Return Data Interleaving for Multi-Channel Embedded CMPs Systems

Fei Hong, Aviral Shrivastava, and Jongeun Lee

***Abstract*—Using multi-channel memory subsystems is an efficient way of satisfying high volume memory requests from CMPs. At the same time, the imbalance between memory bandwidth and bus performance opens up new possibility of optimization before they are sent to bus. This paper presents a new memory controller design for embedded CMPs systems when the return data from the return buffer is sent back to bus. Our scheduling policy, called return data interleaving (RDI) interleaves the return data of each request in a round robin manner. Further, for each request, it sends the critical word first. To evaluate our technique, we model an Intel XScale-based CMPs using M5 simulator for CMPs simulation and DRAMsim for memory subsystem simulation and examine the performance of MiBench and SPEC2000 benchmarks. Simulation results show that for memory-bound benchmarks running on the CMPs systems with the number of cores from 6 to 16, RDI can improve the execution time by average 11% and up to 16.9%.**

***Index Terms*—Chip multi-core processor, multi-channel memory, return data interleaving (RDI).**

I. Introduction

Processor and memory performance have been increasing in the past decades, but at significantly different increasing rates [21]. Memory latency continues to remain a significant performance bottleneck [6]. As we turn to CMPs for higher performance it places more pressure on the memory subsystem. In order to maintain the same amount of off-chip bandwidth per core, the total off-chip bandwidth for the processor chip must also double every process generation [18]. This bandwidth increase could be met by increasing the number of pins and/or increasing the bandwidth per pin. However, the maximum number of pins per package is increasing gradually at a rate of 10% per generation [1]. Furthermore, packaging costs increase significantly with pin count [1]. Therefore, the limited width of processor—memory bus is an important bottleneck in CMPs systems.

In a multi-channel memory subsystem, each DRAM channel is controlled by a DRAM controller and works independently and therefore can alleviate the memory bandwidth bottleneck. Multi-channel memory subsystems open new opportunities for optimization and better use of the processor-memory bus. In particular, there is opportunity to optimize the way data is sent back to the processor after the requests are satisfied by DRAM channels. In this paper, we propose return data interleaving (RDI), which interleaves the return data of each request in a round robin manner. It sends the critical words of each request to the system bus first and sends the rest words of each request next.

To evaluate our approach, we model each core in the CMPs system based on the Intel XScale microarchitecture [3] for each core using the M5 simulator [2] for the CMPs processor and DRAMsim [19] for memory subsystem. We then implement RDI technique and examine the execution of MiBench benchmarks in the multi-dimensional design space of CMPs. Simulation results show that RDI is most effective when there is slight imbalance between the processor demands and memory bandwidth (with bus width of 8- and 16-byte and the number of cores between 12 and 16). For memory-bound benchmarks running on the CMPs systems with the number of cores from 6 to 16, RDI-based memory controller can improve the execution time by average 11% and up to 16.9%.

II. Prior Work on Improving Memory Bandwidth

Techniques for improving memory bandwidth mainly focus on eliminating bank conflicts [20]. Skewing [4], prime memory systems [15], [16], and compiler transformations [12] attempt to arrange the order of memory commands to minimize bank conflicts. Arbitrated memory access [14] dynamically eliminate bank conflicts by enforcing a strict round robin ordering of bank accesses. Adaptive history-based memory scheduling [5] maintains information about the state of the DRAM along with a short history of previously scheduled operations. While some techniques focus on avoiding bank conflicts, stream memory controller [9]–[11] maximizes the bandwidth for streaming applications, by focusing on: 1) hitting the hot row; 2) avoiding bank conflicts; and 3) avoiding switching between Reads and Writes. Parallel vector access [8], [17] present a scheme to maximize row hits in open-page memory systems.

Most previous efforts have focused on scheduling the memory requests before they are sent to DRAMs. The interval after the requests are satisfied but before they are sent to system bus has not been paid much attention to. Wrap-around Fill [7] is the only scheme that we know of, that works on return data. However, this technique only focuses on the single request. In contrast, we propose to interleave the words of multiple requests. Our scheduling policy, called RDI, interleaves the return data of each request in a round robin manner. In addition, it sends the critical words of each request to system bus first and sends the rest words of each request next.

III. Our Approach

A. Illustration of RDI

When there is a cache miss, the processor core makes a request for a word (the critical word) to the memory. However, as a response, memory sends a block of words containing the requested word to the processor caches. While the rest of the words in the block may be used in the near future by the processor (thereby exploiting spatial locality), but they are not as urgently needed as the critical word. Instead of sending the words in the block in normal increasing address manner, critical word first scheme sends the critical word for a request first, followed by the rest of the words in the block.

In multi-channel memory subsystem, when several memory requests are pending in processor, sending the return data of requests serially (as in conventional memory) is not how the processor needs the data. In order to send the data to the processor in the way processor needs it, we propose to interleave the words of multiple requests in the Return Buffer (RB), which is used to cache data before sending them to bus. We call this scheme Return Data Interleaving, or simply RDI. We send

Manuscript received July 26, 2010; revised February 04, 2011; accepted March 12, 2011.

F. Hong and A. Shrivastava are with the School of Computing Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281 USA (e-mail: fei.hong@asu.edu; aviral.shrivastava@asu.edu).

J. Lee is with the the School of Electrical and Computer Engineering, Ulsan National Institute of Science and Technology, Ulsan 689-798, South Korea (e-mail: jlee@unist.ac.kr).

This work was supported in part by the National Science Foundation (NSF) under Grants CCF-0916652, CCF-1055094 (CAREER) and by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology, under Grant 2010-0011534.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2011.2157368

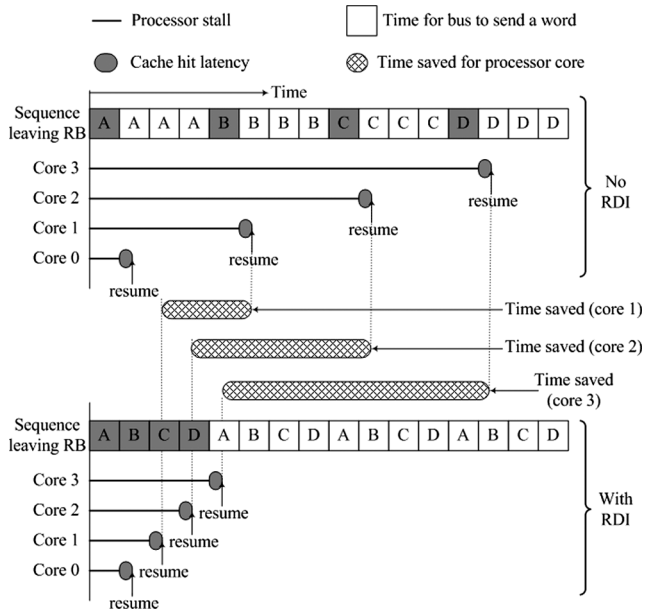

Fig. 1. Comparison between interleaving and non-interleaving. Core 0 resumes at the same time in the two examples. The other cores resume earlier with RDI. With RDI those cores will get the CWs earlier and can resume running earlier.

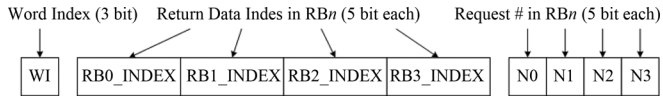

Fig. 2. Additional information needed for RDI in hardware. WI is responsible for locating the word inside a return data, while RBn_INDEX is responsible for locating the return data in each RB. Nn stores the number of return data available for interleaving in RBn.

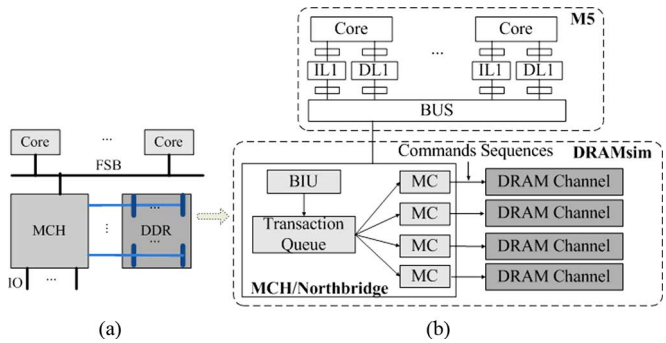

Fig. 3. (a) Architecture of CMP systems with multi-channel memory subsystem. MCH handles all the memory requests from process and other devices. (b) Hardware architecture modeled by simulators. M5 simulator is for building the whole system architecture, DRAMsim is for modeling memory subsystem.

the critical words of the requests first and then send the rest of the data to the corresponding caches.

Fig. 1 illustrates the working of RDI. Suppose the return data of 4 requests (A to core 0, B to core 1, C to core 2 and D to core 3) are in the RB. Each request's return data is divided into four words and the dark blocks represent the critical words. Assume it takes 2 bus cycles to send a word of return data. In a conventional memory architecture (critical word first with no interleaving), the critical words of four requests arrive at the cache after 2, 10, 18, 26 bus cycles. With RDI technique, we will send the critical words of each request first and then send the non-critical parts. We can see that the critical words of the four requests arrive at corresponding cache after 2, 4, 6, 8 bus cycles. Thus, the waiting time of processor cores can be reduced by sending the critical words first.

B. Implementation and Hardware Cost

An implementation of RDI requires storing additional information for each RB in the DRAM controllers to let the memory controller know with word inside a return data should be sent at a given cycle. Fig. 2 shows the required additional information added to the memory controller. From experiments we found that in a 4-channel memory subsystem, the maximum total number of return data of each RB from 4 channels can be up to around 100, which means in average each RB could have up to 25 return data queuing up in the return buffer. Here we use 5 bits for each RB ($N0$ to $N3$) indicating the number of return data available in each RB for interleaving. RBn_INDEX stores the return data index indicating which return data the memory controller should send in each RB, after all the words in a return data from RBn is send to bus, the index value in RBn_INDEX will increase by 1, indicating the index of next return data will be send. The size of these four buffers is same as the size of Nn, each of which is 5 bits. The value in WI buffer indicates which word in a return buffer should be sent currently. For a bus width of 8-byte, a 64-byte cache line will be divided into 8 words and WI will need 3 bits to indicate 8 different words. The four RB share the same value in the WI buffer since the words inside a return data is already reordered and the order of sending words in the return data is the same for all return data in RB. After each RB sends a word in a round the value in WI will increase by 1, indicating the next word index the RB should send in the next round. At any given cycle, the memory controller finds the word to send according to the value in WI, RBn_INDEX, and Nn, in a round-robin manner through $RB0$ to $RB1$. WI is responsible for locating the word inside a return data, while RBn_INDEX is responsible for locating the return data in each RB.

We quantify the cost of our design based on two metrics, the amount of storage required in bits and the number of bit-comparisons. The storage cost for a system employing n-channel memory subsystem, w-byte width bus, and c-byte cache line, is $(\log 2(c/w)+10n)$ bits. For a typical configuration (8-byte bus width, 64-byte cache line, 4-channel memory subsystem), we need only 43 bits. For sending a return data, we need 8 comparisons for locating each word and 1 comparison for locating the return data, totally 9 comparisons. Therefore, we only need small amount of storage and simple comparison logic additionally to implement this architecture.

IV. Experimental Setup

We evaluate the effectiveness of our technique across a range of reasonable architectural parameters and wide variety of application memory behaviors. To model a range of architectural parameters, we have developed a parameterizable simulation environment to model a typical shared-bus CMPs system [13] [illustrated in Fig. 3(a)]. Fig. 3(b) shows the CMP system we model. We simulate the processor using the cycle-accurate M5 [2] and the memory subsystem using the detailed memory simulator DRAMsim [19]. We integrate DRAMsim into M5 to form a CMPs system.

The architectural parameters of each core for the baseline processor are based on the embedded Intel XScale microprocessor [3]. We assume a fixed sized die with the area equal to 8 XScale microprocessors and caches comprise approximately 40% of the die area. We scale the processor architecture to different number of cores and explore system bus width, from 4-byte to 32-byte to model a range of embedded multi-core systems [1]. We experiment with memory subsystems with

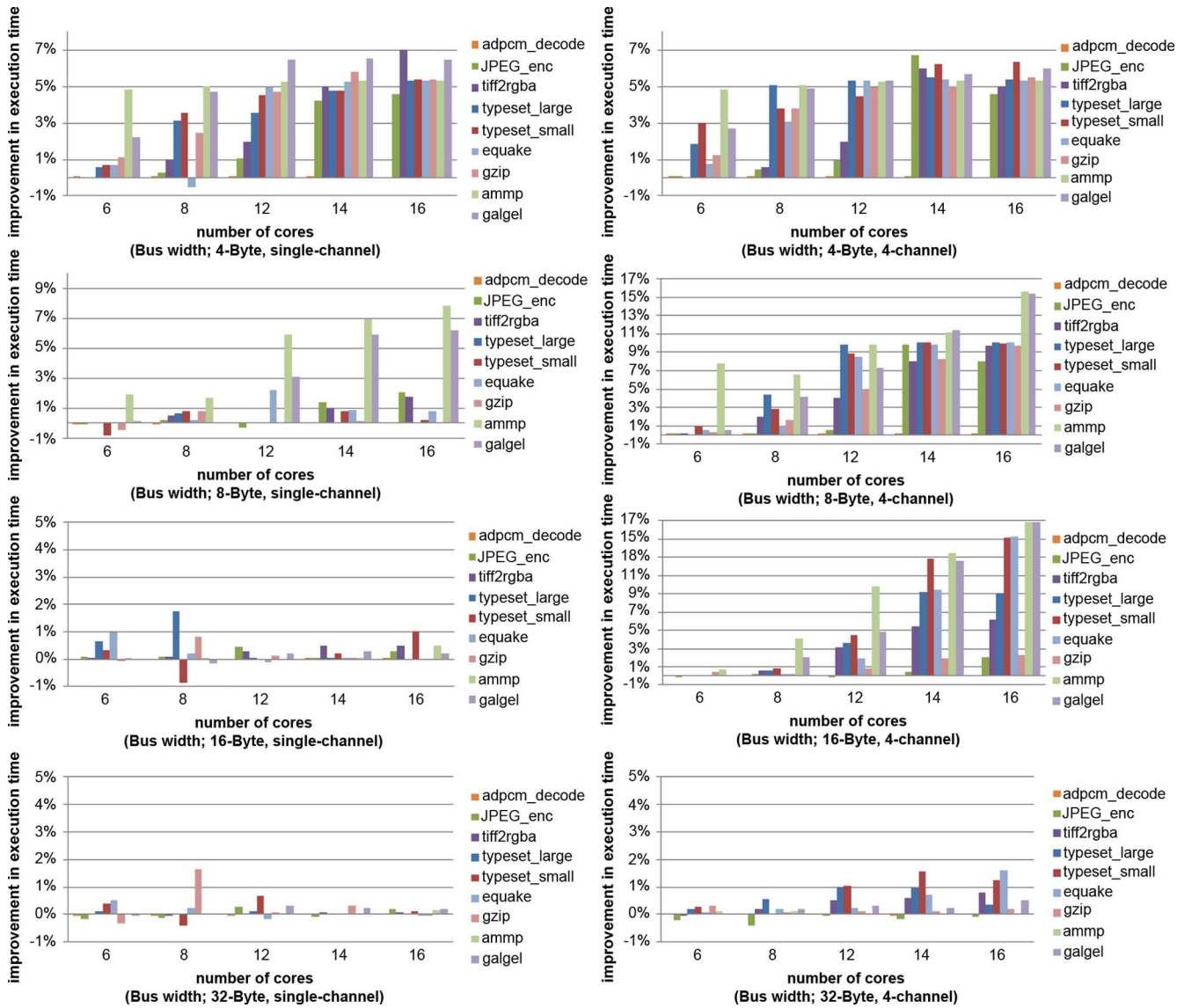

Fig. 4. Improvement on execution time using RDI technique compared to no RDI used in single and 4-channel memory subsystem. RDI is useful for benchmarks with memory-bound configurations. Bus width is an important factor that affects the effectiveness of RDI technique. The improvement can be up to 16.9% for memory-bound benchmarks running with 8- or 16-byte bus width.

TABLE I
BASELINE CONFIGURATIONS OF MEMORY SUBSYSTEM

DRAM Controller	**Open Page Fist transaction select policy; 1 controller/DRAM channel, 128-entry BIU; Critical Word First, Wrap-Around-Fill;**
DRAM and internal bus	**DDR2 400MHz, 8B width/ DRAM channel; 8 banks/rank, 4 ranks/channel; Rank size: 4 DRAM/bank * 8 banks/rank * 8192 rows/bank * 512 columns/row = 256MB**

up to 4 DRAM channels. Table I shows the parameters of the shared memory subsystem configuration.

We categorize application-architecture configuration as: Processor-bound (if the working sets are captured easily in the caches and require few external DRAM accesses) and Memory-bound (if the performance is limited by cache and the rate that data can be moved between the processor and the DRAM). An important point to note here is that the benchmarks are not bound to one class or another. They move among these domains as the processor, cache and bus width capacities are modulated.

V. EXPERIMENTAL RESULTS

Fig. 4 plots the improvement in execution time by using RDI for each application and configuration. RDI is useful in the single-channel memory systems with 4-byte bus width for the memory-bound configurations (typeset with more than 6 cores, JPEG and gsm with more than 12 cores). This is because the 4-byte bus width is too small and causes return data queue up in the return buffer (RB). RDI works when the return data of multiple requests exist in the RB. However, the improvement is limited (less than 7%) because 4-byte bus width is too small. With larger bus width, there is not enough return data in the RB in single channel memory subsystem and therefore RDI is not useful.

The improvements in execution time for the memory-intensive configurations with 4-byte bus width are similar between 4-channel and single-channel configurations. When the bus width becomes larger (8-byte), the transmission rate increases significantly and still the return data of multiple requests queues up in the RB and the improvement becomes significant (up to 12.8%). When the bus width is increased to 16-byte, the improvement in execution time for some benchmarks decreases because high bus transmission rate reduces the

return data in the RB; but the improvement can still be up to 16.9% for the memory-bound benchmarks (ammp). Finally when the bus width is increased to 32-bytes, there is very little improvement (1% for typeset, almost 0% for the other benchmarks) because of the increased bus transmission rate and not enough return data in the RB.

There is very little improvement (no improvement for adpcm) for the processor-bound configurations. This is because they issue fewer memory requests and therefore there is not enough return data queuing up in the RB. Also note that that in some cases the execution time increases when we use RDI. This is because RDI changes the order of return data returning to processor and subsequent order of issuing requests among the cores will also be different. This combined with the fact that the latency of a memory request accessing a bank varies depending on the state of the row buffer [5], we may have different execution times. However, the degradations in runtime are less than 0.2%.

Experimental results demonstrate that using multiple channels is very effective in reducing execution time and RDI can further improve the execution time, especially when the execution time is limited by the processor-memory bandwidth bottleneck. RDI is useful when there are frequent memory references (typically with more than 10 memory references/1000 instructions). What is more, the bus width can neither be too small (for example 4-byte, which will limit the performance), nor too large (for example 32-byte, which will make not enough return data queuing up in the RB). In this sense, RDI is able to exploit slight imbalance between the processor demands and memory bandwidth.

In addition, it is relatively easier to increase the performance of both the processor and memory subsystems but more difficult to the increase bus width [1]. The limited width of interconnect between processor and memory subsystem is rapidly becoming the most important roadblock to improving the performance of the processor-memory system as a whole. Note that besides the example embedded system configurations in the experiment section, this limited bus width problem exists in all set of high performance computer systems. Therefore, optimizing the way of sending the return data back to processor is necessary when we employ high performance memory subsystems. For most embedded systems, the system bus width is less than 8 bytes and as shown in the experiments RDI technique will be useful for high performance CMPs embedded systems.

VI. Conclusion

This paper shows that in multi-channel memory subsystems running memory intensive applications, when DRAM bandwidth is larger than the bus width, there is scope for optimizing the manner in which the data is returned from the memory to the processor. This paper proposes RDI, in which we interleave the return data of those requests by sending back the critical words of each request to the system bus first and then send the rest words in a wrap around way. While critical word first is a known scheme to reorder the words in individual return data, there was no previous proposal on interleaving the words of multiple return data. Our simulation results show that, as compared to no RDI on 4-channel memory subsystem, RDI-based memory controller can improve the execution time by average 11% and up to 16.9%.

References

[1] ITRS, "International Technology Roadmap for Semiconductors: Executive Summary," 2005.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The m5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, pp. 52–60, 2006.

[3] L. T. Clark, E. J. Hoffman, J. Miller, M. Biyani, Y. Liao, S. Strazdus, M. Morrow, K. E. Velarde, and M. A. Yarch, "An embedded 32-b microprocessor core for low-power and high-performance applications," *IEEE J. Solid-States Circuits*, vol. 36, no. 11, pp. 1599–1608, Nov. 2001.

[4] Q. S. Gao, "The Chinese remainder theorem and the prime memory system," in *Proc. 20th Annu. Int. Symp. Comput. Arch. (ISCA)*, 1993, pp. 337–340.

[5] I. Hur and C. Lin, "Memory scheduling for modern microprocessors," *ACM Trans. Comput. Syst.*, vol. 25, 2007.

[6] W.-F. Lin, "Reducing dram latencies with an integrated memory hierarchy design," in *Proc. 7th Int. Symp. High-Perform. Comput. Arch. (HPCA)*, 2001, p. 301.

[7] S. G. Lloyd, "Burst mode cache with wrap-around fill," U.S. Patent No. 4 912 631, Mar. 27, 1990.

[8] B. K. Mathew, S. A. McKee, J. B. Carter, and A. Davis, "Algorithmic foundations for a parallel vector access memory system," in *Proc. 12th Annu. ACM Symp. Parallel Algorithms Arch. (SPAA)*, 2000, pp. 156–165.

[9] S. A. McKee, "Maximizing memory bandwidth for streamed computations," Ph.D. dissertation, Dept. Comput. Sci., Univ. Virginia, Charlottesville, VA, 1995.

[10] S. A. Mckee, S. A. Mckee, and S. A. Mckee, "Hardware support for dynamic access ordering: Performance of some design options," 1993.

[11] S. A. McKee, W. A. Wulf, J. H. Aylor, M. H. Salinas, R. H. Klenke, S. I. Hong, and D. A. B. Weikle, "Dynamic access ordering for streamed computations," *IEEE Trans. Comput.*, vol. 49, no. 11, pp. 1255–1271, Nov. 2000.

[12] S. A. Moyer, "Access ordering and effective memory bandwidth," Ph.D. dissertation, Dept. Comput. Sci., Univ. Virginia, Charlottesville, VA, 1993.

[13] C. Natarajan, B. Christenson, and F. Briggs, "A study of performance impact of memory controller features in multi-processor server environment," in *Proc. 3rd Workshop Memory Perform. Issues (WMPI)*, 2004, pp. 80–87.

[14] M. Peiron, M. Valero, E. Ayguadé, and T. Lang, "Vector multiprocessors with arbitrated memory access," *SIGARCH Comput. Arch. News*, vol. 23, no. 2, pp. 243–252, 1995.

[15] R. Raghavan and J. P. Hayes, "On randomly interleaved memories," in *Proc. ACM/IEEE Conf. Supercomput. (Supercomputing)*, 1990, pp. 49–58.

[16] B. R. Rau, "Pseudo-randomly interleaved memory," *SIGARCH Comput. Archit. News*, vol. 19, no. 3, pp. 74–83, 1991.

[17] B. M. Sally, S. A. Mckee, J. B. Carter, and A. Davis, "Design of a parallel vector access unit for SDRAM memory systems," in *Proc. 6th Annu. Symp. High Perform. Comput. Arch.*, 2000, pp. 39–48.

[18] L. Spracklen and S. G. Abraham, "Chip multithreading: Opportunities and challenges," in *Proc. 11th Int. Conf. High-Perform. Comput. Arch. (HPCA-11)*, 2005, pp. 248–252.

[19] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "DRAMsim: A memory system simulator," *ACM SIGARCH Comput. Arch. News*, vol. 33, no. 4, pp. 100–107, Sep. 2005.

[20] D. T. Wang, "Modern dram memory systems: Performance analysis and scheduling algorithm," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Maryland, College Park, MD, 2005.

[21] W. A. Wulf and S. A. Mckee, "Hitting the memory wall: Implications of the obvious," *Comput. Arch. News*, vol. 23, pp. 20–24, 1995.