Exploring DRAM Cache Architectures for CMP Server Platforms

Li Zhao, Ravi Iyer, Ramesh Illikkal and Don Newell
System Technology Lab, Intel Corporation, Hillsboro, OR
Email: {li.zhao, ravishankar.iyer, ramesh.g.illikkal, donald.newell}@intel.com

Abstract

As dual-core and quad-core processors arrive in the marketplace, the momentum behind CMP architectures continues to grow strong. As more and more cores/threads are placed on-die, the pressure on the memory subsystem is rapidly increasing. To address this issue, we explore DRAM cache architectures for CMP platforms. In this paper, we investigate the impact of introducing a low latency, large capacity and high bandwidth DRAM-based cache between the last level SRAM cache and memory subsystem. We first show the potential benefits of large DRAM caches for key commercial server workloads. As the primary hurdle to achieving these benefits with DRAM caches is the tag space overheads associated with them, we identify the most efficient DRAM cache organization and investigate various options. Our results show that the combination of 8-bit partial tags and 2-way sectoring achieves the highest performance (20% to 70%) with the lowest tag space (<25%) overhead.

1 Introduction

The momentum behind CMP architectures [8] is pushing architects and designers to consider integrating more and more cores on the die. Within this decade, we expect that moderate to large-scale single-socket CMP platforms with 8 to 32 cores will be commonplace. Intel's Tera-scale research initiative [9] and Sun's Niagara 2 [6] are excellent examples of this trend. Such platforms are very suitable for throughput computing [4] and therefore are a highly attractive option for commercial servers.

As more cores are integrated on-die, the performance and scalability of CMP architectures are even more heavily dependent on the memory subsystem. It is well understood that the memory latency and bandwidth is improving at a much slower pace than that of CPU. One approach to solving this problem is to develop memory technologies such as fully buffered DIMMs (FBD) [5][12] that reduce pin count and offer more memory channels. Another approach is to add more cache space. Cached DRAM [20] integrates an SRAM cache in the DRAM memory to exploit the locality in memory accesses and thus reduces the miss penalty. Another potential solution is to add an off-chip DRAM cache in between the last-level processor cache and the DRAM memory. Using DRAM allows for a much larger capacity but at a somewhat longer latency. Previous studies [10][21] and our own internal analysis has shown that DRAM-based caches (using Multi-Chip Package or 3D-stacking technology) can be designed to provide twice the bandwidth of main memory at about one third of the memory latency. Our goal is to take advantage of this potential for CMP-based server platforms. To our knowledge, this is the first study that explores the potential of DRAM cache and evaluates the architectural options for CMP-based server platforms in detail.

To determine the potential of large DRAM caches, the first step is to evaluate the miss rate benefits that it provides. We start by showing this potential for five different server workload scenarios. The first four scenarios are based on four server applications (TPC-C, SPECjappserver, SPECjbb and SAP SD/2T) running individually on an 8-core single-socket server. The final scenario is based on a virtualized environment running all four of the above server workloads simultaneously on a 32-core single-socket server platform. Since server consolidation [19] is a growing usage model, we felt that large-scale CMP server platforms are best evaluated using these virtualized scenarios. We show the miss rate benefits of large DRAM caches can translate into significant reduction in terms of average memory latency and consumed memory bandwidth.

The next step is to determine a performance and area efficient design option. A primary hurdle to determining a DRAM cache option is the large tag space overhead that needs to be accommodated. Placing the tags in the off-die DRAM cache implies that the tags need to be accessed before the data and therefore requires almost twice the latency. On the other hand, placing the tags on the die implies that it displaces the last-level cache to stay within the same area. To understand the implications of tag space overhead and identify the most suitable DRAM cache implementation, we study five DRAM cache architecture options: 1) DRAM cache with off-die tag, 2) DRAM cache with full on-die tags, 3) sectored DRAM cache with on-die tag, 4) DRAM

1-4244-1258-7/07/$25.00 ©2007 IEEE

cache with partial on-die tags and 5) sectored DRAM cache with partial on-die tags. We show that the use of sectoring and partial tags can save on-die tag space but have different trade-offs in terms of performance We also show that combination of sectoring and partial tags provides the maximum performance gain at minimized tag space overhead. Compared to previous studies [21], we perform a detailed evaluation of various design options for a DRAM cache, especially the design tradeoffs to reducing on-die tag space.

The rest of this paper is organized as follows. Section 2 motivates the need for DRAM caches and discusses the considerations and trade-offs. In Section 3, we present five DRAM cache design options and discuss their implications on implementation. Section 4 evaluates the DRAM cache design options based on the five workload scenarios mentioned above. Section 5 concludes the paper with a direction for future work.

2 Memory Overheads and DRAM Cache

In this section, we discuss memory-related overheads in server workloads and motivate DRAM caches for CMP server platforms. We also show the potential benefits of the DRAM cache along with the considerations and trade-offs.

2.1 Memory Overheads in CMP Platforms

The memory wall problem [11] has been around for a long time. This problem is exacerbated in CMP platforms as more hardware cores or threads are actively executing with a relatively smaller amount of cache space available per thread. We collected data on memory stall times when running key commercial server workloads in a quad-core CMP platform. It shows that memory overheads are significant, which varies from 45% to 70% depending on the workload. This motivates the need for addressing memory stall time through better memory hierarchies or better memory technologies.

2.2 DRAM Caches for CMP Servers

Researchers have studied (and in some cases even implemented) large SRAM caches [13][18][20] and DRAM caches [10][21] in the past. The context has either been to introduce external large capacity caches in node controllers or chipsets in single-core processors. Recently, however, the potential of Multi-Chip Package and 3D stacking [3] has opened up the possibility of embedding the DRAM cache within the package. In this paper, our focus is to evaluate the performance potential of a DRAM cache and the possible design options when integrating the DRAM cache into the package of a CMP-based server processor.

2.2.1 Cache Size for BW/Latency Mitigation

Figure 1 illustrates a typical cache/memory hierarchy and the potential of introducing a DRAM cache layer between the last-level cache and main memory. To highlight the potential of DRAM caches, we studied DRAM miss rates for realistic server workloads. Using long hardware bus traces of well known server benchmarks (TPC-C, SPECjbb2005, SAP SD/2T and SPECjappserver), we conducted simulations with 8 cores, a shared 8M L3 cache and a 64M DRAM (L4) cache. We also studied a virtualized scenario where all four workloads were run simultaneously to mimic consolidation on a 32-core CMP server with a shared 8M L3 and a 64M DRAM (L4) cache. More simulation details are provided later in Section 4.

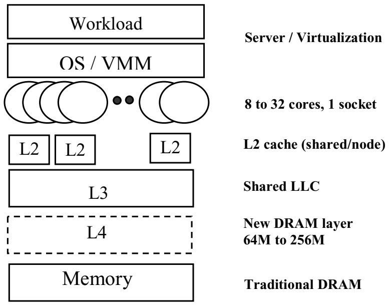

Figure 1. Introducing a DRAM cache in a typical cache/memory hierarchy

Figure 2 shows the DRAM cache hit rate for the five server workloads. The hit rate benefits are quite significant, ranging from around 50% to 80%. These DRAM cache hits will have significantly lower latency since the DRAM cache access latency is expected to be much lower than the memory latency. Our internal analysis of embedding DRAM within the package (not provided in this paper due to proprietary reasons) indicates that the DRAM cache access latency can be lower than one-third of the DRAM memory latency in most server configurations (even with integrated memory controllers). In addition, the DRAM cache hit rate also points to a significant reduction in main memory bandwidth and thereby much lower queuing delays for main memory accesses. Figure 3 shows the estimated reduction in average memory latency as well as main memory bandwidth requirement. Note that our choice of a 64MB DRAM cache size was due to simulation limitations to ensure sufficient warm-up. However, the potential of DRAM caches is even more significant than shown because it is feasible to accommodate much larger DRAM caches (up to 256MB).

2.2.2 Key Consideration - Tag Overheads

As the cache become large, so does the tag space. A primary design consideration for DRAM caches is the tag size and its implications on location and organization. Assuming that the physical memory space is 16TB (44 bits) and DRAM cache is 16-way, the required tag space for 64 to 256MB caches are illustrated in Figure 4. Take an example

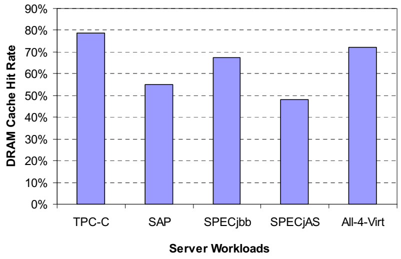

Figure 2. 64M DRAM cache hit rate (over 8M LLC)

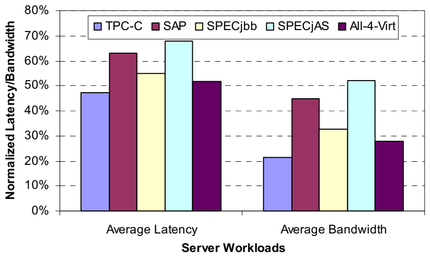

Figure 3. Potential memory latency/bandwidth benefits

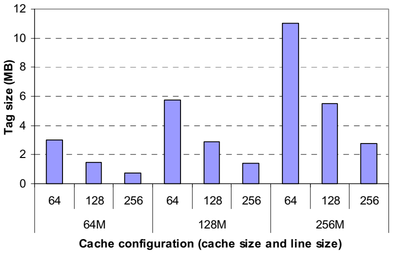

Figure 4. Tag space overhead for DRAM caches

of a 256MB cache with 64B lines. The tag space is about 11MB. The traditional approach is to place the tags along with the data. However, since the DRAM cache is off-die, any access to the DRAM cache will suffer the overhead of looking up the 16 tags and determining the hit/miss. The alternative is to place the tags on-die. However, this implies that 11MB of space is required on the main die. This either requires that the die area increases or that it replaces the last-level cache space. Both of these approaches have unacceptable impact on cost and performance respectively. Other alternatives include employing large line sizes or direct mapped caches. Figure 4 shows that increase in line size can reduce the tag space linearly. However, it also has a direct implication on miss rate and memory bandwidth. In the rest of this paper, we will present various design options and do an in-depth evaluation.

3 DRAM Cache Architecture Options

In this section, we introduce the DRAM architecture options, describe their operation and discuss the key considerations/trade-offs for each one.

3.1 DRAM Cache with Off-Die Tags

A simplistic option for DRAM caches (DRAM$) is to place tags off-die along with the data. In this case, the appropriate DRAM cache organization depends on its set associativity. For a cache with n-way set associativity ($n > 1$), it is desired that all tags for each set are placed together within one block (shown in Figure 5a). This allows the tags within a set to be read in one data transfer for hit/miss determination. If there is a hit found, the second request is sent to the DRAM data cache to retrieve the data. Otherwise the request is sent to memory. The miss latency can be reduced by sending the request to memory at the same time that it is sent to the DRAM cache. However, the ability to cancel the request if a hit is found in the DRAM cache is critical. If the data is fetched before the DRAM hit is detected, then the primary purpose of reducing memory bandwidth is defeated. In any case, this organization requires two block transfers (tags and data) for a hit, and one data transfer (tags) for a miss. To avoid multiple data transfers for hits, it is possible to build a direct-mapped DRAM cache, where the tag is placed along with the data block. This allows the tag and the data to be read at the same time. If the tag comparison results in a hit, the data is available immediately. However a direct-mapped cache usually has much higher miss rate than set-associativity cache. To quantify this, we compared the miss rates of direct-mapped to set-associative DRAM caches. Our results using a 64MB cache in an 8-core CMP show that 8 or 16-way caches provide a miss rate reduction of 20 to 40% over direct-mapped caches. In addition, as cores continue to increase on the die, we expect that this benefit will increase due to more conflict misses. As a result, we consider set-associative (8-way or 16-way) DRAM caches for this study.

3.2 DRAM Cache with On-Die Tags

As off-die tag placement incurs higher DRAM cache access latency and requires higher bandwidth, we can move all of the tags on-die (as shown in Figure 5b). When a memory request is sent to the last level cache (LLC), it is also sent to the on-die tags of the DRAM cache. If a miss occurs in last-level cache and a hit is found in DRAM tags, the request is sent to the DRAM cache for retrieving the data. Otherwise the request is sent to memory, and the data from memory is sent to DRAM cache and last-level cache simultaneously. Therefore, only one data transfer is required for DRAM cache hits. The drawback of this approach is that additional die area is required to accommodate the DRAM tags on-die. If die area increase cannot be allowed, which is the typical case, then the DRAM tags will need to be accommodated by reducing the size for last-level cache.

3.3 Sectored DRAM Cache with On-Die Tags

One way to reduce the on-die tag space is to use large cache line sizes. While the use of large line sizes can benefit from spatial locality, it also comes at the expense of consuming more memory traffic and a potential increase in cache access latency. For systems with multiple sockets, large line sizes also increases the possibility of false sharing.

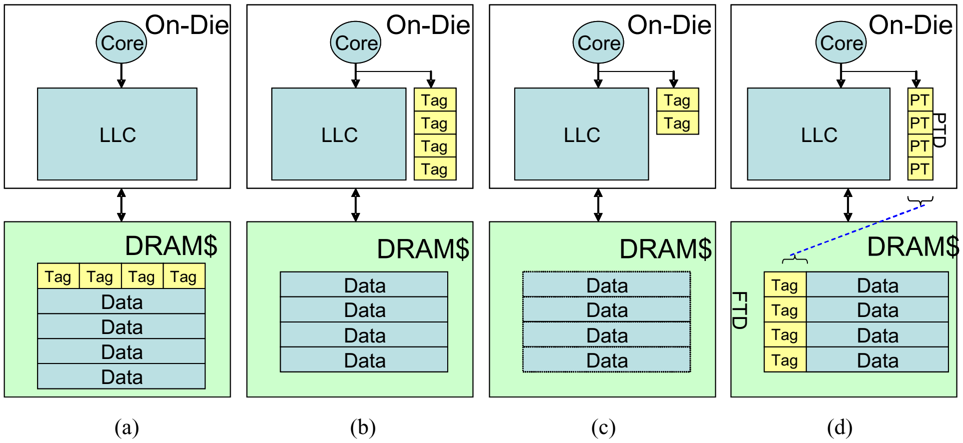

Figure 5. Potential DRAM cache design/organization options: (a) off-die tag, (b) on-die tag, (c) on-die tag sectored cache, (d) on-die partial tag, off-die full tag

A sectored cache [1] is a well-known alternative to building caches with large lines. A sectored cache essentially uses the concept of sub-blocks where each sub-block consists of a small cache line along with its state and the overall block is made up of these sub-blocks along with the block address. A sectored DRAM cache will support on-die tag space similar to a large-lined cache while not incurring as much traffic. Therefore we evaluate the performance effectiveness of a sectored DRAM cache, as shown in Figure 5c, and compare it to the other alternatives. Since sectored cache usually introduces a higher miss rate than non-sectored cache, it is straightforward to use sector prefetching, which prefetches neighboring sub-blocks when a miss occurs. Therefore, we also investigate the use of sector prefetching. Other perfeching techniques will be evaluated in future work.

3.4 DRAM Cache with On-Die Partial Tags

In order to reduce the on-die tag space and also maintain a low miss rate, we propose the use of on-die partial tag directories for DRAM caches. As shown in Figure 5d, the full tag directory (FTD) is placed off-die along with the data so that they can be read in one transfer. Each tag entry in the partial tag directory (PTD) contains the *m* least significant bits from the corresponding tag entry in FTD. When a L2 miss occurs, the request is sent to both the LLC (L3) and the PTD. If it results in an LLC miss and a PTD miss, the request is sent to memory. If it turns out to be PTD hit, the request is sent to the DRAM cache because it needs to check the full tag in the FTD to ensure that it is a true hit. If the full tag(s) comparison results in a match, then a true hit is determined and the data is returned to the core. If it results in a mismatch (i.e. a false PTD hit), the request has to be serviced in the main memory. It should be noted that a partial tag lookup (in the PTD) could result in multiple matches when a hit occurs. This happens when two lines have the same partial tag. Since multiple matches make the operation much more complex (requiring multiple full tags to be retrieved from the FTD), we only allow a single match by restricting the number of unique partial tags in the PTD to 1. This restriction is simply implemented in the replacement policy such that an allocation of a new line that maps to an existing partial tag in the set chooses the partial tag as the victim and replaces it. We have verified that this makes a negligible difference in the miss rate and therefore is an attractive approach.

The performance of partial tags depends on the number of bits chosen from the full tag. This number is chosen such that it can meet the following requirements: 1) it can provide a reasonable space reduction from the full tag; 2) there is a high probability that a true hit occurs. The probability of a true hit depends on the probability of finding a unique partial tag within a set. We statistically estimated the probability of finding a unique tag in an *n-way* set associative cache with *m* partial bits. For this analysis, we assumed that the tag values are randomly distributed within the set. This assumption is true as long as the workload has a uniformly random distribution of access across the main memory space (commercial server workloads like TPC-C exhibit this behavior). Since the primary requirement for a unique partial tag is that each of the *m* bits is different amongst the n ways in the set, the probability of this occurrence is

$$P_u = (2^m - 1)^{n-1}/(2^m)^{n-1} \quad (1)$$

Table 1 lists the probability as we increase *m* from 4-bit to 10-bit when n is set as 4 and 16 respectively. We can see that when *m=8*, the probability to find a unique partial tag is 94% for a 16-way cache, and can be as high as 98% for a 4-way cache. As expected, the probability increases as we increase *m*. Note that 8-bit partial tags provide almost a

60% space reduction for a 64MB cache and 16TB memory space (22-bit full tags).

Table 1. Prob. of unique partial tag in a set

PTD bits	Associativity	
M	N = 16	N = 4
4	0.380	0.824
5	0.621	0.909
6	0.790	0.954
7	0.890	0.977
8	0.943	0.988
9	0.971	0.994
10	0.985	0.997

To save more on-die space, another option is to implement the on-die partial tags for sectored DRAM caches. This is expected to combine the tag space reduction benefits of both approaches and minimize the negative effects of sectoring. We will evaluate the benefits of this hybrid approach in the next section.

3.5 Inclusion Property

One aspect that is also important is to determine whether the DRAM cache is inclusive [2], non-inclusive or exclusive with respect to the last-level cache. For the on-die cache hierarchy that consists of L1, L2 and the shared L3, we employ an inclusive approach. For the DRAM cache however, we chose to make it non-inclusive relative to the on-die cache hierarchy. This avoids the need for back-invalidation messages from the DRAM cache to the L3. The downside with non-inclusion is that the DRAM cache has to forward all DRAM cache snoops to the L3. However, since we expect that DRAM tags will likely be collocated with the last-level cache on-die, this is not an issue at all.

4 DRAM Cache Evaluation

In this section, we first describe the simulation framework and the workloads that we use to evaluate the DRAM cache performance, and then to analyze the experiment results. The major metrics that we focus on are IPC (Instructions per Cycle) for performance, and MPI (Misses per Instruction) for cache behavior.

4.1 Architecture Configuration

The simulated CMP architecture is illustrated in Figure 6. It consists of 8 cores operating at a frequency of 4GHz. The L2 cache is 512KB and is shared by two cores. L3 cache is 8MB by default (varied depending on DRAM tag placement optionsis) and is shared by all cores. Between the L2 and L3, there is an on-die interconnect that mimics a bi-directional ring. DRAM cache size is fixed at 64MB. Both DRAM cache and memory are off the main processor die. L2 and L3 are inclusive whereas L3 and DRAM cache are non-inclusive. The detailed experiment parameters are listed in Table 2.

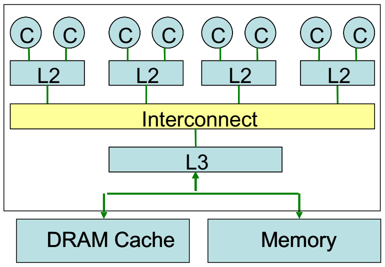

Figure 6. Simulation configuration

Table 2. CMP configurations and parameters

Parameters	Values
Core	4GHz, In-order
L2 cache	512K bytes, 8-way, 64-byte block, 18 cycles hit time
L3 cache	8M bytes, 16-way, 64-byte block, 30 cycles hit time
Interconnect BW	512GB/s
DRAM cache	64M bytes, 16-way, 64-byte block, 110 cycles access time, peak bandwidth at 64GB/s
Memory	400 cycles access time, peak bandwidth at 16GB/s

4.2 Workloads and Traces

We chose four key commercial server workloads: TPCC, SAP, SPECjbb2005 and SPECjappserver2004. TPC-C [17] is an online-transaction processing benchmark that simulates a complete computing environment where a population of users executes transactions against a database. The SAP SD 2-tier benchmark [14] is a sales and distribution benchmark to represent enterprise resource planning (ERP) transactions. SPECjbb2005 [16] is a Java-based server benchmark that models a warehouse company with warehouses that serve a number of districts (much like TPC-C). SPECjappserver2004 [15] is a J2EE 1.3 application server. It is a multi-tier e-commerce benchmark that emulates an automobile manufacturing company and its associated dealerships.

For all of these workloads, we collected long bus traces on Intel Xeon MP platform with 8 hardware threads running simultaneously with the last-level cache disabled. The traces include both instruction and data accesses, synchronization and inter-thread dependencies if there is any. They were replayed in the 8-core and 32-core simulator with different cache hierarchies and memory configurations as shown in Table 2.

4.3 Results and Analysis

In this section, we present the evaluation of the five DRAM cache options and its variants. We start by looking at off-die DRAM cache performance.

4.3.1 Off-die DRAM$ Performance

Figure 7 shows the performance of five workloads when we add an 64MB DRAM cache. It should be noted that this includes the memory access overheads of accessing both tags and data off-die. As the DRAM cache tags are placed off-die, the LLC size is not affected and fixed at 8M. The hit rate of the DRAM cache is also shown in the figure. We can see that all the four workloads get improvement from 6.1% to 18.6%. For TPCC, its hit rate is about 78%, which leads to a 12.7% increase in its throughput. SPECjbb is improved by 18.7%, although its hit rate is smaller than that of TPCC. This is because SPECjbb is more memory intensive, thus requires more memory bandwidth. Adding a DRAM cache reduces not only the average memory access latency, but also the memory bandwidth requirement. The hit rate for SAP and SPECjAppServer is 55.1% and 48.1%, and their IPC is improved by 6.1% and 9.7% respectively. When four servers are running simultaneously, the use of a DRAM cache can reduce the memory bandwidth requirement significantly and increase the throughput by 69.2%.

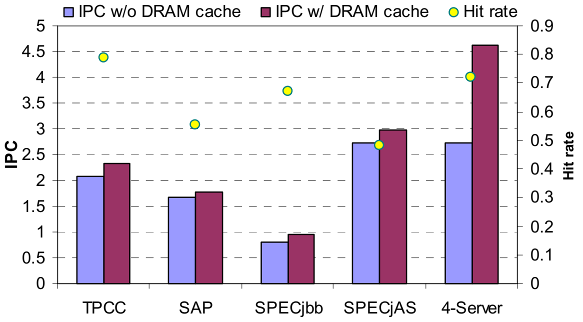

Figure 7. Off-Die DRAM cache performance

4.3.2 On-die Sectored DRAM Cache Performance

As we described in the previous section, placing tags on-die incurs a large space overhead. Therefore we consider sectored DRAM cache, which has smaller tag space. Given the same die area, the LLC size is reduced as we add on-die tags for DRAM cache. We ensure that the total size of LLC and on-die tag for DRAM cache is fixed at 8MB. Table 3 lists the on-die tag space for DRAM cache with various numbers of sectors, and the remaining LLC size. To make the number of sets power of two, the degree of set associativity is adjusted. For example with 5MB LLC size, the set associativity is set to 10 instead of 16. Besides this, some of the LLC size is also rounded up or down, but the increase of decrease is within 3% of their original size.

Figure 8 shows the performance for the five workloads as we vary the number of sectoring for DRAM cache. We can see that for the first four workloads, when the number of sectors is increased from 1 to 2, the performance is improved by about 5%. 4-sectored cache does not increase IPC any more, and 8-sectored cache even degrades the performance a little bit. To see why this happens, we also plot the behavior for LLC and DRAM cache. Figure 9 shows MPI for SPECjAppServer as an example. Other three workloads have the similar behavior. As can be seen, the MPI for DRAM cache is increased very little when we increase the number of sectoring. The MPI for LLC, on the other hand, is reduced significantly from 1 sector to 2 sectors (about 20%). The reduction is about 5% from 2 to 4 and 4 to 8 sectors. This is because the LLC size is increased by 27.5% from 1 sector to 2 sectors, whereas it is only increased by 10.8% and 4.9% from 2 to 4 and 4 to 8 sectors respectively. Therefore, the improved performance of LLC and degraded performance of DRAM cache makes 2-sectored cache achieve the best performance. This is also true for the last workload. Specifically, as we keep increasing the number of sector, the performance starts to degrade due to the fact that MPI for DRAM cache is increased much more than that for the first four workloads (not shown here due to space limitation).

Table 3. Space usage for LLC and on-die tag

No. of sector	LLC size (MB)	On-die tag size (MB)
1	5	3
2	6.375	1.625
4	7.0625	0.9375
8	7.40625	0.59375

To exploit the spatial locality existed in server workloads, we also look at the performance impact by sector prefetching, where the next line within the same sector is prefetched when a miss occurs. Figure 10 shows the performance when we use 2-sectored DRAM cache. The lines indicate the MPI without and with prefetching, and the bars show the corresponding IPC. We can see that prefetching can reduce the MPI significantly. MPI for TPCC is reduced by 30%, and MPI of all other four workloads are reduced by about 45%. However, the throughput is not improved as much. For SAP and SPECjbb, their IPC is increased about 8%, and for other workloads, their IPC is increased little. This is because SAP and SPECjbb have more spatial locality and unable to hide memory latency compared to the other workloads.

4.3.3 Partial Tag Performance

We look at the impact of DRAM cache with on-die partial tag. Fixing the total size of LLC and the on-die tag for DRAM cache as 8M, Table 4 shows the space consumption for LLC and on-die partial tag as we vary the number of partial tag bits. As we have already described in previous section, eight bits of partial tag seems to have a high true hit rate, therefore we choose the number of partial bits from 4 to 10.

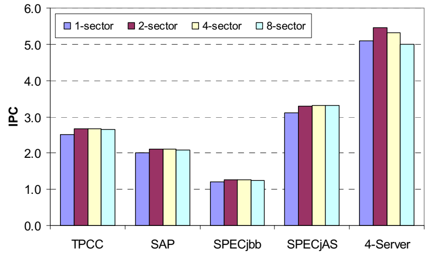

Figure 8. Impact of sectored DRAM cache

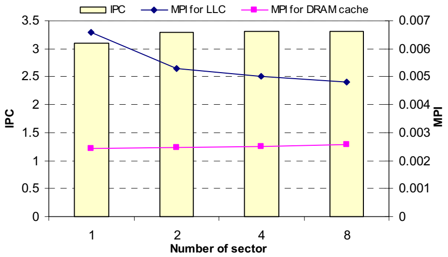

Figure 9. Impact of sectored DRAM cache on SPECjAS

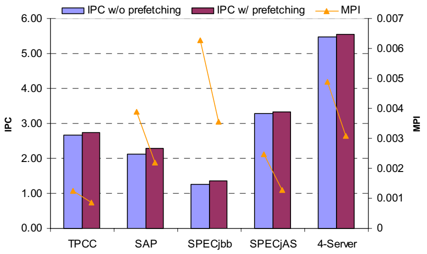

Figure 10. Impact of sector prefetching (2-sectored)

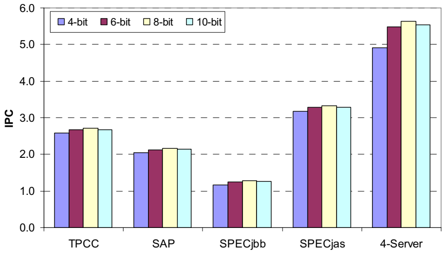

Figure 11. Impact of partial tag

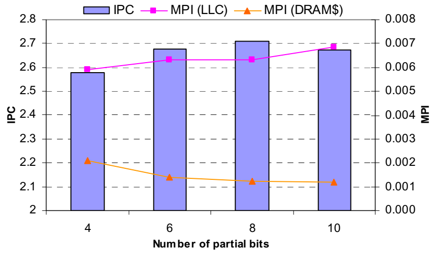

Figure 12. Impact of partial tag for SAP

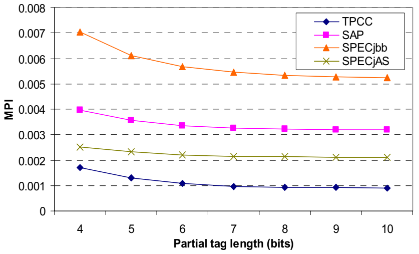

Figure 13. Impact of partial tag on MPI

Table 4. Space usage for LLC and on-die tag

Partial tag bits	LLC size (MB)	On-die tag size (MB)
4	7.25	0.75
6	6.6	1
8	6.75	1.25
10	6.5	1.5

Figure 11 shows the throughput of the five server workloads with various partial tag lengths. The IPC is increasing as we increase the partial tag length from 4 to 8, with 8 bits having the best performance, and then it starts reducing after 8. This behavior is because of the combined impact of LLC and partial tag. Take TPCC for example, Figure 12 illustrates the MPI for both LLC and DRAM cache. From the figure, we can see that the MPI for LLC is increasing as we increase the partial tag length due to the reduced LLC size. On the other hand, the MPI for DRAM cache is reducing because of the increased true hit rate from partial tag. While the MPI for LLC continues increasing even after 10-bit partial tag, the MPI for DRAM cache almost keeps constant after 8. Therefore, 8-bit partial tag DRAM cache shows the best performance. This is true for all the other four server workloads.

In previous figures, both the LLC size and the partial tag length are varied. To study more the partial tag performance, we ignore the partial tag space consumption and fix the LLC size at 8MB. Figure 13 illustrates the MPI of DRAM cache as we increase the partial tag length from 4 to 10. We can see that the MPI is reduced significantly from 4 to 6 (about 13% to 27%). After 6, the MPI is reduced much slower, and almost keeps constant after 8. In fact, 8-bit partial tag can achieve the same MPI as the full tag. Our measurement with the four workloads show that 8-bit partial tag can achieve the true hit rate at around 96% to 98% (not shown here due to space limitation).

4.3.4 Overall Performance

Figure 14 illustrates the performance comparison for the five server workloads with all the design options that we have discussed. First four workloads have the similar behavior. Compared to the based case, where no DRAM cache is used, off-die DRAM cache improves the performance by about 6 to 18%. With on-die tagged DRAM cache, SAP and SPECjbb get another 13% and 27% improvement, whereas TPCC and SPECjAppServer get only 7% and 4% increase. When 2-sectored DRAM cache is used, which reduces the on-die tag space about half, the throughput is improved by around 5 to 6%. Prefetching the next line improves SAP and SPECjbb about 7% while does not affect TPCC and SPECjAppServer as much. When 8-bit partial on-die tag is used, which saves 60% of the on-die full tag, the throughput is improved by 6 to 8% compared to the on-die full tagged cache. Notice that this improvement is a little bit more than that with a 2-sectored cache. Finally we combined a 2-sectored cache with 8-bit partial on-die tag, and also prefetch the next line in the sector. The last bar in the figure shows that it can improve the performance by 25 to 73% compared to the base case, and 9 to 17% compared to the on-die full tagged cache.

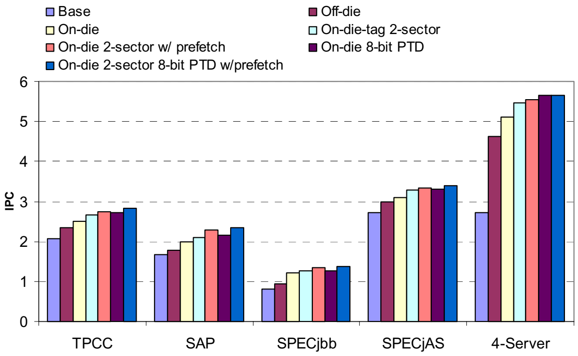

Figure 14. Performance comparison with various DRAM cache design options

When all four server workloads are running simultaneously on a 32-way CMP system, we see different behavior from the first four workloads. In the base case where no DRAM cache is used, the memory utilization is almost 100%. An off-die DRAM cache reduces the memory utilization by about 50%, which leads to 79% improvement in the overall throughput. Compared to the first four workloads where only 8 threads are running simultaneously, the DRAM cache reduces the memory bandwidth requirement significantly. With on-die tag DRAM cache, the performance is increased by another 11%. Finally if we combine all the design options, where the DRAM cache is 2-sectored with 8-bit partial on-die tag and prefetching, the throughput is improved the by 2x compared to the base case, and 22% compared to the on-die full tagged cache.

5 Conclusions and Future Work

In this paper, we investigated the integration of large-capacity, high-bandwidth and low-latency DRAM caches to address memory stall overhead. We showed that the DRAM caches of moderate size (64MB) provide a hit rate of 50 to 80% for key commercial workloads in 8-core and 32-core CMP platforms. We also described that placement of DRAM cache tags and its associated space overhead are the key issues that need to be addressed in order to achieve the performance potential of DRAM caches. We evaluated five different DRAM cache architecture options differing in placement of tags as well as tag reduction mechanisms (sectoring and partial tags). Through detailed evaluation using commercial server workloads, we showed that a hybrid DRAM cache organization (combing partial on-die tag and sectored cache with prefetching) can achieve a performance improvement of 25% to 75%. This makes DRAM caches very attractive for future CMP server platforms.

For future work, we will evaluate DRAM caches for multi-socket CMP platforms, and also consider other opportunities such as cache compression [7] and intelligent prefetching schemes to take additional advantage of large capacity, high bandwidth DRAM caches.

References

[1] C. Anderson and J.-L. Baer. Design and evaluation of a subblock cache coherence protocol for bus-based multiprocessors. *Tech. Report 94-05-02, Univ. of Washington*, 1994.

[2] J. Baer and W. Wang. On the inclusion properties for multilevel cache hierarchies. In *15th International Symposium on Computer Architecture*, 1988.

[3] B. Black and et al. Die stacking (3D) microarchitecture. In *39th IEEE/ACM International Symposium on Microarchitecture*, 2006.

[4] S. Choudhary, P. Caprioli, and et al. High performance throughput computing. In *38th Annual IEEE/ACM International Symposium on Microarchitecture*, 2005.

[5] R. Faramarzi. High speed trends in memory. http://www.jedex.org/images/pdf/reza_hynix_keynote.pdf.

[6] R. Golla. Niagara2: A highly threaded server-on-a-chip, 2006. Fall Microprocessor Forum.

[7] E. Hallnor and S. Reinhardt. A unified compressed memory hierarchy. In *International Symposium on High Performance Computer Architecture*, 2005.

[8] Intel Corporation. Intel dual-core processors - the first in the multi-core revolution. http://www.intel.com/technology/computing/dual-core.

[9] Intel Corporation. Tera-scale computing. http://www.intel.com/research/platform/terascale/index.htm.

[10] R. Iyer. Performance implications of chipset caches in web servers. In *International Symposium on Performance Analysis of Systems and Software*, 2003.

[11] C. Liu and et al. Organizing the last line of defense before hitting the memory wall for cmps. In *International Symposium on High Performance Computer Architecture*, 2004.

[12] Micron. FBDIMMS – a revolutionary new approach to memory modules. http://www.micron.com/products /modules/ddr2sdram/fbdimm.html.

[13] A. Moga and M. Dubois. The effectiveness of sram network caches in clustered dsms. In *International Symposium on High Performance Computer Architecture*, 1998.

[14] Sap America Inc. SAP standard benchmarks. http://www.sap.com/solutions/benchmark/index.epx.

[15] SPEC. SPECjAppServer java application server benchmark. http://www.spec.org/jAppServer.

[16] SPEC. SPECjbb2005. http://www.spec.org/jbb2005.

[17] TPC. TPC-C design document. http://www.tpc.org/tpcc.

[18] R. B. Tremaine and et al. Pinnacle: Ibm mxt in a memory controller chip. *IEEE Micro*, March/April 2001.

[19] VMware Corporation. Server consolidation and containment with vmware virtual infrastructure. http://www.vmware.com/pdf/server_consolidation.pdf.

[20] Z. Zhang, Z. Zhu, and X. Zhang. Cached dram: A simple and effective technique for memory access latency reduction on ilp processors. *IEEE Micro*, July/August 2001.

[21] Z. Zhang, Z. Zhu, and X. Zhang. Design and optimization of large size and low overhead off-chip caches. *IEEE Transactions on Computer*, July 2004.