Design and Optimization of Large Size and Low Overhead Off-Chip Caches

Zhao Zhang, *Member*, *IEEE*, Zhichun Zhu, *Member*, *IEEE*, and Xiaodong Zhang, *Senior Member*, *IEEE*

Abstract—Large off-chip L3 caches can significantly improve the performance of memory-intensive applications. However, conventional L3 SRAM caches are facing two issues as those applications require increasingly large caches. First, an SRAM cache has a limited size due to the low density and high cost of SRAM and, thus, cannot hold the working sets of many memory-intensive applications. Second, since the tag checking overhead of large caches is nontrivial, the existence of L3 caches increases the cache miss penalty and may even harm the performance of some memory-intensive applications. To address these two issues, we present a new memory hierarchy design that uses cached DRAM to construct a large size and low overhead off-chip cache. The high density DRAM portion in the cached DRAM can hold large working sets, while the small SRAM portion exploits the spatial locality appearing in L2 miss streams to reduce the access latency. The L3 tag array is placed off-chip with the data array, minimizing the area overhead on the processor for L3 cache, while a small tag cache is placed on-chip, effectively removing the off-chip tag access overhead. A prediction technique accurately predicts the hit/miss status of an access to the cached DRAM, further reducing the access latency. Conducting execution-driven simulations for a 2GHz 4-way issue processor and with 11 memory-intensive programs from the SPEC 2000 benchmark, we show that a system with a cached DRAM of 64MB DRAM and 128KB on-chip SRAM cache as the off-chip cache outperforms the same system with an 8MB SRAM L3 off-chip cache by up to 78 percent measured by the total execution time. The average speedup of the system with the cached-DRAM off-chip cache is 25 percent over the system with the L3 SRAM cache.

Index Terms—Cached DRAM, DRAM latency, memory hierarchy, memory-intensive applications, off-chip caches.

1 Introduction

As the processor-memory speed gap continues to widen, application performance is increasingly dependent on the performance of memory hierarchy. The design and optimization of memory hierarchy involve trade offs among a number of factors, such as the number of cache levels, cache size, access latency, cost, and power consumption. Modern processors have included two-level caches on-chip. However, the sizes of on-chip caches are limited. For example, the 2.4 GHz Pentium 4 processor has only 512KB L2 cache. Many applications have large working sets that cannot fit into such on-chip caches. Architects have provided a large off-chip L3 cache to further reduce the number of main memory accesses. For example, an early study [9] shows that adding a 2MB off-chip L3 cache to AlphaServer 4100 (with 96 KB on-chip L2 cache) can improve the performance of memory-intensive workloads by 20 percent.

Ideally, an off-chip L3 cache should be large enough to hold the working sets for most applications and fast enough to reduce the memory access latency. Off-chip L3 caches are normally made by SRAM, the same technology used for on-chip caches. SRAM is fast but has several limitations of being off-chip caches. First, because of the low density and high cost of SRAM, the size of an SRAM cache is usually limited to less than 10 megabytes, which is not large enough for of many memory-intensive applications. As Fig. 1 shows, the cache miss rates of the selected SPEC 2000 benchmarks drop significantly as the cache size increases beyond a certain point. For example, the number of misses per 100 instructions of 179.art drops from 1.22 to almost zero as the L3 cache size increases from 2MB to 4MB and that value of 189.lucas drops from 1.17 to 0.24 as the cache size increases from 8MB to 32MB. A 2MB SRAM cache will benefit none of these applications, while a 4MB cache will benefit only one application (179.art).

Second, because the volume of L3 tags is large and the tags are usually stored off-chip, the L3 tag checking overhead is significant and increases L3 cache miss penalty. The performance of some memory-intensive applications can be harmed by the use of off-chip cache. For instance, a study on AlphaServer [9] reports that, because of the increased memory access latency, the off-chip cache can degrade the performance by up to 10 percent for those applications whose working sets cannot fit into the off-chip cache.

Since constructing the off-chip L3 cache with SRAM cannot satisfy the capacity requirements for many memory-intensive applications, we propose using DRAM to substitute for SRAM to address this issue. However, simply applying a DRAM cache will significantly increase the cache access latency. In this study, we present a new off-chip cache design, named CDC (cached-DRAM cache), to

- *Z. Zhang is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011. E-mail: zzhang@iastate.edu.*
- *Z. Zhu is with the Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, IL 60607. E-mail: zhu@ece.uic.edu.*
- *X. Zhang is with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23187. E-mail: zhang@cs.wm.edu.*

Manuscript received 1 Nov. 2002; revised 27 May 2003; accepted 16 Oct. 2003.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 117696.

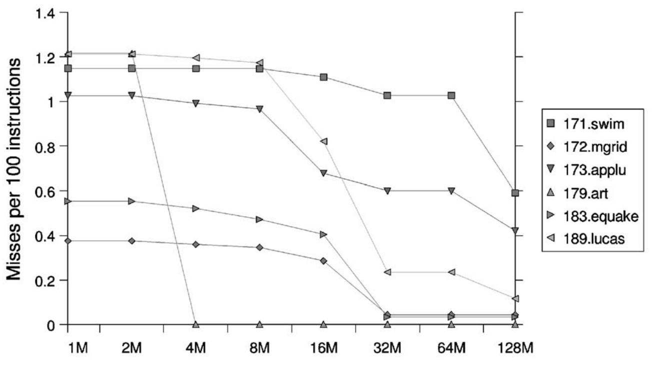


Fig. 1. The number of misses per 100 instructions for selected SPEC2000 programs, collected by SimpleScalar [5] 3.0 Alpha version and using precompiled SPEC2000 programs [42]. The cache is 8-way associative with block size of 128-byte. All programs are fast forwarded by four billion instructions and are executed for the next one billion instructions.

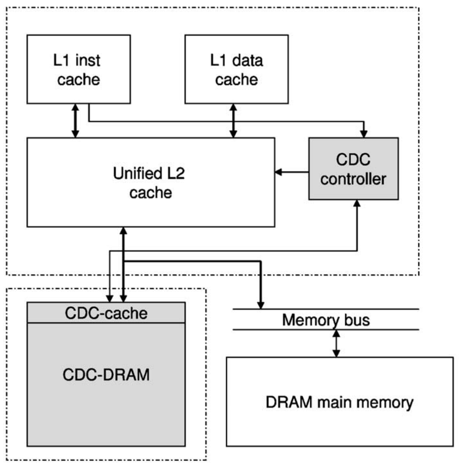


Fig. 2. A diagram of memory hierarchy using CDC.

achieve both high capacity and low latency by using cached DRAM to construct the off-chip L3 cache.

This design can improve the performance of memory-intensive applications running on high-performance server systems. Cached DRAM integrates a small SRAM cache into DRAM to exploit the spatial locality that appears in miss streams from the upper level cache [15], [16], [23], [50]. Thus, it has a much shorter average access latency than that of conventional DRAMs. The off-chip cache constructed by cached DRAM has the potential to achieve average access latency close to that of SRAM cache. The unique advantages of the CDC are *its large capacity, equivalent to the DRAM size, and its low average latency, close to that of an SRAM cache.*

Our CDC design has three features that distinguish it from a simple design of DRAM cache. First, it adopts the structure of a sector cache [31] in order to achieve a high degree of spatial locality without increasing the memory bandwidth consumption. Normally, the increase of spatial locality requires an increase on cache block size, which will in turn increase the memory bandwidth usage for a cache fill request. With the structure of a sector cache, it is only necessary to fetch the missed subblock when a CDC miss happens. In our CDC design, the block size equals the DRAM page size and the subblock size equals the on-chip L2 cache block size. Without increasing the memory bandwidth consumption, the block size of CDC can be very large (e.g., 32 times the L2 cache block size). This allows the SRAM cache inside the CDC to exploit the spatial locality in the CDC hit streams so as to reduce the average CDC access time.

Second, our CDC design only requires a small amount of logic inside the processor chip by placing the CDC tag off-chip. In the design of the off-chip cache, the use of on-chip tags allows fast tag checking. However, because the tags occupy a large amount of on-chip space, this approach will limit other resources that are allowed on-chip and may harm the performance of those non-memory-intensive applications. In our design, a very small on-chip tag cache is used to facilitate fast tag checking. The details of the on-chip tag cache will be presented in Section 2.

Finally, the CDC design removes most of the additional miss penalty associated with off-chip caches by using a small on-chip hit/miss predictor. For a predicted miss on the off-chip CDC, the request to the main memory will be sent out immediately after the L2 cache miss is detected. Our experimental results show that a simple and small predictor is highly accurate for this purpose.

We use SimpleScalar to simulate a system with a 4-way issue 2GHz processor, split 32KB/32KB L1 caches, a unified 1-MByte on-chip L2 cache, and an off-chip CDC of 128KB SRAM and 64MB DRAM. We compare the performance of the CDC with an 8-MByte SRAM L3 cache under the same processor configuration, for 11 memory-intensive programs from the SPEC CPU2000 benchmark suite. Our results show that the CDC outperforms the L3 SRAM cache for most programs by up to 51 percent. Unlike the off-chip SRAM cache, the CDC does not degrade the performance of any program. The average performance improvement is 25 percent.

The rest of the paper is organized as follows: The next section presents the CDC design in detail. Section 3 describes the experimental setup for our study. The experimental results are presented in Section 4. After discussing the related work in Section 5, we summarize this study and briefly discuss future work in Section 6.

2 CDC Design

2.1 Overview

Fig. 2 presents a memory hierarchy with two-level on-chip caches and an off-chip CDC (Cached-DRAM cache). Cached DRAM is a special DRAM structure that integrates a small SRAM cache to exploit the spatial locality appearing in the access streams [15], [16], [23], [50]. Inside the CDC, the DRAM storage is called *CDC-DRAM* and the SRAM portion is called *CDC-cache*. To solve the problem of long tag check penalty of L3 cache, we use an on-chip predictor to predict whether an access to the DRAM chip is a hit or a miss. The predictor helps eliminate the miss penalty for correctly predicted CDC misses.

We highlight the CDC design as follows:

- CDC-DRAM has the structure of sector cache, where each CDC-DRAM page can hold dozens of on-chip cache blocks.
- Recently accessed CDC-DRAM pages are cached in the CDC-cache and their tags are stored in the on-chip *CDC tag cache*.
- The CDC is connected to the processor by high-bandwidth data paths. We assume that they are put in the same module by using the multichip module (MCM) technology.

Besides providing a much larger storage than an SRAM off-chip cache, the proposed CDC has the following additional advantages in constructing the L3 off-chip cache:

- *Minimizing on-chip space overhead*. The CDC tags are stored with data pages in the CDC-DRAM. Only the tags of pages cached in the CDC-cache are stored on-chip. The storage requirement for the on-chip tag cache is less than 1KB in the default configuration. The on-chip predictor for predicting CDC-DRAM hit/miss is very simple and occupies a very small space.
- *Minimizing cache miss overhead*. Since the on-chip CDC tag cache is small and the CDC-DRAM hit/miss prediction is simple, the CDC tag checking and the prediction can be performed in parallel with the L2 cache access for every L1 cache miss. When an L2 cache miss is detected, the results of the CDC tag checking and the prediction will be available. There is no additional miss overhead for correctly predicted CDC-DRAM misses. The performance results show that the prediction has high accuracy and the miss overhead related to incorrect predictions is trivial.

  In contrast, the tag directory of the SRAM L3 cache is much larger than that of the L2 cache and has longer latency and cycle time than those of the L2 cache. Thus, there will be a significant overhead for cache misses to the main memory.
- *Exploiting the locality in the CDC-Cache*. Many recent studies have shown that the locality in the cache miss streams is very high [23], [44], [8], [49], [27], [51]. Because of the large block size, the CDC-cache is able to effectively utilize this locality to speed up accesses to the CDC.

Fig. 3 shows a detailed structure of CDC and the data paths connecting the CDC-cache, the CDC-DRAM, the L2 cache, and the tag cache. Each CDC-DRAM page contains a page of data, address tags, and an array of the selection and valid bits.[1] For simplicity, the collection of the address tags and the selection and valid bits of one CDC-DRAM page is called the *page tag* of that page. Upon a CDC-DRAM miss, the refilled L2 block is sent to both the L2 cache and the CDC. Inside the CDC, the page tags are stored and transferred, together with their associated data blocks. Outside the CDC, the data blocks are selected and transferred to and from the L2 cache and the tags are transferred to and from the tag cache.

1. The CDC-DRAM uses decoupled sector cache organization, which requires selection bits; see Section 2.3 for details.

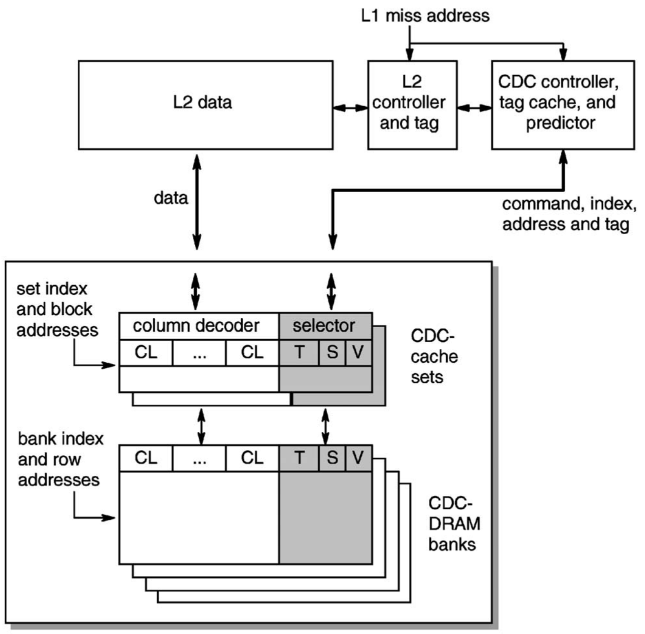

L1 miss address
L2 data
L2 controller and tag
CDC controller, tag cache, and predictor
data
command, index, address and tag
set index and block addresses
column decoder
selector
CL
...
CL
T
S
V
CDC-cache sets
bank index and row addresses
CDC-DRAM banks

Fig. 3. The CDC structure and the data paths between the CDC-cache, the CDC-DRAM, the L2 cache, and the on-chip tag cache. CL represents a block of the cache line size, T represents the address tags of a page, S represents the selection bits of a page, and V represents the valid bits of a page.

For the convenience of discussions, the following configurations and parameters are assumed if they are not mentioned specifically. The CDC-DRAM is 64-MB and the page size is 4KB. Each page is a sector holding 32 L2 cache blocks of 128 bytes. The CDC-cache is 4-way set-associative with 128KB. For the calculation of storage requirement, we assume that the physical address in the system is 48-bit long.

2.2 On-Chip CDC Controller, Tag Cache, and Predictor

The tag cache stores the tags of pages that are cached in the CDC-cache. For each L1 cache miss, the address is compared with the associated address tags in the tag cache in parallel with the L2 cache access. Since the tag cache is very small, the comparison can be done before the L2 cache access finishes. When an L2 cache miss is detected, if a match is found and the valid bit is set, the demanded data block exists in the CDC-cache and its set index and block index in the CDC-cache are known. Then, the CDC controller sends a request to fetch the block from the CDC-cache to the L2 cache (the CDC-DRAM is not involved). Although the CDC-cache is set-associative, it is accessed as a direct-mapped cache because the set index and the block index are already known at the time of the access. In other words, CDC-cache tag and data are accessed sequentially, but its tag checking overlaps with the L2 tag checking. For L2 cache hits, no data will be fetched from the CDC-cache or CDC-DRAM.

If a match cannot be found, the data block may exist in both the CDC-DRAM and main memory or only in the main memory. The hit/miss predictor then makes a prediction, directing an access to the CDC-DRAM or to the main memory (we will discuss the predictor design in Section 2.4). In the case of a CDC-DRAM access, the CDC-DRAM page, including both data and tag, is first transferred to the CDC-cache. Then, the page tag is transferred to the on-chip tag cache to match the missed address. Meanwhile, the demanded data block is selected, transferred to the processor chip, and may be buffered temporarily. If a match if found, the data block is written into the L2 cache; otherwise, it is discarded and the data block is fetched from the main memory.

In the case of a main memory access, both the CDC-DRAM and the main memory are accessed simultaneously. If the block is not found in the CDC, the data fetched from the main memory is sent to the L2 cache and is also written into the CDC-cache. If the block is found in the CDC-DRAM, the data fetched from the main memory is discarded.

The predictor enables exploiting the locality in the high-bandwidth CDC-DRAM, even if the data are not found in the CDC-cache. From another point of view, the predictor filters out unnecessary traffic to the main memory, which could cause congestion on the memory bus and stall the follow-up accesses. Working together, the controller, the tag cache, and the predictor ensure fast tag checking, fast CDC-cache accesses, and low memory bus traffic.

2.3 CDC-DRAM Mapping and CDC-Cache Mapping

The CDC-DRAM has a sector cache structure [31], where the data part of a page is a sector holding 32 L2 cache blocks. We consider two mapping methods. The first one is the direct mapping scheme of the sector cache in which each cache block is mapped onto a single location (page and block) in the CDC-DRAM. This method is simple and requires only one tag for each page. However, a potential drawback for this scheme is that the CDC-DRAM storage may not be efficiently used. The selection bit array S is not needed for this method.

The second one is to use two different mapping schemes for the data and for the tag, as used in the decoupled sector cache [35]. We still use direct mapping for the data part because a set associative mapping would require accessing more than one CDC-DRAM page simultaneously. However, the mapping for the tag part is set associative. Each CDC-DRAM page is associated with K tags and blocks from up to K pages in the physical memory can be cached in the same page. For each block, $\log K$ selection bits determine the tag that the block is associated with. In the default configuration, K is set to 4. In this way, we reduce the chance of page thrashing when two pages conflict in the CDC. For the details of the decoupled sector cache, interested readers may refer to [35]. This decoupled mapping is used in the default configuration.

Although the decoupled cache structure is complex, the major complexity lies in the on-chip tag cache. The only changes in the off-chip CDC are the additional tags and the selection bit array, which do not change the manufacturing process of the underlying cached DRAM. Because the tag cache is very small compared to the L2 cache, the tag cache access is faster than an L2 cache access, and does not affect the CDC access times. In the default configuration, the tag cache consists of only 32 entries with 184 bits in each entry: 4×22 for four page address tags, 32×2 for selection bits, and 32 valid bits. Each page address tag has 22 bits for 64MB CDC (48 bits in physical address minus 26 bits in page index and offset). The total storage space needs 776 bytes. For each CDC-DRAM page, the whole page tag occupies less than 1 percent of storage.

The CDC-cache mapping decides how the CDC-DRAM pages are mapped to the cache blocks in the CDC-cache. It is actually determined by the organization of the underlying cached DRAM. Recent studies have shown that cached DRAMs integrated with set-associative caches have significant lower cache miss rates than those with the direct mapped ones when used as main memories [44], [23], [50]. Thus, we only consider set-associative CDC-cache in this study. It is 4-way set-associative in the default configuration.

2.4 Predictor Design

Cache hit/miss prediction has been used for a variety of purposes. For example, it can improve load instruction scheduling and prefetching [48], [32]. In order to reduce the memory bandwidth requirement, the authors in [41] use miss prediction to dynamically mark which load instruction is cacheable/nonallocatable. We want to apply the prediction technique to predict CDC-DRAM hit/miss for CDC-cache misses so that accesses to the main memory will not be delayed when the CDC-DRAM miss is correctly predicted.

We adopt a two-level adaptive predictor using a global history register and a global pattern history table (GAg) [47]. The original predictor design is used for dynamic branch predictions. It has a two-level structure. The first level is a branch history register which records the current branch pattern. The second level is a pattern history table that uses saturating up-down counters to record the branch history for that pattern. When a branch is encountered, the saturating counter indexed by the current pattern is used to predict whether the branch should be taken or not. The real branch behavior will feed back to the history register and the counter.

We adapted the prediction mechanism such that the miss pattern instead of the branch pattern is used to train the predictor. The predictor observes memory references that have missed from the L2 cache and the CDC-cache and records the CDC hit/miss history for the most recent references in a global history register. The predictor is simple with a low implementation cost. For example, a predictor with an 8-bit history register, a 256-entry pattern history table, and 2-bit saturating counters requires only 520-bit storage. The additional cost to implement the logic is also small. The predictor is fast enough that the prediction will finish sooner than the L2 cache access.

There may be other predictor design alternatives. However, we have found that this simple scheme is very effective at predicting the CDC-DRAM hit/miss, as will be shown in Section 4. For this reason, we have not explored other alternatives.

2.5 Write Policy

The CDC can be organized as a write-through cache or a writeback cache. With the write-through policy, a replaced dirty L2 block will be written into the CDC and main memory at the same time. The write-through policy is simple to implement for the CDC, but it increases the traffic to the main memory. The writeback policy is more complicated. When a CDC-cache miss is encountered, the CDC-DRAM must be accessed for correctness no matter whether the access is predicated to be a hit or not (the predictor still works to filter out unnecessary accesses to the main memory). Furthermore, replacing a CDC-DRAM page with multiple dirty blocks will cause a burst of writebacks to the main memory. Additional buffers may be needed to hold those dirty blocks.

We believe that the write policy has little influence on the CDC performance for the following reasons: Writeback L2 cache has been extensively used to reduce the memory traffic. Furthermore, a recent study shows that eager writeback [26] can minimize the effect of write traffic on the overall system performance. This technique may also be applied to writeback CDC to avoid the bursts of writebacks. For simplicity, we only consider write-through CDC in this study.

The inclusion property [3] is enforced between the CDC-cache and CDC-DRAM at the page level: Every CDC-cache block is of the page size and may only cache the content of a CDC-DRAM page. However, the inclusion is not enforced between the CDC-DRAM and L2 cache. Because the L2 cache is set associative but the CDC-DRAM has a decoupled sector cache organization (it is similar to a direct mapped cache when only one memory page is mapped to each CDC-DRAM page), enforcing the inclusion property may reduce the effective associativity of the L2 cache and cause additional overhead. In general, not enforcing inclusion complicates cache coherence implementation. We will discuss cache coherence next.

2.6 Cache Coherence in Multiprocessor Environment

The CDC design can be used in multiprocessor systems with large main memories as well, in which cache coherence must be maintained. Because of the complexity, we only discuss this issue for bus-based shared-memory symmetric multiprocessors (SMPs) using write-back L2 cache and the MESI write invalidation protocol [7]. We assume each processor has one write-through CDC. In the MESI protocol, every L2 block is in one of four states: modified, exclusive, shared, and invalid. The local cache controller may initialize and respond to three types of bus transactions: bus read, bus read exclusive, and bus writeback. For every read or read exclusive request put on the bus by a remote processor, a local cache controller checks its local cache and, if it finds a cached copy of the requested block, the controller may generate response or transit the state of the block or do both. The invalidation is performed at the arrivals of read exclusive requests. The local controller does not respond to a bus writeback unless the writeback is a response to an earlier request sent by the controller.

The CDC may cause complication when the local controller receives a read request and there is no copy for the requested data in L2 cache. The CDC may or may not have a copy for that data and, thus, the CDC tag has to be read and matched with the incoming address. The CDC does not have to supply the data because the main memory has a valid copy. However, if the CDC does have a copy, the local controller must inform the requesting processor to put the remote cache block in shared state (by asserting a "shared" signal in the bus, for example) so that it will receive an invalidation if the remote copy is changed. Otherwise, the remote block can be put in exclusive state. When a local controller receives a read exclusive request, it must invalidate both copies in L2 cache and in CDC if they are found there. If the data is in CDC-DRAM and is not cached in CDC-cache, it is possible that the CDC checking is not finished when the data is returned from the memory. Because CDC-DRAM is at least as fast as the main memory, such a case is rare. When it does happen, the local controller may delay the completion of the transaction, for example, by asserting an inhibit line in the bus (this may also happen for L2 cache checking). Alternatively, the local controller can assume the data is not cached in CDC. If, later, it turns out the CDC does have a copy, the controller can invalidate the CDC copy (in case the remote processor would change the remote copy without notification). After all, there is no change in cache coherence implementation related to the L2 cache and the system bus.

CDC accesses may be interfered with by cache coherence traffic. Because only CDC tags are needed for maintaining cache coherence, a dedicated tag cache can be added to reduce the interferences. When a CDC-DRAM page is read for cache coherence, the tag portion is sent to this secondary tag cache and may be reused there. There is no need to keep the data part. Accesses from either the system bus or the local CPU will be matched with both tag caches. The whole page may be cached as the last entry in the CDC-cache LRU list or in a single-entry dedicate buffer, so it will be replaced quickly. The additional tag cache also speeds up cache coherence processing and reduces potential delay in responding to incoming requests.

3 Simulation and Experimental Setup

The simulation parameters are listed in Table 1. We use SimpleScalar 3.0 [5] to simulate a 4-way issue, 2GHz processor, and use SPEC CPU2000 benchmark [39] as the workload. We add the simulations of the CDC, the MSHR (miss information/status holding register), the writeback buffer, and the DRAM main memory system to the original simulator.

The latency of the L3 SRAM cache is estimated as follows: We assume that the L3 SRAM cache is on a separate die but in the same module with the CPU, using multichip module technology. The cache is connected to the CPU through a 1GHz, 32-byte wide internal bus (this is not as aggressive as the IA-64 Itanium processor cartridge, which has a 16-byte internal bus operating at the full CPU speed [33]). The access latency is 9.12ns, calculated by the Cacti model [37] (version 3.0, using $0.13\mu m$ technology). Synchronizing to the bus cycle and adding two bus cycles for bus delay, the total latency for transferring the first trunk is 24 CPU cycles. Adding three more bus cycles for transferring the following trunks, the total latency to fetch a 128-byte L2 block from L3 cache is 30 CPU cycles. The L1 cache latency calculated by the Cacti model is slightly higher than 1ns. We assume that an optimized cache implementation can reduce the access time to 1ns.

TABLE 1
Simulation Parameters

Processor speed	2GHz, 4-way issue
RUU (register update unit)	64-entry
LSQ (load/store queue)	32-entry
L1 inst. cache	32KB, 4-way associative 64-byte block, 2 cycle latency
L1 data cache	32KB, 4-way associative 64-byte block, 2 cycle latency
Unified L2 cache	1MB, 8-way associative 128-byte block, 8 cycle latency
MSHR	32 entries
Writeback buffer	16 entries
Memory bus bandwidth	6.4 GB/s
DRAM latency (excluding data transfer time)	50ns (100 cycles)
Off-chip cache bandwidth	25.6GB/s
L3 SRAM cache	8MB, direct mapped, 128-byte block, 24 cycle latency
CDC-cache	32 block, 4-way associative, 20 cycle latency for the first chunk of data
CDC-DRAM	32MB/64MB/128MB, 4KB page size, 16 banks, 20ns (40 cycles) row access latency
CDC hit/miss predictor	GAg with 8-bit history register and 2-bit saturating counter

The latency of the CDC-cache is estimated as follows: Since the CDC-cache tag checking is decoupled from a CDC-cache data access, the block index is known before accessing the CDC-cache. Thus, the CDC-cache is accessed like a direct mapped cache. The access time is estimated as 4.67ns by the Cacti model. However, the layout of the CDC-cache may not be optimized as the Cacti model suggests. Nevertheless, the Enhanced DRAM product has achieved an on-chip SRAM access time of 10.6ns [11]. Thus, we set the CDC-cache access time as 8ns (16 CPU cycles). The total latency to fetch a 128-byte L2 block from the CDC-cache is 26 CPU cycles: 20 cycles for the first 32-byte data chunk (after adding the bus delay) plus 6 cycles for three additional trunks.

We simulate a DRAM main memory system that supports split bus transactions and can schedule reads prior to earlier writes. At the DRAM level, the close-page mode and auto-precharge are used and the bank conflict is negligible because of the large number of banks in today's DRAM. The memory parameters are based on an 800MHz 4-channel Direct Rambus DRAM system. Each channel has 1.6GB/s bandwidth and the total bandwidth is 6.4GB/s. The initial DRAM latency is the delay from the time when the processor sends the request to DRAM to the time when it receives the first bit of data. The delay includes 20ns row access time, 20ns column access time, and 10ns bus delay. The total delay to fetch a 128-byte block from the main memory is 70ns (140 processor cycles).

In the original SimpleScalar, virtual memory addresses are always used to access caches and memory, while, in most systems, physical addresses are used to access L2/L3 caches and the main memory with an OS support of virtual-physical address translation. SimpleScalar locates text, data, and stack segments to remote, disjoint virtual memory regions in the virtual memory space. In its PISA architecture, for example, text code starts at memory address 0x00400000, data starts at memory address 0x10000000, and stack starts at 0x7ffc0000 (address decrementing with stack allocation). A real OS would map those segments onto closer regions in physical memory. Thus, cache miss rates for a large, direct-mapped L2 or L3 cache (or even a cache of low associativity) may be artificially higher in SimpleScalar. We added an OS-like virtual-physical address translation that maps virtual memory pages onto continuous "physical" memory pages by the

TABLE 2
L3 SRAM Cache Hit Rates and CDC Hit Rates for the 11 SPEC 2000 Programs

Program	SRAM-8M	CDC-32M		CDC-64M		CDC-128M	
		Overall	Cache	Overall	Cache	Overall	Cache
176.gcc	86.0%	96.4%	88.1%	96.4%	88.1%	96.4%	88.1%
181.mcf	74.6%	99.8%	50.5%	99.8%	50.5%	99.8%	50.5%
300.twolf	99.6%	99.6%	34.9%	99.6%	34.9%	99.6%	34.9%
168.wupwise	13.8%	19.8%	18.8%	39.1%	37.3%	51.8%	49.5%
171.swim	0.3%	3.9%	3.4%	19.3%	18.2%	54.2%	51.9%
172.mgrid	7.7%	29.9%	28.2%	88.2%	84.4%	88.2%	84.4%
173.applu	8.4%	33.9%	28.4%	41.7%	35.9%	66.2%	59.6%
179.art	100.0%	100.0%	95.8%	100.0%	95.8%	100.0%	95.8%
183.equake	13.9%	66.2%	61.3%	93.9%	86.3%	93.9%	86.3%
187.facerec	43.4%	97.3%	93.0%	97.3%	93.0%	97.3%	93.0%
189.lucas	14.2%	37.5%	7.6%	62.9%	20.1%	89.2%	32.9%
Average	42.0%	62.2%	46.4%	76.2%	58.6%	85.1%	66.1%

For CDCs, the "Overall" columns indicate the overall CDC hit rate, defined as the ratio between CDC hits and L2 misses. The "Cache" columns indicate the CDC-cache hit rate, defined as the ratio between CDC-cache hits and L2 misses. The CDC-cache hit rate is always lower than or equal to the CDC hit rate.

allocation order of those virtual memory pages. In our simulation, only L1 instruction and data caches are accessed by virtual addresses and L2 and L3 caches are accessed by physical addresses. (SimpleScalar simulates TLBs and can report TLB miss rates, but the TLBs do not really translate virtual addresses.)

In our study, we use the precompiled SPEC CPU2000 binaries provided in [42]. We select 11 programs from the benchmark suite that have significant memory stall times. All programs are fast forwarded by four billion instructions and are executed for the next one billion instructions.

4 Performance Results

In this section, we compare the performance of the 8MB SRAM L3 cache with the 32MB, 64MB, and 128MB CDCs in terms of cache hit rates and overall performance (IPC) and measure the accuracy of the CDC hit/miss predictor. To study the combination of CDC and hardware prefetching, we also compare the performance of CDC and SRAM L3 cache when a stream buffer exists.

4.1 Comparison of Cache Hit Rates

Table 2 gives the hit rates of the 8MB SRAM L3 cache (SRAM-8M), the 32MB CDC (CDC-32M), the 64MB CDC (CDC-64M), and the 128MB CDC (CDC-128M). The size of the CDC-cache is 128KB (32 4-KB cache blocks). The *overall CDC hit rate* is the ratio between CDC hits and L2 misses and the *CDC-cache hit rate* is the ratio between CDC-cache hits and L2 misses. Since a CDC-cache hit must be a CDC-DRAM hit, the CDC-cache hit rate is always lower than or equal to the CDC hit rate. The SRAM cache tends to have either very high or very low hit rates. Its hit rates for two programs, 300.twolf and 179.art, are close to 100 percent. However, its hit rates for six other programs are less than 15 percent. The hit rate for 171.swim is close to zero. The average hit rate is only 42 percent. This indicates that the 8MB SRAM cache is not large enough to hold the working sets for many applications.

For most programs, the CDC-DRAM hit rates of the CDCs are much higher than the hit rates of the SRAM cache. This is not a surprise since the CDCs have much larger capacities than the SRAM cache. The average CDC-DRAM hit rates are 62.2 percent, 76.2 percent, and 85.1 percent for 32MB, 64MB, and 128MB CDC, respectively. These values are higher than that of the SRAM L3 cache. The CDC-cache hit rates of the 64MB CDC are close to the respective CDC-DRAM hit rates for eight programs, indicating that the CDC-cache can well exploit the spatial locality in the CDC hit streams.

For six programs, the increase of the CDC size results in dramatic increases of the CDC-DRAM and CDC-cache hit rates. For example, the CDC-DRAM hit rate for program 171.swim jumps from 1.7 percent to 19.3 percent and then to 54.3 percent as the CDC size doubles from 32MB to 64MB and then to 128MB.

4.2 Overall Performance Improvements

Fig. 4 compares the IPC (instructions per cycle) of the SRAM L3 cache, the CDCs, as well as a base system with no L3 cache (Base) for the 11 SPEC 2000 programs.

The CDC variants significantly outperform the SRAM cache for nine programs. The speedup of the 64MB CDC over the SRAM cache is up to 78 percent (183.equake) and the average speedup is 25 percent. The average speedup of the 32MB CDC and the 128MB CDC is 13 percent and 36 percent, respectively. As shown in Section 4.1, the CDCs can hold a larger portion of the working sets of those programs. The CDCs have positive speedups over the base system for all programs.

The SRAM cache has slightly better performance than the 64MB CDC for only two programs, 300.twolf and 179.art. As shown in Section 4.1, the working sets of these two programs can fit into the 8MB SRAM cache. In general, the SRAM cache is more suitable than CDCs for applications whose working

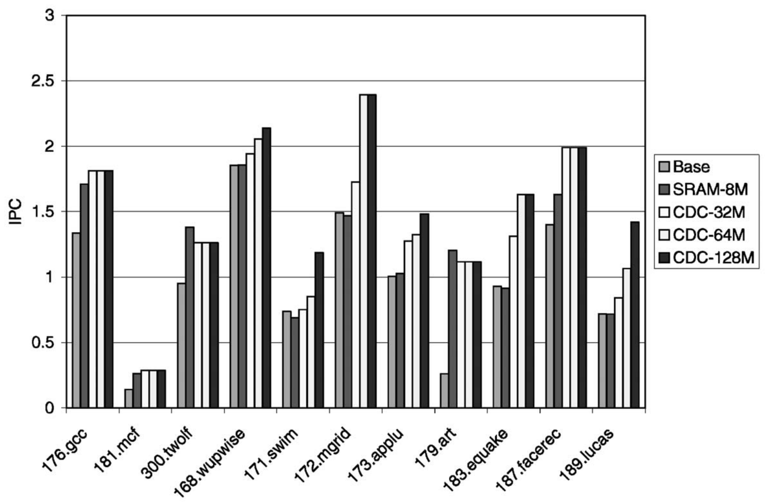


Fig. 4. IPC of the 8MB SRAM L3 cache (SRAM-8M), the 32MB CDC (CDC-32M), the 64MB CDC (CDC-64M), the 128MB CDC (CDC-128M), and a base system with no L3 cache (Base) for the 11 SPEC 2000 programs.

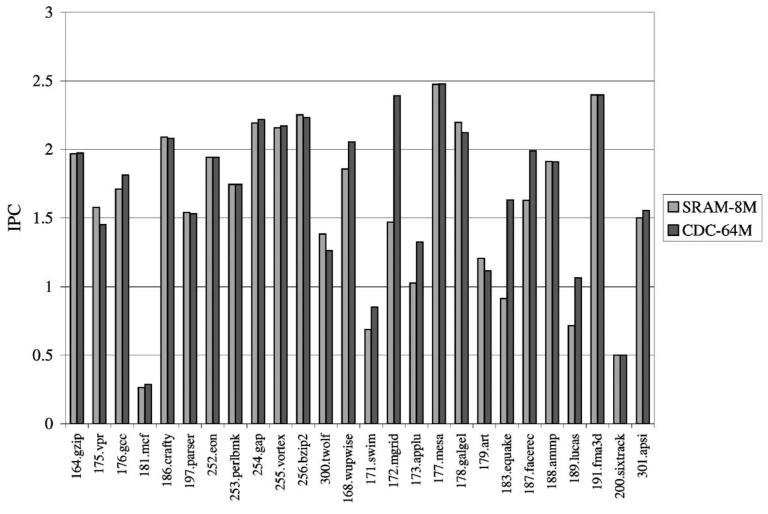


Fig. 5. The IPC comparisons of the CDC and SRAM L3 cache for all 26 SPEC2000 programs.

sets are larger than the on-chip cache sizes but smaller than the off-chip SRAM cache size. However, as shown in Fig. 4, many programs do not belong to this category. Compared with the base system, the SRAM cache has negative speedups for two programs, namely, 171.swim and 172.mgrid, because of the additional miss penalty from the L3 tag checking.

The overall performance of five programs (171.swim, 172.mgrid, 173.applu, 183.equake, and 189.lucas) is improved significantly as the CDC size increases. For example, the IPC of 171.swim increases by 13 percent and 40 percent as the CDC size doubles from 32MB to 64MB and from 64MB to 128MB.

4.3 Performance of Other SPEC 2000 Programs and SRAM L3 Variants

We present the IPC values of the SRAM L3 cache and the 64MB CDC for all the 26 SPEC2000 programs in Fig. 5. Further analysis shows that the CDC achieves high CDC-DRAM and CDC-cache hit rates for most of those applications. Because the other 15 programs are not memory-intensive, as indicated by the small performance differences between SRAM and CDC, the performance improvements for those programs are not significant. In other words, those programs do not demand large caches for high performance. Real-world applications are more memory-intensive than SPEC2000 and we believe a higher ratio of applications will demand large caches. Since the CDC is much larger than the SRAM cache, it will fit a larger range of applications and may achieve significant overall performance improvement when compared with the SRAM cache for real-world applications.

We also present the performance of set-associative SRAM L3 caches in Fig. 6, with the direct-mapped one and the 64MB CDC included for comparisons. The set-associative ones have visible but only slight improvement over the direct-mapped one on programs 176.gcc, 181.mcf, and 168.wupwise. They also cause small but negative impacts on programs 173.applu and 187.facerec. It is known that set-associative caches may cause negative speedups with imperfect replacement policies such as LRU [38], [45], which is used in our experiment. The effect is the most

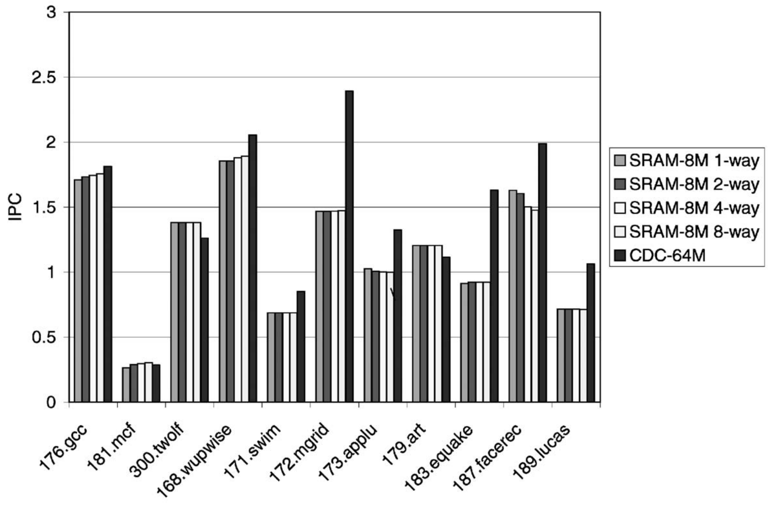


Fig. 6. The IPC comparison of direct-mapped, 2-way set-associative, 4-way set-associative, and 8-way set-associative SRAM L3 caches and the CDC for selected SPEC2000 programs.

visible on program 187.facerec. In general, set-associative SRAM L3 caches achieve lower miss rates than the direct-mapped one. However, they require longer access latency.

4.4 Predictor Performance

We use a GAg two-level adaptive predictor to predict CDC-DRAM hits when the data cannot be found in the CDC-cache. Table 3 gives the prediction accuracies (the number of correct predictions divided by the total number of predictions) of the CDC hit/miss predictor. The accuracies range from 81.2 percent to 100 percent and the average accuracies are 95.0 percent, 96.4 percent, and 97.2 percent with the 32MB, 64MB, and 128MB CDCs, respectively. For most programs, the prediction accuracies are consistent with CDCs of different sizes.

To distinguish the effects of using the predictor to exploit the CDC-DRAM locality, we compare the performance of the CDC with two other CDC variants, basic-CDC and perfect-CDC. With the basic-CDC, the data is always fetched from the main memory when a CDC-cache miss happens. With the perfect-CDC, it is magically known if the data can be found in the CDC-DRAM and the data is fetched from the main memory only when the data is not in the CDC-DRAM. The difference between the CDC and the perfect-CDC shows the effectiveness of the predictor. As discussed in Section 2, if the data can be found in CDC-DRAM but not in CDC-cache, it is still performance-beneficial to access the CDC-DRAM instead of the main memory. This is especially true for bandwidth-bound applications. The difference between the perfect-CDC and the basic-CDC shows this effect. In addition to providing a larger capacity, the CDC-DRAM also allows the CDC-cache to have a large block size for exploiting spatial locality.

Fig. 7 compares the overall performance of the CDC variants with the same size of 64MB. The CDC outperforms the basic-CDC significantly for programs that have high CDC-DRAM hit rates but relatively low CDC-cache hit rates (see Table 2). The two programs in which the SRAM cache outperforms the CDC belong to this category because CDC-DRAM can hold a working set if the SRAM L3 cache can hold it. For example, *179.art* is a program that the SRAM cache achieves an almost 100 percent hit rate and the basic-CDC achieves a 95.8 percent CDC-cache hit rate. Because it is very memory-intensive and bandwidth-bound, the program runs 26 percent slower on the basic-CDC than on the SRAM-L3. With the CDC, those misses are directed to CDC-DRAM, making the program run only 7 percent slower than on the SRAM-L3. In addition, the CDC performs as well as the perfect-CDC.

TABLE 3
The Accuracies of the CDC/Hit/Miss Predictor with the 32MB, 64MB, and 128MB CDCs

Program	CDC-32M	CDC-64M	CDC-128M
176.gcc	97.0%	97.0%	97.0%
181.mcf	99.6%	99.6%	99.6%
300.twolf	99.8%	99.8%	99.8%
168.wupwise	83.2%	80.6%	80.0%
171.swim	98.1%	96.7%	96.6%
172.mgrid	95.4%	98.9%	98.9%
173.applu	96.3%	95.6%	94.5%
179.art	100.0%	100.0%	100.0%
183.equake	81.2%	96.6%	96.6%
187.facerec	99.1%	99.1%	99.1%
189.lucas	78.6%	82.6%	94.4%
Average	93.5%	95.1%	96.1%

4.5 Evaluating CDC with Stream Buffer

To study the combination of CDC and hardware prefetching, we have also investigated the performance of CDC with stream buffer [20], a simple hardware prefetching scheme. A stream buffer is usually put between the L1 and L2 caches or is used to replace the L2 cache. We incorporate a simple implementation of stream buffer between the L1 and L2 caches in our simulation. The stream buffer activates prefetching on sequential L1 miss addresses using FIFO structure [20]. It has an allocation filter similar to the one proposed in [29] in which a stream buffer is allocated only when two misses are found on two consecutive addresses. To

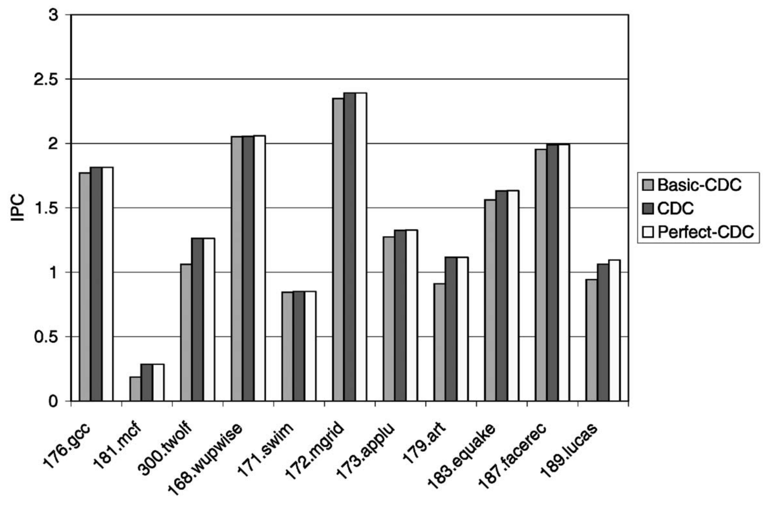


Fig. 7. The comparison of 64MB CDC variants.

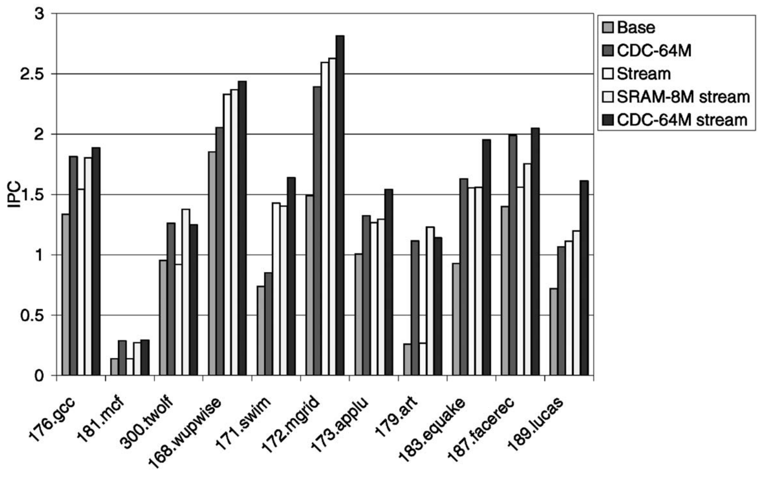


Fig. 8. The comparison of stream buffer only, the 8MB SRAM L3 with stream buffer, and the CDC with stream buffer.

prevent inaccurate prefetching requests from congesting the L2, L3, or memory bus, prefetching requests are sent out only when the DRAM and L3 buses are idle (similar to the configuration in [36]). To simplify our evaluation, we only present results of the stream buffer with eight ways and a depth of four, i.e., up to eight streams may be identified and up to four prefetching requests may be issued for each stream. The stream buffer uses LRU policy to reuse stream buffer ways and uses a round-robin policy to determine the order in sending out prefetching requests from multiple ways.

Fig. 8 shows the IPC of selected SPEC 2000 programs with five configurations: a conventional memory system without an L3 cache or a stream buffer (Base), a 64MB CDC without a stream buffer (CDC-64M), a stream buffer without L3 cache (Stream), the 8MB SRAM L3 cache with a stream buffer (SRAM-8M stream), and the CDC with a stream buffer (CDC-64M stream), respectively. The first two results are given for comparisons. The stream buffer improves the performance for most programs (comparing Base with Stream), but programs 181.mcf, 300.twolf, and 179.art need large caches for significant performance improvement. The program 300.twolf has a negative speedup, which is caused by inaccurate prefetchings. With the stream buffer present, the CDC has significantly better performance than the SRAM L3 cache. Compared with the SRAM L3 cache, the average speedup of using stream buffer with the SRAM L3 cache is 36 percent, while the average speedup of using the stream buffer with CDC is 53 percent.

Fig. 8 also shows that a subset of the selected SPEC 2000 applications favors the use of a large cache, while another subset favors the use of a stream buffer. Using *Base* (without a stream buffer or an off-chip cache) as the reference point, the stream buffer improves the performance of six programs by more than 10 percent, while the CDC improves performance on eight programs by more than 10 percent. When the two approaches are compared, the stream buffer loses to CDC for five applications (176.gcc, 181.mcf, 300.twolf, 179.art, and 187.facerec), beats CDC for two programs (168.wupwise, 171.swim), and is comparable for the other four programs (within 10 percent). Most importantly, their performance improvements over Base can add up, with the best evidence from the results of 173.applu, 183.equake, and 189.lucas.

We believe both approaches should be used, especially for mixed applications. The above results show that the stream buffer and the CDC are complementary to each other. Both the CDC-cache and stream buffer exploit spatial locality in memory references, but in different ways. Upon an L2 cache miss, the CDC-cache in our configuration will fetch 32 128-byte blocks surrounding the demanded block. In other words, the CDC-cache exploits the spatial locality existing in large regions (4KB in our configuration) and is effective when data blocks in those regions have been buffered into the CDC-DRAM. The stream buffer works when cache misses happen on sequential or stride memory addresses. It exploits spatial locality only in those particular forms, but requires a small amount of data storage.

5 Related Work

Cached DRAM is an existing technology that exploits the locality in DRAM access streams, taking advantage of the huge internal bandwidth inside the DRAM. Regarding the cached DRAM design and evaluation, interested readers may refer to [15], [14], [16], [23], [44], [50]. Enhanced DRAM and Virtual Channel SDRAM are commercial products in this category. Some recent studies show that the spatial locality in cache miss streams can be well exploited on the DRAM side [8], [49], [27]. In an early study of using cached DRAM as secondary cache [19], the cache block contains only 16 4-bit words. With such a small block size, the SRAM cache in the cached DRAM cannot exploit the spatial locality in the cache miss streams well.

Conventionally, cached DRAM is used as the main memory and connected to the processor through a low bandwidth memory bus. The long bus delay makes the cached DRAM less effective in reducing the overall memory access latency. In our study, the CDC is a special cached DRAM used as another level of cache on the processor side. Its performance potential is not limited by the memory bus. In addition, it can reduce the memory bus traffic.

IRAM (Intelligent RAM) [30] combines processor and DRAM onto the same chip. Although IRAM has the advantages of low latency DRAM accesses and high energy efficiency, it causes a slowdown of the processor speed due to the change of fabrication. Saulsbury et al. [34] study a processor/memory integration that uses a small size, large block SRAM cache with an on-chip DRAM. The on-chip DRAM is used as the main memory and the cache structure is similar to that of cached DRAM. Wilson and Olukotun find that the performance of an on-chip (embedded) DRAM cache is comparable to that of an on-chip SRAM cache [43]. The on-chip DRAM has also been studied for chip multiprocessors [21], [46]. In recent years, studies on VIRAM have shown that vector processing can be combined into IRAM to achieve significant speedup for SIMD applications whose performance is limited by memory system [24], [25], [12]. The DIVA processor [10] incorporates data processing ability into DRAM on a single chip. Micron Technology Inc. has produced a prototype chip called Yukon [22] that integrates DRAM and an array of small processing elements. In general, processors with on-chip DRAM are targeted for embedded applications that highlight the considerations of power and cost rather than performance. In our study, the CDC is used with high-performance processors that have large on-chip caches.

The approach of using a conventional DRAM as an off-chip L3 cache has been used in high-performance graphic systems [13], [6] and in hardware compressed main memory [1], [40]. In high-performance graphic systems, the bus bandwidth is the major performance bottleneck. By using a large DRAM cache, data can be kept in the local memory of the graphics system. The performance is insensitive to the DRAM speed. In the hardware compressed main memory, the DRAM L3 cache buffers uncompressed data so that the processor can directly use the data. For those applications, the DRAM has the same access latency as the conventional DRAM. In contrast, the CDC has a shorter access latency than the conventional DRAM.

Latency tolerant techniques such as speculative execution and prefetching [2], [4], [28], [27] are different approaches to reducing the memory stall time. The CDC technique not only reduces the memory latency, but may also decrease the memory traffic.

Recently, the IBM Power4 processor used DRAM as an off-chip L3 cache. The Power4 [18] has dual processor cores on a single chip, 128KB instruction cache, 64KB data cache, about 1.5MB unified L2 cache, and a 32MB off-chip L3 cache constructed by eDRAM (embedded DRAM). The L3 cache is 8-way set-associative organized in 512-byte blocks, with coherence maintained on 128-byte sectors for compatibility with the L2 cache. (The L2 cache is 8-way set-associative organized in 128-byte blocks.) Although the L3 data array is stored off-chip, the L3 tag array and cache controller are on the processor chip. Each L3 cache block requires 20 bits for storing its address tag (for the standalone configuration of the L3 cache) and each of the four sectors requires 3 bits for maintaining cache coherence. Thus, each block requires 32 bits for address tag and coherence bits. The tag array for the 32MB L3 cache (64K 512-byte blocks) requires 256KB SRAM storage on the processor chip. In our design, the tag array is stored off-chip with the data array in DRAM and a tag cache is used to facilitate tag comparisons. The on-chip tag cache is very small. In contrast, the on-chip tag array in the Power4 design is large and must increase proportionately with the L3 cache. In addition, a large L3 tag array requires a relatively long tag comparison time, increasing the miss penalty for misses to main memory.

The approach of attaching a fast, small device to a slow device of a similar media has also appeared in operating systems and I/O areas. For example, the DCD (Disk Caching Disk [17]) uses a small disk region to buffer the on-the-fly data. In general, there exists a high-bandwidth connection between the fast device and slow device which can be used for transferring large chunks of data from the slow device to the fast device.

6 Conclusion

We have presented the design and performance evaluation by constructing CDC (cached DRAM cache) as the L3 off-chip cache, which is intended for high-end server systems. We show that the CDC effectively addresses two major concerns of the SRAM off-chip cache: the relatively small size and the miss overhead. The CDC is close to a DRAM cache in terms of capacity and close to an SRAM cache in terms of speed. More applications can benefit from the CDC than from the SRAM off-chip cache and no applications will lose performance due to the existence of CDC.

Contemporary computers tend to have large on-chip caches and gigabytes of main memories. An even larger off-chip cache will help memory-intensive applications. However, both access speed and storage size gaps between the on-chip caches and the main memory continue to widen. We believe that the low density and the high cost of SRAM will eventually limit its usage as off-chip caches. The CDC plays a unique role in addressing the speed and size gaps. Because the CDC can be much larger than the on-chip caches, it balances the accesses between the on-chip caches and the main memory. In contrast to off-chip caches constructed by DRAM such as that used in the IBM Power4 processor, the CDC requires very little on-chip space, can avoid overheads for main memory accesses, and can better utilize the spatial locality in cache miss streams.

Additional research can be done based on the CDC design framework presented in this paper to further improve its performance. One approach is to put data with spatial locality into the CDC instead of the on-chip caches, improving the efficiency of the on-chip caches for data with only temporal locality (such as pointer variables) when the on-chip caches cannot hold data of both locality types. Another approach is to use aggressive prefetching techniques with CDC. Data can be prefetched from the CDC-DRAM to the CDC-cache or to the on-chip caches, utilizing the high bandwidth inside the CDC and the one between the CDC and the processor. As future work, we will investigate those approaches.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their insightful and constructive comments. They also appreciate Bill Bynum for reading the paper and making suggestions and corrections. The work has been supported in part by the US National Science Foundation under grants CCR-9812187, CCR-0098055, and ACI-0129883.

REFERENCES

[1] B. Abali, H. Franke, X. Shen, D.E. Poff, and T.B. Smith, "Performance of Hardware Compressed Main Memory," *Proc. Seventh Int'l Symp. High-Performance Computer Architecture,* pp. 73-81, 2001.

[2] M.M. Annavaram, J.M. Patel, and E.S. Davidson, "Data Prefetching by Dependence Graph Precomputation," *Proc. 28th Ann. Int'l Symp. Computer Architecture,* pp. 52-61, 2001.

[3] J.-L. Baer and W.-H. Wang, "On the Inclusion Properties for Multi-Level Cache Hierarchies," *Proc. 15th Ann. Int'l Symp. Computer Architecture,* pp. 73-80, 1988.

[4] R. Balasubramonian, S. Dwarkadas, and D.H. Albonesi, "Dynamically Allocating Processor Resources between Nearby and Distant ILP," *Proc. 28th Ann. Int'l Symp. Computer Architecture,* pp. 26-37, 2001.

[5] D. Burger, "System-Level Implications of Processor-Memory Integration," Technical Report CS-TR-1997-1349, Univ. of Wisconsin, Madison, June 1997.

[6] M. Cox, N. Bhandari, and M. Shantz, "Multi-Level Texture Caching for 3D Graphics Hardware," *Proc. 25th Ann. Int'l Symp. Computer Architecture,* pp. 86-97, 1998.

[7] D. Culler, J.P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach.* San Mateo, Calif.: Morgan Kaufmann, 1999.

[8] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A Performance Comparison of Contemporary DRAM Architectures," *Proc. 26th Ann. Int'l Symp. Computer Architecture,* pp. 222-233, 1999.

[9] Z. Cvetanovic and D.D. Donaldson, "AlphaServer 4100 Performance Characterization," *Digital Technical J.,* vol. 8, no. 4, pp. 3-20, 1996.

[10] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C.W. Kang, I. Kim, and G. Daglikoca, "The Architecture of the DIVA Processing-in-Memory Chip," *Proc. 16th Int'l Conf. Supercomputing,* pp. 14-25, 2002.

[11] Enhanced Memory Systems Inc., *64 Mit ESDRAM Components, Product Brief r1.8,* 2000.

[12] B. Gaeke, P. Husbands, X. Li, L. Oliker, K. Yelick, and R. Biswas, "Memory-Intensive Benchmarks: IRAM vs. Cache-Based Machines," *Proc. 16th Int'l Parallel and Distributed Processing Symp.,* pp. 30-30, 2002.

[13] Z.S. Hakura and A. Gupta, "The Design and Analysis of a Cache Architecture for Texture Mapping," *Proc. 24th Ann. Int'l Symp. Computer Architecture,* pp. 108-120, 1997.

[14] C.A. Hart, "CDRAM in a Unified Memory Architecture," *Proc. CompCon '94,* pp. 261-266, 1994.

[15] H. Hidaka, Y. Matsuda, M. Asakura, and K. Fujishima, "The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory," *IEEE Micro,* vol. 10, no. 2, pp. 14-25, Apr. 1990.

[16] W.-C. Hsu and J.E. Smith, "Performance of Cached DRAM Organizations in Vector Supercomputers," *Proc. 20th Ann. Int'l Symp. Computer Architecture,* pp. 327-336, 1993.

[17] Y. Hu and Q. Yang, "DCD-Disk Caching Disk: A New Approach for Boosting I/O Performance," *Proc. 23rd Ann. Int'l Symp. Computer Architecture,* pp. 169-178, 1996.

[18] "POWER4 System Architecture,"white paper, IBM, Oct. 2001.

[19] F. Jones et al., "A New Era of Fast Dynamic RAMs," *IEEE Spectrum,* pp. 43-49, Oct. 1992.

[20] N.P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proc. 17th Ann. Int'l Symp. Computer Architecture,* pp. 364-373, 1990.

[21] P. Keltcher, S. Richardson, and S. Siu, "An Equal Area Comparison of Embedded DRAM and SRAM Memory Architectures for a Chip Multiprocessor," Technical Report HPL-2000-53, HP Laboratories, Palo Alto, Calif., Apr. 2000.

[22] G. Kirsch, "Active Memory: Micron's Yukon," *Proc. Int'l Parallel and Distributd Processing Symp.,* p. 89b, 2003.

[23] R.P. Koganti and G. Kedem, "WCDRAM: A Fully Associative Integrated Cached-DRAM with Wide Cache Lines," *Proc. Fourth IEEE Workshop Architecture and Implementation of High Performance Comm. Systems,* 1997.

[24] C. Kozyrakis, "A Media-Enhanced Vector Architecture for Embedded Memory Systems," Technical Report CSD-99-1059, Univ. of California, Berkeley, 1999.

[25] C. Kozyrakis, J. Gebis, D. Martin, S. Williams, I. Mavroidis, S. Pope, D. Jones, and D. Patterson, "Vector IRAM: A Media-Enhanced Vector Processor with Embedded DRAM," *Proc. Hot Chips 12,* 2000.

[26] H.-H. Lee, G. Tyson, and M. Farrens, "Eager Writeback—A Technique for Improving Bandwidth Utilization," *Proc 33rd IEEE/ACM Int'l Symp. Microarchitecture,* pp. 11-21, 2000.

[27] W. Lin, S. Reinhardt, and D. Burger, "Reducing DRAM Latencies with an Integrated Memory Hierarchy Design," *Proc. Seventh Int'l Symp. High-Performance Computer Architecture,* pp. 301-312, 2001.

[28] C.-K. Luk, "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors," *Proc. 28th Ann. Int'l Symp. Computer Architecture,* pp. 40-51, 2001.

[29] S. Palacharla and R.E. Kessler, "Evaluating Stream Buffers as a Secondary Cache Replacement," *Proc. 21st Ann. Int'l Symp. Computer Architecture,* pp. 24-33, 1994.

[30] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A Case for Intelligent RAM," *IEEE Micro,* pp. 34-44, Mar./Apr. 1997.

[31] J.-K. Peir, W.W. Hsu, and A.J. Smith, "Functional Implementation Techniques for CPU Cache Memories," *IEEE Trans. Computers,* vol. 48, no. 2, pp. 100-110, Feb. 1999.

[32] J.-K. Peir, S.-C. Lai, S.-L. Lu, J. Stark, and K. Lai, "Bloom Filtering Cache Misses for Accurate Data Speculation and Prefetching," *Proc. 16th Int'l Conf. Supercomputing (ICS-02),* pp. 189-198, 2002.

[33] W.A. Samaras, N. Cherukuri, and S. Venkataraman, "The IA-64 Itanium Processor Cartridge," *IEEE Micro,* vol. 21, no. 1, pp. 82-89, Jan./Feb. 2001.

[34] A. Saulsbury, F. Pong, and A. Nowatzyk, "Missing the Memory Wall: The Case for Processor/Memory Integration," *Proc. 23rd Ann. Int'l Symp. Computer Architecure,* pp. 90-103, 1996.

[35] A. Seznec, "Decoupled Sectored Caches: Conciliating Low Tag Implementation Cost and Low Miss Ratio," *Proc. 21st Ann. Int'l Symp. Computer Architecture,* pp. 384-393, 1994.

[36] T. Sherwood and B. Calder, "A Decoupled Predictor-Directed Stream Prefetching Architecture," *IEEE Trans. Computers,* vol. 52, no. 5, Mar. 2003.

[37] P. Shivakumar and N.P. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power, and Area Model," technical report, COMPAQ Western Research Lab, Aug. 2001.

[38] J.E. Smith and J.R. Goodman, "A Study of Instruction Cache Organization and Replacement Policies," *Proc. 10th Ann. Int'l Symp. Computer Architecture,* pp. 132-137, 1983.

[39] Standard Performance Evaluation Corp., http://www.spec.org, 2004.

[40] R.B. Tremaine, T.B. Smith, M. Wazlowski, D. Har, K.-K. Mak, and S. Arramreddy, "Pinnacle: IBM MXT in a Memory Controller Chip," *IEEE Micro,* vol. 21, no. 2, pp. 56-68, Mar/Apr. 2001.

[41] G. Tyson, M. Farrens, J. Matthews, and A.R. Pleszkun, "A Modified Approach to Data Cache Management," *Proc. 28th Ann. Int'l Symp. Microarchitecture,* pp. 93-103, 1995.

[42] C. Weaver http://www.simplescalar.org/spec2000.html, SPEC2000 binaries, 2004.

[43] K.M. Wilson and K. Olukotun, "Designing High Bandwidth On-Chip Caches," *Proc. 24th Ann. Int'l Symp. Computer Architecture,* pp. 121-132, 1997.

[44] W. Wong and J.-L. Baer, "DRAM On-Chip Caching," Technical Report UW CSE 97-03-04, Univ. of Washington, Feb. 1997.

[45] W.A. Wong and J.-L. Baer, "Modified LRU Policies for Improving Second-Level Cache Behavior," *Proc. Sixth Int'l Symp. High-Performance Computer Architecture,* pp. 49-60, 2000.

[46] T. Yamauchi, L. Hammond, and K. Olukotun, "A Single Chip Multiprocessor Integrated with High Density DRAM," Technical Report CSL-TR-97-731, Computer Systems Laboratory, Stanford Univ., Aug. 1997.

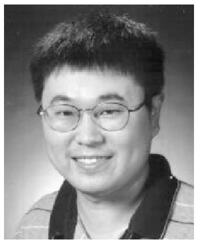
[47] T.-Y. Yeh and Y.N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *Proc. 19th Ann. Int'l Symp. Computer Architecture,* pp. 124-134, 1992.

[48] A. Yoaz, M. Erez, R. Ronen, and S. Jourdan, "Speculation Techniques for Improving Load Related Instruction Scheduling," *Proc. 26th Ann. Int'l Symp. Computer Architecture,* pp. 42-53, 1999.

[49] Z. Zhang, Z. Zhu, and X. Zhang, "A Permutation-Based Page Interleaving Scheme to Reduce Row-Buffer Conflicts and Exploit Data Locality," *Proc. 33rd IEEE/ACM Int'l Symp. Microarchitecture,* pp. 32-41, 2000.

[50] Z. Zhang, Z. Zhu, and X. Zhang, "Cached DRAM: A Simple and Effective Technique for Memory Access Latency Reduction on ILP Processors," *IEEE Micro,* vol. 21, no. 4, pp. 22-32, July/Aug. 2001.

[51] Z. Zhu, Z. Zhang, and X. Zhang, "Fine-Grain Priority Scheduling on Multi-Channel Memory Systems," *Proc. Eighth Int'l Symp. High-Performance Computer Architecture,* pp. 107-116, 2002.



Zhao Zhang received the BS degree and MS degrees in computer science from Huazhong University of Science of Technology, China, in 1991 and 1994, respectively, and the PhD degree in computer science from the College of William and Mary in 2002. He is an assistant professor of computer engineering at Iowa State University. His research interests include computer architecture, and parallel and distributed systems. He is a member of the IEEE and ACM.

Zhichun Zhu received the BS degree in computer science from Huazhong University of Science of Technology, China, in 1992, and the PhD degree in computer science from the College of William and Mary in 2003. She is an assistant professor of electrical and computer engineering at the University of Illinois at Chicago. Her research interests include computer architecture, performance evaluation, and low-power designs. She is a member of the IEEE and ACM.

Xiaodong Zhang received the BS degree in electrical engineering from Beijing Polytechnic University in 1982 and the MS and PhD degrees in computer science from the University of Colorado at Boulder in 1985 and 1989, respectively. He is the Lettie Pate Evans Professor of Computer Science and the Department Chair at the College of William and Mary. He was the program director of Advanced Computational Research at the US National Science Foundation from 2001 to 2003. He is a past editorial board member of the *IEEE Transactions on Parallel and Distributed Systems* and currently serves as an associate editor of *IEEE Micro*. His research interests are in the areas of parallel and distributed computing and systems and computer architecture. He is a senior member of the IEEE.