An Efficient Temporal Data Prefetcher for L1 Caches

Mohammad Bakhshalipour, *Student Member, IEEE,*
Pejman Lotfi-Kamran, *Member, IEEE,* and Hamid Sarbazi-Azad

Abstract—Server workloads frequently encounter L1-D cache misses, and hence, lose significant performance potential. One way to reduce the number of L1-D misses or their effect is data prefetching. As L1-D access sequences have high temporal correlations, temporal prefetching techniques are promising for L1 caches. State-of-the-art temporal prefetching techniques are effective at reducing the number of L1-D misses, but we observe that there is a significant gap between what they offer and the opportunity. This work aims to improve the effectiveness of temporal prefetching techniques. To overcome the deficiencies of existing temporal prefetchers, we introduce Domino prefetching. Domino prefetcher is a temporal prefetching technique that looks up the history to find the last occurrence of the last one or two L1-D miss addresses for prefetching. We show that Domino prefetcher captures more than 87% of the temporal opportunity at L1-D. Through evaluation of a 16-core processor on a set of server workloads, we show that Domino prefetcher improves system performance by 26% (up to 56%).

Index Terms—Server workloads, L1-D misses, data prefetching, temporal correlation.

1 Introduction

Server workloads have vast datasets beyond what can be captured by on-chip caches of modern processors [1], [2]. Consequently, server workloads encounter frequent cache misses during their execution. The cache misses prevent server processors from reaching their peak performance, because cores are idle waiting for the data to arrive. Specifically, L1-D cache misses are responsible for a significant performance degradation in server workloads.

Data prefetching is a historical approach to eliminate cache misses or reduce their effect. While it has been shown that simple prefetching techniques, such as stride prefetching, are ineffective for server workloads [1], more advanced data prefetchers may eliminate or reduce the effect of L1-D cache misses in such workloads. One of the promising prefetching techniques, especially for L1 caches, is temporal prefetching [3], [4]. In temporal prefetching, the sequence of past cache misses/accesses is used to predict future cache misses.

While existing temporal data prefetchers are useful at eliminating cache misses, we observe that there is a significant gap between what they offer and what opportunity analysis shows for temporal prefetching. Figure 1 shows the opportunity for temporal prefetching in L1-D and what idealized forms of STMS [3] and ISB [4], two state-of-the-art temporal prefetchers, offer for several server workloads. We use the Sequitur hierarchical data compression algorithm [5] to identify the opportunity of temporal prefetching. While STMS looks for temporal correlation in the global miss sequence, ISB applies temporal correlation to the PC localized miss sequences.[1] As the figure clearly shows, there is a significant gap between the opportunity and what idealized forms of existing state-of-the-art temporal prefetchers offer across many workloads. On average, the best state-of-the-art temporal prefetcher captures less than 72% of the opportunity.

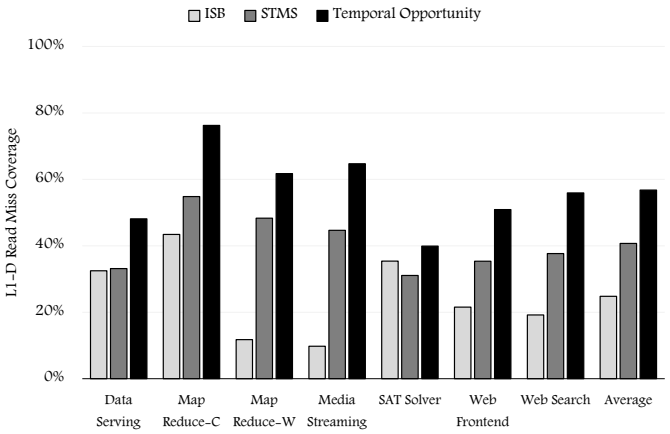

Fig. 1. Coverage of idealized forms of two state-of-the-art temporal prefetchers versus the opportunity.

This work aims to improve the effectiveness of temporal prefetching techniques. Corroborating prior work [6], our studies show that PC localization is not useful for temporal correlation in server workloads. Moreover, most of existing temporal prefetching techniques (e.g., [3]) rely on a single miss address to lookup the history to identify a temporal stream. Our analysis shows that a single miss address cannot always locate the right stream in the history. To overcome these limitations, we propose a temporal prefetcher, named *Domino*, that uses a combination of one and two previous miss addresses to lookup the global history and prefetches the next address. We show that Domino prefetcher captures most of the temporal opportunity and performs better than the state-of-the-art temporal prefetchers at L1 data caches.

2 Motivation

In this section, we reduce the problem of temporal prefetching to predicting the next miss based on the previously-observed miss

- *M. Bakhshalipour is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. E-mail: bakhshalipour@ce.sharif.edu*
- *P. Lotfi-Kamran is with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM).*
- *H. Sarbazi-Azad is with the Department of Computer Engineering, Sharif University of Technology and the School of Computer Science, Institute for Research in Fundamental Sciences (IPM).*

1. While ISB is originally proposed to be applied to the PC localized LLC access sequences, for fairness, we apply it to the PC localized L1 miss sequences.

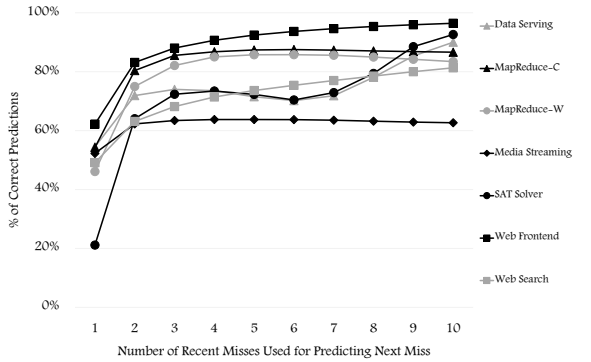

Fig. 2. The fraction of lookups that result in a correct prediction over all lookups that find a match in the history, as a function of the number of addresses that they attempt to match.

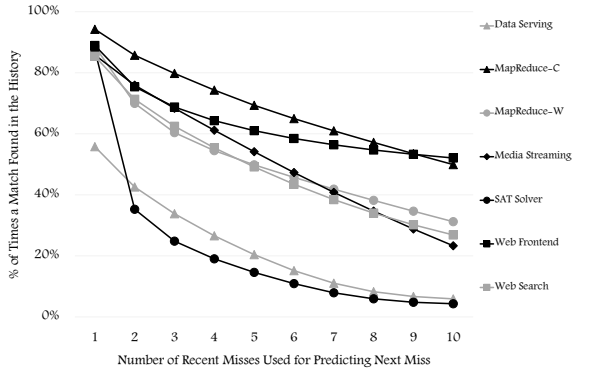

Fig. 3. The fraction of lookups that find a match in the history over all lookups, as a function of the number of addresses that they attempt to match.

sequence. Figure 2 shows the fraction of lookups that lead to a correct prediction over all lookups that find a match in the history, as a function of the number of addresses that the lookup attempts to match. As the figure shows, the fraction of lookups that lead to a correct prediction is low if lookups match only a single address. So, temporal prefetchers that rely on just one miss address to look up the history (e.g., [3]), frequently prefetch incorrectly.

Moreover, Figure 3 shows the fraction of lookups that find a match in the history as the number of addresses that they match increases. As expected, the number of lookups that find a match reduces as the number of matching addresses increases. So, temporal prefetchers that rely on a couple of misses to lookup the history (e.g., Digram [6]), miss significant opportunity due to lack of finding a match in the previously-observed misses.

Based on these observations, we propose Domino. Logically, Domino prefetcher looks up the history with the last N misses. If a match is found, Domino issues a prefetch, and otherwise, it looks up the history with one fewer miss, in a recursive manner. We found that, except for one workload, increasing N beyond two yields little improvement in coverage and accuracy of the prefetcher, and as such, we choose $N = 2$.

3 The Proposal

Domino prefetcher relies on two per-core tables to record past L1-D misses and replay the sequence of future L1-D misses. We refer to these two tables as Miss History Tables (MHTs).

As a result of high coverage and low overprediction rate, Domino prefetcher directly prefetches into L1-D caches. It

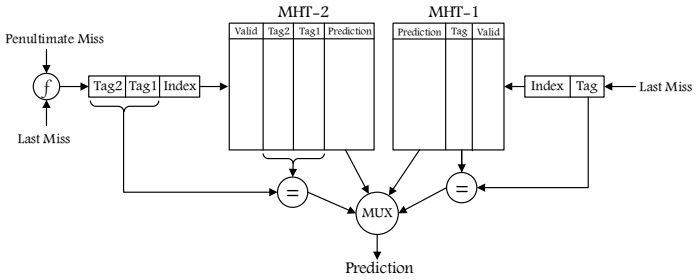

Fig. 4. Logical view of the miss history tables (MHTs).

means that Domino prefetcher does not need a special storage for keeping the prefetched cache blocks. It only requires a storage for the meta-data (i.e., the history) that guides it what to prefetch. Also, Domino prefetcher takes action (i.e., looks up the history and attempts to prefetch cache blocks) both on L1-D cache misses and on accesses to already prefetched cache blocks. As a result, Domino prefetcher requires one bit per L1-D cache block that indicates whether the cache block is prefetched or not.

Using the triggering events (i.e., cache misses and accesses to the prefetched blocks), Domino predictor constructs two miss history tables (MHTs). An entry in an MHT associates past triggering events to a future L1-D miss. One table associates a single triggering address to a future miss address and the other one associates two consecutive triggering addresses to a future miss. We refer to the former as MHT-1 and the latter as MHT-2. Figure 4 shows the logical view of the history tables.

An entry in an MHT has tags (one tag in MHT-1 and two tags in MHT-2), a prediction field, and a valid bit. The tag stores part of the identifier that is used for the lookup but has not been used as the index; the prediction is the future miss address, and the valid bit determines if this entry has valid data. Note that for MHT-2, we hash the two triggering addresses (we use a simple XOR for the hash function) to form a single identifier and use the identifier to look up MHT-2, but store the tags of both addresses in the table.

Recording. Upon a triggering event, we use the previous triggering address, which is stored in a special buffer named PreMiss, to look up MHT-1. Domino predictor updates the row with writing part of PreMiss as tag and the current triggering address as the prediction. Domino predictor also sets the valid bit of the row. Moreover, Domino predictor uses the hash of the two previous triggering addresses to look up MHT-2. The second previous triggering address is stored in a special buffer named Pre2Miss. The predictor updates the row with parts of Pre2Miss and PreMiss bits as the tags and the current triggering address as the prediction. It also sets the valid bit of the row. Finally, the predictor updates PreMiss and Pre2Miss.

Replaying. As a result of low overprediction rate, Domino prefetcher issues prefetches directly into the L1-D cache. Upon a triggering event (either an L1-D miss or a successful use of a prefetched block), the Domino prefetcher looks up the MHT-1 with the address of the triggering event and MHT-2 with the hash (i.e., XOR) of the address of the triggering event and the content of PreMiss. If MHT-2 lookup results in a match, regardless of the status of the MHT-1 lookup, a prefetch request will be sent for the prediction in MHT-2. Otherwise, if MHT-1 lookup results in a hit, a prefetch request will be sent for the prediction in MHT-1. If the lookups of both MHT-1 and MHT-2

TABLE 1
Evaluation parameters.

Parameter	Value
Processing Nodes	UltraSPARC III ISA, Sixteen 3 GHz OoO cores 8-stage pipeline, 4-wide dispatch/retirement 128-entry ROB, 48-entry LSQ
L1-I Caches	32KB, 2-way, 2-cycle load-to-use, PIF-enabled [7]
L1-D Caches	32KB, 2-way, 2-cycle load-to-use
L2 NUCA Cache	Unified, 8 MB, 16-way, 64 MSHRs 20-cycle hit latency
Main Memory	40 ns access latency, 64-byte transfers
Interconnect	4×4 2D mesh, three cycles per hop

fail to find a match, no prefetch request will be sent.

As both recording and replaying require access to the tables, and due to the fact that replaying is on the critical path but the recording is not, Domino prefetcher prioritizes replaying over recording. Only when replaying is done, Domino prefetcher attempts to follow the steps necessary for recording.

Prefetching Degree. Just like other prefetchers, Domino prefetcher may issue more than one prefetch request on a triggering event by using the already prefetched address to look up the tables. While increasing the degree of prefetching enhances the timeliness of prefetch requests, it may decrease the accuracy of the prefetcher. To overcome the inaccuracy of multi-degree prefetching, Domino just uses MHT-2 for additional prefetch requests. Although the process of issuing additional prefetch requests can be repeated as long as the predicted pattern is found in MHT-2, empirically we found that allowing up to three extra prefetch requests is a good choice for preserving both accuracy and timeliness.

4 Methodology

Table 1 summarizes key elements of our methodology. Our target is a 16-core processor with 8 MB of last-level cache. Cores have 32 KB L1-I and L1-D caches. Cache line size is 64 bytes.

We use server workloads from CloudSuite [8]. The workloads include Data Serving, MapReduce, Media Streaming, SAT Solver, Web Frontend, and Web Search. We consider two MapReduce workloads – text classification (MapReduce-C) and word count (MapReduce-W).

We estimate the performance of various processor designs using Flexus full-system simulator [9]. Flexus uses the SimFlex multiprocessor sampling methodology [10]. Our samples are drawn over an interval of 10 seconds (30 seconds for Media Streaming) of simulated time. For each measurement, we launch simulations from checkpoints with warmed caches and branch predictors, and run 450 K cycles to achieve a steady state of detailed cycle-accurate simulation before collecting measurements for the subsequent 50 K cycles. Performance measurements are computed with 95% confidence and an average error of less than 5%.

As we want to evaluate prefetching algorithms, we devote infinite area for storing the meta-data of all prefetchers. We evaluate and compare Domino with the following prefetchers.

Irregular Stream Buffer. ISB [4] combines the use of PC localization and address correlation. We implement idealized PC/AC with an infinite-size zero-cycle history table. It has been shown that the idealized PC/AC has significantly better coverage and accuracy as compared to its practical implementation [4]. Other parameters are taken from the original proposal [4].

Sampled Temporal Memory Streaming. STMS [3] records miss sequences in a global per-core history buffer (named CMOB) and locates streams through an index table. STMS employs a fully-associative buffer next to an L1 cache (named SVB) and puts the prefetched blocks in it for preventing cache pollution. It also benefits from a stream-end detection mechanism to reduce pollution. We implement STMS with infinite-size zero-cycle CMOB, index table, and SVB. As all meta-data tables are located on-chip, we set the sampling probability to 100%. Other parameters are taken from the original proposal [3].

Variable Length Delta Prefetcher. We include VLDP [11] because it has similarities with the lookup mechanism of Domino. Unlike Domino, VLDP is a prefetcher that relies on spatial correlation for prefetching and benefits from multiple previous deltas (the difference between two successive miss addresses in a page) for lookup. We equip VLDP with 16-entry DHB, 64-entry OPT, and three infinite-size DPTs. The delay of all components is set to zero. Other parameters are taken from the original proposal [11].

Digram. Like STMS, Digram [6] stores misses in CMOB and locates streams through an index table. We include Digram because, like Domino, it uses two misses for locating streams (but unlike Domino it does not look up the history with one miss address). We equip Digram with an infinite-size fully-associative SVB, and infinite-size CMOB and index table (all with zero latency). We also add STMS' stream-end detection mechanism to Digram. Other parameters are taken from the original proposal [6].

5 Evaluation Results

Figure 5 shows the coverage and overpredictions of the competing prefetching techniques. Covered misses are the ones that successfully are eliminated by a prefetcher (not necessarily restricted to the next miss). Overpredictions are incorrectly prefetched cache blocks, which cause bandwidth overhead and potentially pollute the cache or the special buffer that is used for prefetching. The incorrect prefetches are normalized against the number of L1-D cache misses in the baseline processor. On average, Domino increases the coverage of best-performing prefetcher (i.e., STMS) by 10% and comes within 13% of an ideal temporal prefetcher (i.e., Sequitur). With respect to overpredictions, Domino comes within 3% of the best-performing prefetcher (i.e., Digram).

Corroborating prior work [6], our results show that PC localized prefetchers (e.g., ISB) are not useful in the context of server workloads. VLDP works poorly because, unlike LLCs, L1 caches cannot greatly exploit the spatial correlation of data accesses due to the low residency of data in the cache [12]. Digram loses significant opportunity because it cannot predict for a stream until it observes two misses of that stream [6]. STMS relies on just a single miss, and hence, frequently picks wrong streams and prefetches incorrectly.

Figure 6 shows the performance improvement of Domino prefetcher along with ISB, VLDP, Digram, and STMS, over a baseline with no prefetcher. The average performance improvement of Domino prefetcher over the baseline is 26% (56% on MapReduce-C). The second best prefetcher is STMS with the average performance improvement of 18%.

In six out of seven workloads, Domino outperforms other prefetchers, thanks to its higher coverage. For SAT Solver, Domino is the third best prefetcher. With SAT Solver, Domino cannot frequently pick the right stream even using last two

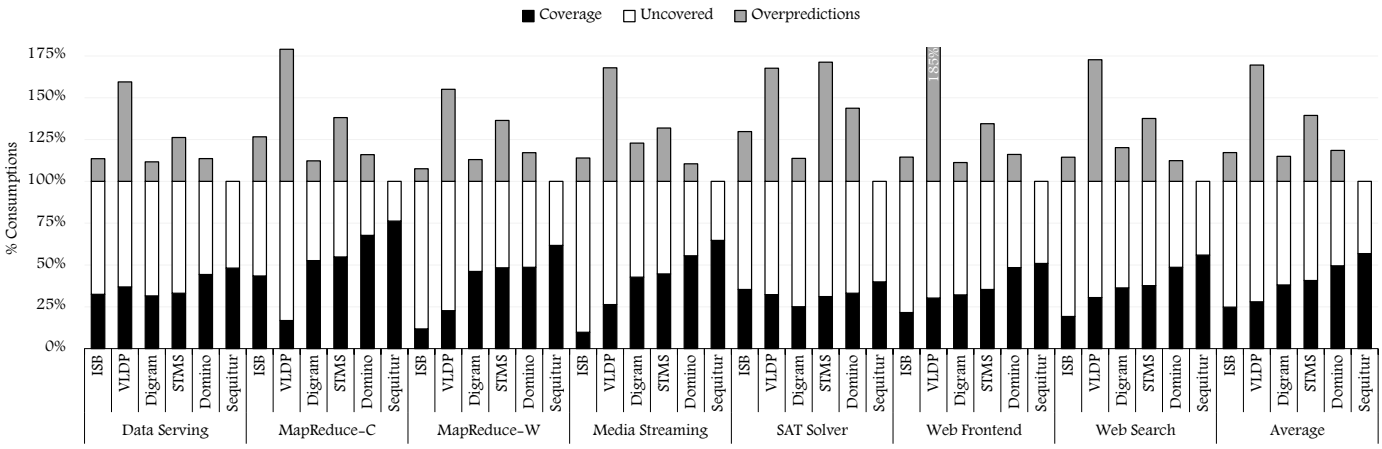

Fig. 5. Coverage and overpredictions of Domino prefetcher and the competing prefetchers.

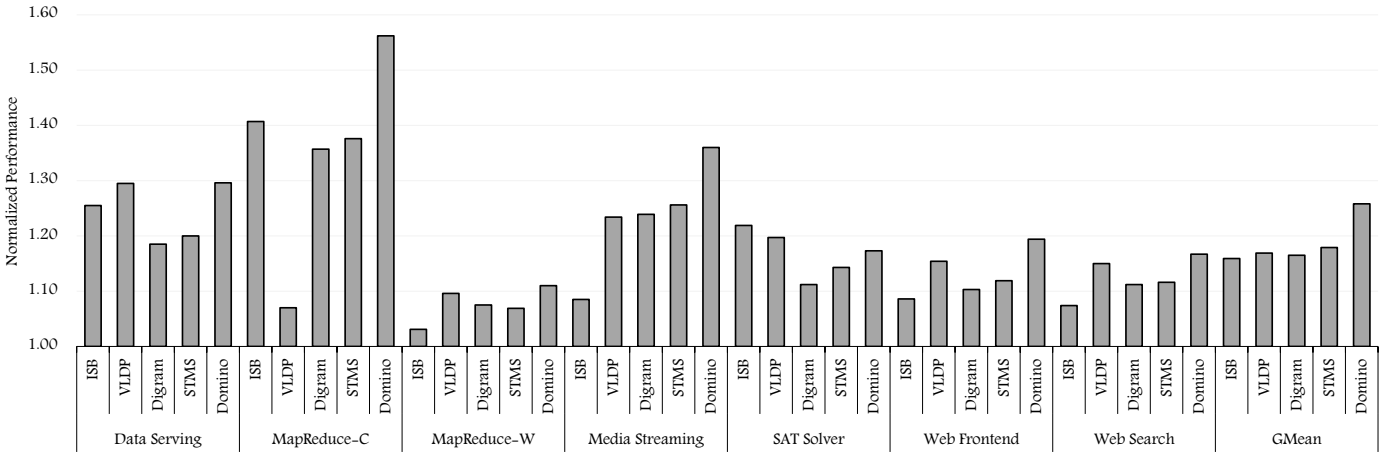

Fig. 6. Performance improvement of Domino prefetcher and the competing prefetchers over a baseline with no prefetcher.

misses. Consequently, it offers lower coverage and performance improvement as compared to ISB, which separates streams based on their PCs and finds patterns in localized streams successfully.

6 Conclusion

L1-D cache misses are a major source of performance degradation in server workloads. Data prefetching is a technique for reducing the number of cache misses or their effect. Among various data prefetching techniques, temporal prefetching has a potential to eliminate many L1-D cache misses due to the high temporal correlation in the L1-D access sequence. Unfortunately, existing temporal prefetchers cannot capture a significant part of the opportunity. This work introduced Domino prefetcher and showed that it captures 87% of the temporal opportunity at L1-Ds. Through evaluation of a 16-core processor, we showed that Domino prefetcher improves system performance by 26% on average across a set of server workloads.

References

[1] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A Study of Emerging Scale-Out Workloads on Modern Hardware," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2012, pp. 37–48.

[2] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-Out Processors," in *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2012, pp. 500–511.

[3] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos, "Practical Off-chip Meta-data for Temporal Memory Streaming," in *Proceedings of the 15th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2009, pp. 79–90.

[4] A. Jain and C. Lin, "Linearizing Irregular Memory Accesses for Improved Correlated Prefetching," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2013, pp. 247–259.

[5] C. G. Nevill-Manning and I. H. Witten, "Identifying Hierarchical Structure in Sequences: A Linear-time Algorithm," *Journal of Artificial Intelligence Research*, vol. 7, no. 1, pp. 67–82, Sep. 1997.

[6] T. F. Wenisch, "Temporal memory streaming," Ph.D. dissertation, Carnegie Mellon University, Jul. 2007.

[7] M. Ferdman, C. Kaynak, and B. Falsafi, "Proactive Instruction Fetch," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2011, pp. 152–162.

[8] CloudSuite, http://cloudsuite.ch.

[9] Flexus, http://parsa.epfl.ch/simflex/flexus.html.

[10] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "SimFlex: Statistical Sampling of Computer System Simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, July-August 2006.

[11] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently Prefetching Complex Address Patterns," in *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2015, pp. 141–152.

[12] D. Jevdjic, S. Volos, and B. Falsafi, "Die-Stacked DRAM Caches for Servers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2013, pp. 223–230.