f_{mem})$ ) of all the arrivals from the *LLSC* are sent to memory for fetching full DRAM Cache blocks for filling them into the DRAM Cache as line fills. The weighted average latency of such requests is:

$$L_{Mem}^{Pred\_Misses\_Cached} = (h_{pred} * (1 - h_{cache}) * (1 - f_{mem})) * L_{Mem} \quad (20)$$

- **Un-Predicted Hits:** This is the same as in Equation 13 above.
- **Un-Predicted Misses:** This is the same as in Equation 14.

The overall average miss penalty  $L_{LLSC}(f_{mem})$  seen by the *LLSC* is a sum of the above weighted latencies similar to Equation 15. Treating this as a function of the single variable  $f_{mem}$ , we can numerically minimize  $L_{LLSC}(f_{mem})$  at some  $f_{mem}^*$ . The value  $f_{mem}^*$  defines the optimal distribution of cache requests to the main memory in order to minimize overall average latency. It should be noted that this optimal  $f_{mem}^*$  depends on the configurations of the DRAM Cache and main memory. Changes to the cache organization and/or the main memory will require re-computing  $f_{mem}^*$ .

To emphasize, this set up has taken into account both the cache and memory performance models into a single hybrid model that allows estimation of a global minimum latency. In Section 5.4 we explore this load-balancing opportunity in real systems and demonstrate that this can lead to as much as 74% reduction in average latency.

Processor	3.2 GHz OOO Alpha ISA
L1I Cache	32kB private, 64B blocks, Direct-mapped, 2 cycle hit latency
L1D Cache	32kB private, 64B blocks, 2-way set-associative, 2 cycle hit latency
L2 Cache (LLSC)	For 4/8 cores: 4MB/8MB, 8-way/16-way, 128/256 MSHRs, 64-byte blocks, 7/9 cycles hit latency
Tags-In-DRAM (AlloyCache)	For 4/8 cores: 128MB/256MB, <b>Direct-Mapped</b> , <b>64-byte blocks</b> , 80-byte tag+data burst, Cache Memory in 2/4 Channels, Total of 16/32 DRAM banks, 2KB page, 128-bit bus width, 1.6GHz, CL-nRCD-nRP=9-9-9
Tag-Pred	A 2-way Set Associative Tag Cache For 4/8 cores: 48KB/96KB, (Design similar to that in [11])
Tags-In-SRAM	For 4/8 cores: 128MB/256MB, <b>2-Way Set Associative</b> , <b>1024-byte blocks</b> , Cache Memory in 2/4 Channels, Total of 16/32 DRAM banks, 2KB page, 128-bit bus width, 1.6GHz, CL-nRCD-nRP=9-9-9
Memory Controller	For 4/8 cores: 1/2 off-chip data channels Each MC: 64-bit interface to channel, 256-entry command queue FR_FCFs scheduling [20], open-page policy Address-interleaving: row-rank-bank-mc-column
Off-Chip DRAM	For 4/8 cores: 4GB/8GB main memory using: DDR3-1600H, BL (cycles)=4, CL-nRCD-nRP=9-9-9 in 2/4 ranks, 16/32 banks Refresh related: $T_{REFI}$ of 7.8us and $T_{RFC}$ of 280nCK

Table 1: CMP configuration

## 4. EXPERIMENTAL SETUP

We evaluate the accuracy of the proposed model by comparing the model-predicted latency with results from detailed simulations. Using multiprogrammed workloads running on the GEM5 [3] simulation infrastructure, we obtained the observed latency for the configurations listed in Table 1. For quad-core workloads, timing simulations were run for 1 billion instructions on each core after fast-forwarding the first 10 billion instructions to allow for sufficient warm-up. As is the norm, when a core finishes its timing simulation, it continues to execute until all the rest of the cores have completed<sup>9</sup>. In case of 8 and 16-core workloads, due to the amount of simulation time required, we collected statistics on timing runs of 500M and 250M instructions per core respectively. In all cases, the total instructions simulated across all the cores amount to more than 4B.

We obtained the DRAM Cache model inputs ( $h_{pred}, h_{cache}, w$ ) using a trace-based DRAM cache simulator. This (un-timed) simulator simulates the various *Tags-In-SRAM* and *Tags-In-DRAM* organizations. Traces collected from GEM5 simulations of 4, and 8-core architectures running for 75 billion instructions on each core were supplied as input to this cache simulator. This has resulted in 120M – 450M accesses to the DRAM cache, with an average of 310M DRAM cache accesses per workload.  $t_{pred}$  is obtained from the CACTII tool [23] with 22nm technology. All the other workload parameters are estimated as in *ANATOMY*. For completeness, Table 2 lists the relevant parameters used in the model, their sources and when they are obtained/updated.

Our workloads are comprised of programs from SPEC 2000 and SPEC 2006 benchmark [9] suites. The 4, and 8-core multiprogrammed workloads are listed in Table 3. These benchmarks were carefully combined to create high, moderate and low levels of memory *intensity*<sup>10</sup> in the chosen workloads to ensure a representative mix. Workloads marked with a “\*” in Table 3 have high memory intensity (*LLSC* miss rate  $\geq 10\%$ ). We also measured the footprints of these workloads in terms of the number of distinct *64B*

<sup>9</sup>The statistics are collected only during the first 1 Billion instructions.

<sup>10</sup>Intensity was measured in terms of the last-level SRAM cache miss rate.



Parameter	Source	Update Frequency
$B_s$	Input to the model	For each DRAM Cache block size explored
$h_{pred}$	(Un-timed) Cache Simulator	For each predictor organization and size
$t_{pred}$	CACTII tool [23]	For each predictor table size
$\lambda$	From <i>LLSC</i> trace, as in <i>ANATOMY</i>	Once per workload
$\lambda_{cache}$	Estimated using Equation 8	For each DRAM Cache size and $B_s$
$h_{cache}$	(Un-timed) Cache Simulator	For each DRAM Cache size and $B_s$
$w$	(Un-timed) Cache Simulator	For each DRAM Cache size and $B_s$
$RBH_{cache}$	Estimated using Equation 7	For each DRAM Cache size, $B_s$ and cache page size
$BLP_{cache}, S_{cache}$	From <i>LLSC</i> trace, as in <i>ANATOMY</i>	For each DRAM Cache size, $B_s$ and number of DRAM Cache banks
$\lambda_{mem}$	Estimated using Equation 10	For each DRAM Cache size and $B_s$
$RBH_{mem}$	Trace of misses from DRAM cache, as in <i>ANATOMY</i>	For each main memory configuration (Banks and Page Size)
$BLP_{mem}, S_{mem}$	Trace of misses from DRAM cache, as in <i>ANATOMY</i>	For each main memory configuration (Banks and Page Size)
DRAM Cache and Memory Timing Values	Input to the model. Obtained from JEDEC specification	For each memory technology/device type

Table 2: Sources of model parameters

Quad-Core Workloads
*Q1:(462,459,470,433), *Q2:(429,183,462,459), *Q3:(181,435,197,473), *Q4:(429,462,471,464), *Q5:(470,437,187,300), *Q6:(462,470,473,300), *Q7:(459,464,183,433), *Q8:(410,464,445,433), *Q9:(462,459,445,410), *Q10:(429,456,450,459), *Q11:(181,186,300,177), *Q12:(168,401,435,464), *Q13:(434,435,437,171), *Q14:(444,445,459,462), *Q15:(401,410,178,177), *Q16:(300,254,255,470), *Q17:(171,181,464,465), *Q18:(464,450,465,473), *Q19:(453,433,458,410), *Q20:(462,471,254,186), *Q21:(462,191,433,437), *Q22:(197,168,179,187), *Q23:(401,473,435,177), *Q24:(416,429,454,175) *Q25:(254,172,178,188)
Eight Core Workloads
E1:(462,459,433,456,464,473,450,445), *E2:(300,456,470,179,464,473,450,445), *E3:(168,183,437,401,450,435,445,458), *E4:(187,172,173,410,470,433,444,177), E5:(434,435,450,453,462,471,164,186), E6:(416,473,401,172,177,178,179,435), *E7:(437,459,445,454,456,465,171,197), E8:(183,179,433,454,464,435,444,458), *E9:(183,462,450,471,473,433,254,168), *E10:(300,173,178,187,188,191,410,171), *E11:(470,177,168,434,410,172,464,171), E12:(459,473,444,453,450,197,175,164), E13:(471,462,186,254,465,445,410,179), *E14:(187,470,401,416,433,437,456,454), *E15:(300,458,462,470,433,172,191,471), E16:(183,473,401,435,188,434,164,427)

Table 3: Workloads

blocks accessed. The average memory footprints in 4-core and 8-core workloads are 990MB and 2.1GB respectively. We also found that on average 87% of all the DRAM cache misses are due to capacity/conflict. Thus our workloads are sufficiently exercising the DRAM cache.

The details of our architecture configurations are summarized in Table 1. We simulate two typical configurations: a *Tags-In-DRAM* configuration based on the *AlloyCache* [19] organization, and a *Tags-In-SRAM* configuration based on the *FootprintCache* [12] organization.

## 5. RESULTS

We first validate the accuracy of estimating key model input parameters. Next we perform end-to-end validation of the *Tags-In-DRAM* and *Tags-In-SRAM* designs in Section 5.2 and show that the average *LLSC* miss penalty is estimated with good accuracy. Finally, we use this validated model to explore two important cache organizational topics in Sections 5.3 and 5.4.

### 5.1 Validation of the model input parameters

In this section, we validate the accuracy of the estimated model input parameters. We only focus on two parameters -  $\lambda_{cache}$  and  $RBH_{cache}$  - as the other parameters are either trivially computed from the cache access trace ( $\lambda$ ,  $h_{pred}$ ,  $h_{cache}$ , write-back rate  $w$ ) or their estimates have already been validated for accuracy in the baseline study presented in *ANATOMY* [7] (Bank-Level Parallelism *BLP*, and Request Spread  $S$ ). The validations used both the *Tags-In-DRAM* and *Tags-In-SRAM* organizations.

#### 5.1.1 Exponential Inter-Arrival Times

We validated the assumption of exponential inter-arrival times seen by the cache (with a mean of  $\frac{1}{\lambda_{cache}}$  where  $\lambda_{cache}$  is computed using Equation 8) by comparing the actual inter-arrival times observed in the detailed simulations against the theoretical distribution. Using the *Chi-square* goodness-of-fit test [18] with 30 degrees of freedom, the average  $p$ -values<sup>11</sup> for 4, and 8-core workloads are 0.03, and 0.01 respectively, denoting reasonably high confidence of match between the actual and theoretical distributions.

#### 5.1.2 Estimating $RBH_{cache}$

We compared  $RBH_{cache}$  estimated using Equation 7 with the actual  $RBH$  in simulations and obtained very low average errors of 4.3% and 3.7% in 4-core and 8-core workloads.

## 5.2 End-to-End Model Validation

Having validated the accuracy of the parameters, we demonstrate that the overall model provides accurate estimates of the *LLSC* miss penalty.

#### 5.2.1 Validation of the Tags-in-DRAM Model:

We use the *AlloyCache* [19] organization to validate the accuracy of *Tags-In-DRAM* model. The *AlloyCache* organizes the cache as direct-mapped 64B blocks with metadata and data co-located in the same DRAM pages. We implemented the *SAM* (Serial Access Model) in our simulator, wherein the cache is first probed for hit/miss detection and subsequently a miss is sent to the main memory. Thus every *LLSC* access is sent to the cache for hit/miss evaluation. Since no predictor is used, we set  $h_{pred}$  and  $t_{pred}$  both to 0. In order to incur lower DRAM latency, the *AlloyCache* uses a larger data burst of 80 bytes to read both the tag and data associated in a single access. *ANATOMY* framework incorporates this by setting the data bus server's bus length ( $BL$ ) value to 5 clock cycles (see Section 2.2.4). Further, since the DRAM cache uses 64B block size, the block size factor  $B_s = 1$ . Estimates for  $h_{cache}$ , and  $w$  were obtained by simulating traces on the DRAM Cache simulator. We estimate  $RBH_{cache}$ ,  $\lambda$ ,  $\lambda_{cache}$ , and  $\lambda_{mem}$  as discussed in Section 3.1. Additional *ANATOMY* inputs namely *BLP* and Request Spread  $S$  are estimated from the memory access trace. These estimates are used to compute the estimated latency  $L_{LLSC}$  as in Equation 15.

Figures 9a and 9b report the errors in the estimation of  $L_{LLSC}$  when compared to results from detailed simulation of the *Alloy-*

<sup>11</sup> $p$ -value is a statistical measure of deviation of the actual distribution from the hypothesis.

Cache Size (MB)	Block Size (B)	Tag Store Size (MB)	Access Latency (cycles)
64	64	4	10
64	256	1	6
64	512	0.5	4
128	64	8	12
128	256	2	8
128	512	1	6
512	64	32	16
512	256	8	12
512	512	4	10

Table 4: SRAM Tag Storage Size and Latency

Cache in quad and 8-core configurations<sup>12</sup>. The averages of (absolute) errors are 10.8% and 9.3% respectively showing that the model captures key architectural elements reasonably well.

### 5.2.2 Validation of the Tags-in-SRAM Model:

We use a configuration similar to the proposal in [12] wherein 1024B blocks are employed with the metadata stored on SRAM. For simplicity, we omitted the feature of bypassing *singleton* blocks<sup>13</sup> in both the simulation and in the model. We set  $t_{pred}$  to the latency estimate provided by CACTII [23] (see Section 5.3) and  $h_{pred}$  to 1.0. We also set  $B_s = 16$  since each DRAM Cache block is  $16 \times$  the size of the *LLSC* block. The rest of the model parameters are estimated as indicated above in Section 5.2.1.

Figures 10a and 10b report the errors in latency estimation of the *Tags-In-SRAM* model when compared to results from detailed simulation in quad and 8-core configurations. The averages of (absolute) errors are 10.5% and 8.2% respectively.

Averaged over both the *Tags-In-DRAM* and *Tags-In-SRAM* configurations, the model has average errors of 10.7% and 8.8% in quad and 8-core configurations respectively. These results indicate that the proposed model accurately captures the salient properties of a wide range of DRAM Cache organizations. Thus the model can be used as an analytical tool for rapid exploration of different cache organizations. In Sections 5.3 and 5.4 we use the model to explore two important design considerations.

## 5.3 Insight 1 - Can Tags-In-DRAM designs outperform Tags-In-SRAM?

We quantitatively argue that *Tags-In-SRAM* designs generally outperform *Tags-In-DRAM* designs except when the *Tag-Pred* can achieve very high hit rates. Table 4 lists the tag storage size and associated access latency for various DRAM Cache sizes and block sizes (assuming a 4B tag overhead per block) when tags are held on SRAM. The latency estimates are obtained from CACTII [23] at 22nm using high performance cells. Cycles are in units of a 3.2GHz clock. This corresponds to the tag access latency ( $t_{pred}$ ) in *Tags-In-SRAM* designs.

In case of *Tags-In-DRAM*, the average tag access time may be expressed as:

$$t_{tag}^{tags-in-dram} = h_{pred} * t_{pred} + (1 - h_{pred}) * t_{dram\_cache} \quad (21)$$

Note that this equation does not take any contention at the *Tag-Pred* or at the DRAM cache into account and thus provides an aggressive estimate for the tag access time. Figure 11 plots  $t_{tag}^{tags-in-dram}$  as a function of  $h_{pred}$  at 3 different DRAM access latencies ( $t_{dram\_cache}$  set to 10ns, 15ns, 20ns) and  $t_{pred} = 2$  (i.e., prediction in 2 cycles).

<sup>12</sup>A negative error indicates that the model estimated a higher latency than observed from simulation.

<sup>13</sup>Singleton blocks are defined to be those 1024B blocks that receive only one access to a 64B sub-block when they are cache-resident.

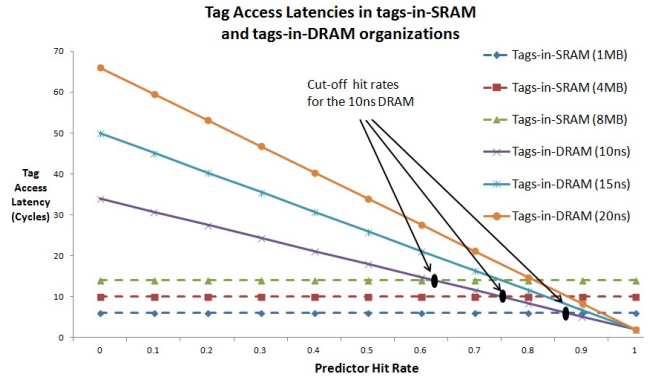


Figure 11: Comparing tag latencies of SRAM and DRAM tags

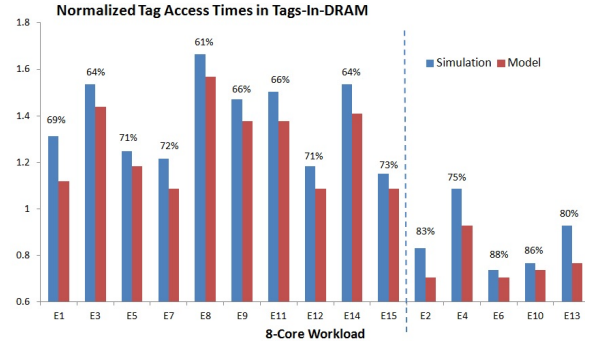


Figure 12: Tag Access Times in Eight-Core Workloads

For comparison the fixed tag access latency of the *Tags-In-SRAM* designs are also plotted for 3 different tag store sizes (1MB, 4MB, 8MB) (using latency estimates with Table 4). The intersections of the *Tags-In-DRAM* lines (solid) with the *Tags-In-SRAM* lines (dotted) denote the *cut-off* predictor hit rates - lower hit rates suffer higher latencies in the *Tags-In-DRAM* organization. This reveals the importance of achieving high prediction rates - for example, even with a fast DRAM Cache access time of 10ns, the predictor needs to achieve a hit rate in excess of 75% to outperform a 4MB *Tags-In-SRAM* organization.

We verified this claim by measuring the tag access latencies in detailed simulations of the *Tags-In-DRAM* configuration with predictor (refer Table 1) and comparing them to estimated tag access latencies (from Equation 21). Figure 12 plots these latencies normalized to the tag access time of the *Tags-In-SRAM* organization. The values on top of the bars are the  $h_{pred}$  values. 10 out of 15 workloads (grouped to the left of the vertical dashed line) have higher tag access latency in the *Tags-In-DRAM* design than in the *Tags-In-SRAM* design. The model predicts this correctly except for workload E4. In workloads E2, E6, E10 and E13, the  $h_{pred}$  values are low (all below 75%) while in the other workloads, the  $h_{pred}$  values are high, confirming the analysis presented above.

Thus we conclude that the use of this auxiliary tag hit/miss prediction structure in SRAM is beneficial only when the predictor achieves a high enough prediction rate to outperform the SRAM tag look up time.

## 5.4 Insight 2 - Bypassing the Cache Results in Overall Latency Reduction

In this section, we show that contrary to the expectation that higher cache hit rates are always better, it is often helpful to di-

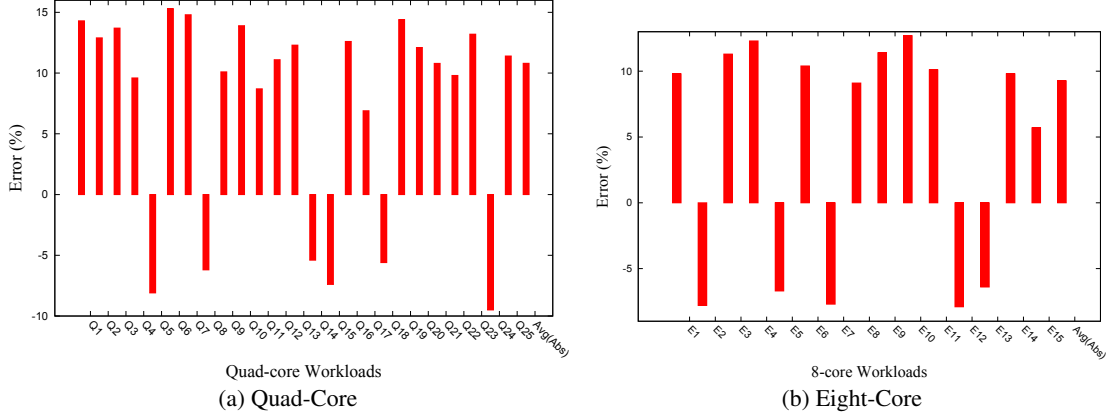


Figure 9: *Tags-In-DRAM* Model Validation: Errors in  $LLSC$  Miss Penalty ( $L_{LLSC}$ ) Estimation

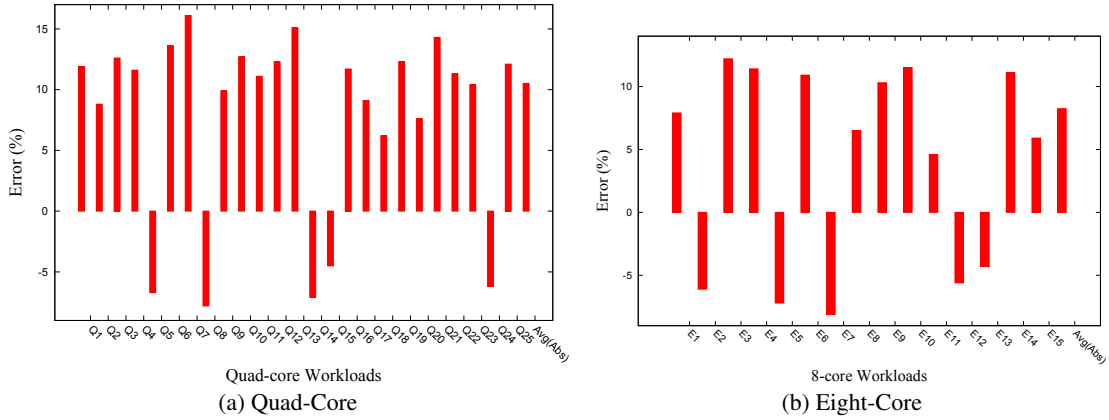


Figure 10: *Tags-In-SRAM* Model Validation: Errors in  $LLSC$  Miss Penalty ( $L_{LLSC}$ ) Estimation

vert a fraction of cache accesses to the main memory. Even though the latency of an individual access to the DRAM Cache is lower than that of main memory, a high arrival rate to the cache builds up congestion at the cache banks causing longer waiting times. Under such conditions, we show that our experimental evaluation corroborates the model’s prediction of the existence of a fraction  $f_{mem}^*$ ,  $0 < f_{mem}^* < 1$ , that results in lower  $L_{LLSC}$  than with  $f_{mem} = 0$  (sending all the requests to the cache).

We explore the results on two *Tags-In-SRAM* 128MB direct-mapped cache organizations, one with a block size of 64B and the other with a block size of 512B. Figures 13a and 13b show the observed and model-predicted  $LLSC$  latencies in quad-core workload Q1 as  $f_{mem}$  is varied from 0 to 1 in steps of 0.1 for two cache block sizes. It is evident that there is an optimal distribution of requests across the cache and main memory that results in the least  $LLSC$  miss penalty. From Figure 13a, we observe that the queuing contention has a lesser impact at the 64B block size (since  $B_s = 1$  and the cost of a line fill is not high) and the cache bypass reduces the latency by a maximum of 16% at  $f_{mem} = 0.2$ .

In Figure 13b, we observe that at both the extremes ( $f_{mem} = 0$  and  $f_{mem} = 1$ ) the average latency steeply increases. In fact, the latency experienced with no bypass ( $f_{mem} = 0$ ) is worse than when the DRAM Cache is not even used ( $f_{mem} = 1$ ). This is due to the high cost of cache line fills (filling 512B). Recall that when the cache is bypassed, only the requested 64B is fetched and re-

turned to the  $LLSC$ . Employing an optimal distribution ( $f_{mem} = 0.3$ ) reduces latency by as much as 74% compared to the baseline ( $f_{mem} = 0$ ). In all cases, very high bypass rates should be avoided since they lead to increased congestion at the slower main memory. Similar results are observed in the other quad and eight-core workloads.

These results reveal the opportunity for architecture enhancement wherein the *Tag-Pred* structure could observe the queuing delays and latencies at the cache and main memory and adaptively estimate the optimal  $f_{mem}^*$  to improve system performance.

## 6. RELATED WORK AND CONCLUSIONS

While both cache [1, 4, 5, 8, 21] and DRAM [2, 6, 15, 22, 26, 27] models exist, no prior analytical model has taken into account the interactions that result from using DRAM as a substrate for cache functionality. In particular, our work encompasses the entire DRAM Cache design space spanning *Tags-In-SRAM* and *Tags-In-DRAM* organizations taking into account the key underlying architectural issues of how metadata is accessed, the role of cache block size, and *RBH* and the significance of the *Tag-Pred*.

By comparing the model with detailed multi-core simulations, we showed that the model predicts latency accurately, achieving average errors of 10.7% and 8.8% in 4 and 8-core workloads respectively. The model offers two insights: one, the tag cache/predictor has to achieve very high hit rate in *Tags-In-DRAM* designs to out-

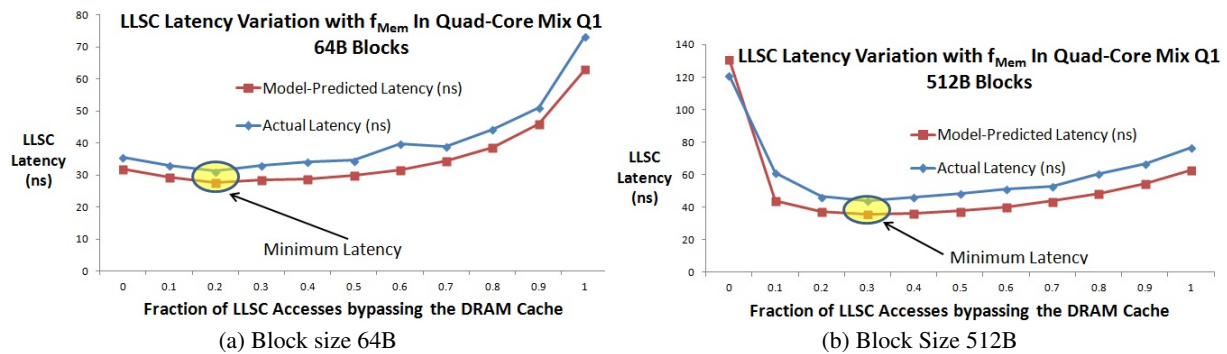


Figure 13: Latency variation with  $f_{mem}$  in Quad-core Workload Q1 at two block sizes

perform *Tags-In-SRAM* designs, and two, bypassing a fraction of DRAM cache accesses results in overall LLSC miss penalty reduction. The model thus serves as a practical analytical tool for rapid design space exploration of DRAM Cache designs.

## 7. REFERENCES

- [1] A. Agarwal, J. Hennessy, and M. Horowitz, "An analytical cache model," *ACM Trans. Comput. Syst.*, 1989.
- [2] J. H. Ahn, M. Erez, and W. J. Dally, "The design space of data-parallel memory systems," in *SC*, 2006.
- [3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, 2011.
- [4] X. E. Chen and T. M. Aamodt, "Hybrid analytical modeling of pending cache hits, data prefetching, and mshrs." in *MICRO*, 2008.
- [5] —, "Modeling cache contention and throughput of multiprogrammed manycore processors." *IEEE Trans. Computers*, 2012.
- [6] H. Choi, J. Lee, and W. Sung, "Memory access pattern-aware DRAM performance model for multi-core systems," in *ISPASS*, 2011.
- [7] N. Guler, M. Mehendale, R. Manikantan, and R. Govindarajan, "Anatomy: An analytical model of memory system performance," *SIGMETRICS*, 2014.
- [8] A. Hartstein, V. Srinivasan, T. R. Puzak, and P. G. Emma, "On the nature of cache miss behavior: Is it v2?" *J. Instruction-Level Parallelism*, 2008.
- [9] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, 2006.
- [10] M. D. Hill, "A case for direct-mapped caches," *Computer*, 1988.
- [11] C.-C. Huang and V. Nagarajan, "Atcache: Reducing dram-cache latency via a small sram tag cache," in *PACT*, 2014.
- [12] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 404–415.
- [13] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "Chop: Adaptive filter-based dram caching for cmp server platforms." in *HPCA*.
- [14] E. Kultursay, M. T. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating stt-ram as an energy-efficient main memory alternative." in *ISPASS*. IEEE, 2013.
- [15] F. Liu et al, "Understanding how off-chip memory bandwidth partitioning in chip multiprocessors affects system performance," in *HPCA-16*, 2010.
- [16] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44, 2011.
- [17] C. D. Pack, "The output of an m/d/1 queue," *Operations Research*, Vol. 23, No. 4, 1975.
- [18] R. L. Plackett, "Karl pearson and the chi-squared test," *International Statistical Review (ISI)* 51(1):59-72, 1983.
- [19] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45, 2012.
- [20] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," *SIGARCH Comput. Archit. News*, 2000.
- [21] G. E. Suh, S. Devadas, and L. Rudolph, "Analytical cache models with applications to cache partitioning," in *Proceedings of the 15th International Conference on Supercomputing*, 2001.
- [22] G. Sun et al, "Moguls: a model to explore the memory hierarchy for bandwidth improvements," in *ISCA-38*, 2011.
- [23] S. J. E. Wilton and N. P. Jouppi, "Cacti: An enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, 1996.
- [24] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, *Proceedings of the IEEE*, 2010.
- [25] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. S. Lee, "An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth." 2010.
- [26] G. L. Yuan et al, "A hybrid analytical DRAM performance model," 2009.
- [27] M. Zhou, Y. Du, B. R. Childers, R. Melhem, and D. Mosse, "Writeback-aware bandwidth partitioning for multi-core systems with pcm," in *PACT*, 2013.