

Addendum to “Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches”

Gabriel H. Loh^{*}
^{*}AMD Research
Advanced Micro Devices, Inc.
gabe.loh@amd.com

Mark D. Hill^{†*}
[†]Department of Computer Sciences
University of Wisconsin – Madison
markhill@cs.wisc.edu

June 1, 2012

Abstract

The MICRO 2011 paper “Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches” proposed a novel die-stacked DRAM cache organization embedding the tags and data within the same physical DRAM row and then using compound access scheduling to manage the hit latency and a MissMap structure to make misses more efficient. This addendum provides a revised performance analysis, an updated performance analysis using more up-to-date timing and system parameters, and also provides additional analysis on the performance of the system.

The analysis shows that Compound Access Scheduling (CAS) and the MissMap are still effective, but not as effective as we originally found. Based on the original parameters used in the MICRO’11 paper, a 1GB DRAM cache with CAS and a MissMap provides approximately 25-50% of the performance benefit, rather than about 80% as previously reported, of an impractical DRAM cache implementation using SRAM tags depending on cache size (128MB-1GB). Based on more up-to-date parameters, the CAS and MissMap approach provides about 50-60% of the benefit compared to the SRAM-tags approach.

1 Errata: Revised Simulation Results

The simulator used to generate the performance results presented in the original MICRO’11 paper¹ was found to have some bugs that impacted the reported results. The primary sources of error included:

1. DRAM-cache fills were treated as write requests because the installation of a cache line requires the writing to the row/set of the cache. However, the simulator’s implementation of generic write requests first checks for whether or not there is a hit in the cache. Because this is for a fill, there will never be a cache hit, and so the subsequent DRAM cache traffic gets skipped. The net result is that the fill operations use up less DRAM cache bandwidth than they should.
2. The DRAM-cache data bus model incorrectly handled some scheduling cases, resulting in over-subscription of the bus. The net result is that some DRAM cache requests would be serviced earlier than they otherwise should.
3. The MissMap latency was misconfigured to have a latency of only 10 cycles when it should have been equal to the L3 cache (24 cycles). This makes every DRAM-cache hit and miss 14 cycles faster than they should be.

The results from the MICRO’11 paper are shown in Figure 1(a), with only the Compound Access Scheduling plus MissMap and the SRAM-tags configurations for clarity. Figure 1(b) shows the revised results after correcting for the simulation errors. Note that in later sections of this addendum, we will revisit the performance results with more up-to-date system and DRAM parameters that provide a more accurate view of the expected benefit of the approach. These updated results have also been further corroborated by analytical modeling.

¹These results were also replicated in our IEEE Micro Top Picks paper, “Supporting Very Large DRAM Caches with Compound-Access Scheduling and MissMaps”.

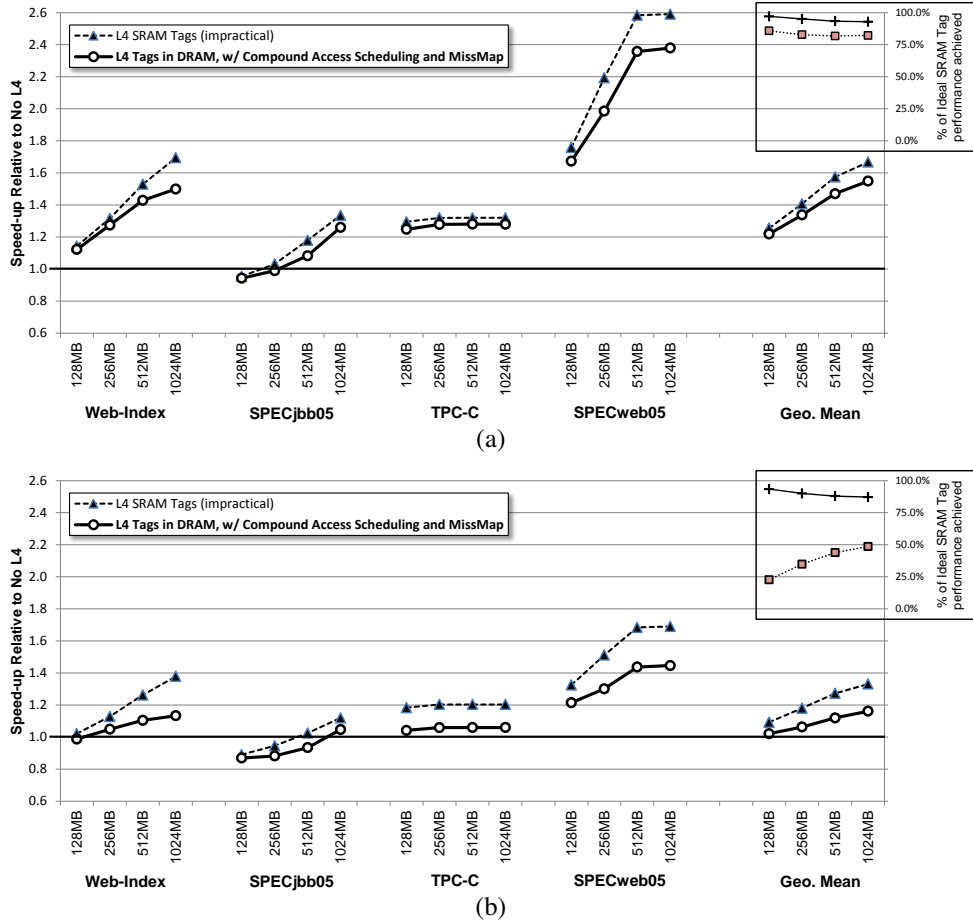


Figure 1: Performance results for a DRAM cache using compound access scheduling and a MissMap and impractically large SRAM tags corresponding to (a) the original MICRO'11 results, and (b) updated results with simulator errors corrected.

In the original MICRO'11 results, Compound Access Scheduling plus MissMap (which we will simply refer to as MissMap from now) bridges approximately 80% of the performance gap from no DRAM cache to the SRAM-tags configuration (on average). In the updated results, MissMap only covers about half the distance from no DRAM cache to the SRAM-tags configuration. When the DRAM cache size is small, the presence of the cache can hurt performance. When hit rates are too low, the DRAM cache simply adds more cycles to the load-to-use latency (i.e., average memory access time increases). This is true even with the MissMap, as the MissMap effectively adds an *extra* L3 access to the overall latency. Note that for SPECjbb05, even the impractical case using the SRAM tags results in a performance degradation for smaller cache sizes.

Note that in the inset sub-figures, we have shown the “% of Ideal SRAM Tag performance achieved,” but we have now added a second lower curve (□). The original curve is still shown on top (+) for reference. In the MICRO'11 paper, we computed this metric by simply taking the performance of the MissMap configuration and dividing it by the performance of the SRAM-tags configuration. Such a metric can potentially be misleading: for example, assume that some optimization provides a speed up of $1.1\times$. By doing *nothing*, one achieves a speedup up $1.0\times$, and so this metric

Parameter	Off-Chip	Die-Stacked	Improvement
tRCD	13.75ns	8ns	1.72×
tCAS	13.75ns	8ns	1.72×
tRP	13.75ns	15ns	0.92×
tRAS	35ns	26ns	1.35×
tFAW ¹	30ns	n/a	n/a
tWR	15ns	12ns	1.25×
tWTR	7.5ns	2ns	3.75×
tRC	48.75ns	40ns	1.22×
tRRD	7.5ns	3ns	2.50×
channels ²	2	8	4×
bus speed ³	800 MHz	1.0GHz	1.25×
bus width	64 bits	128 bits	2.00×
burst ⁴	5ns	2ns	2.50×
interconnect ⁵	15ns	n/a	n/a

Notes:

¹ Stacked DRAMs are expected to have no activation window limitation. The tFAW constraint originates from the limited number of package pins on conventional DRAM, which in turn limits the power delivery. For die-stacked DRAM, the wide TSV or microbump interface provides many, many more connections thereby enabling better and more reliable power delivery.

² The MICRO'11 work assumed four memory channels for the die-stacked DRAM, which is based on the lower-power/mobile Wide-IO memory standard (4 channels × 128 bits each = 512-bit total interface). Projections for high-performance stacked DRAMs may include wider interfaces (e.g., 8 channels/1-kbit, or possibly more).

³ This is the base memory command bus speed. Data transfer rate is at twice this speed (i.e., DDR).

⁴ This “burst” is the amount of time on the bus to transfer a 64-byte block of data. 64B on a 64-bit bus requires eight transfers; with DDR giving two transfers per memory cycle, this yields four cycles; at 800MHz, this is 5ns. For the stacked DRAM, 64B on a 128-bit bus requires four transfers, or two cycles, which at 1GHz is equal to 2ns.

⁵ Off-chip communications require additional latency for the request to make its way through the various interface circuitry (e.g., synchronizing across timing domains, physical interface drivers, etc.) and time to propagate the signal across the PCB/motherboard. This was not modeled in the original MICRO'11 results.

Table 1: Timing parameters used for the simulations presented in this addendum.

would state that doing nothing provides $\frac{1.0}{1.1} = 90.9\%$ of the speedup of the optimized case. The new lower curves in the inset figures of Figure 1 show the percentage of the performance *benefit* obtained (i.e., $\frac{speedup_{MissMap} - 1.0}{speedup_{SRAM\ tags} - 1.0}$).

2 Updated Parameters

Over time, the projected performance of die-stacked DRAM (in terms of raw latency) has gone from very optimistic (e.g., 3-4× faster than off-chip DRAM) to more conservative values. For example, our original MICRO paper effectively assumed a 2× reduction in stacked-DRAM latencies by keeping all parameters constant in terms of clock cycles, but then assuming a bus running at twice the speed as the off-chip memory interface. As the technologies mature, the projections on the stacked DRAM performance continue to be refined.

2.1 Parameter Values

In the remaining results presented in this addendum, we make use of the parameter values listed in Table 1. The table includes the parameters for both the off-chip memory, the die-stacked DRAM, and the “improvement,” showing that for many parameters, an assumption of a 3-4×, and even a 2× improvement may be too optimistic. It is of course possible (or even likely?) that future die-stacked DRAMs will continue to improve latencies, but these values reflect our current best guesses.

2.2 Updated Performance Values

Figure 2 shows the performance of the MissMap and SRAM-tags configurations using the updated memory parameters described above. These results also assume two ranks per channel instead of only one (as assumed in the MICRO'11 paper) as server systems often make use of multi-rank DIMMs.² On average, the overall trends are somewhat similar to the results shown in Figure 1(b), i.e., after simulator errors were corrected for, where the MissMap approach provides

²The difference in the results between single-rank and dual-rank results are not very large.

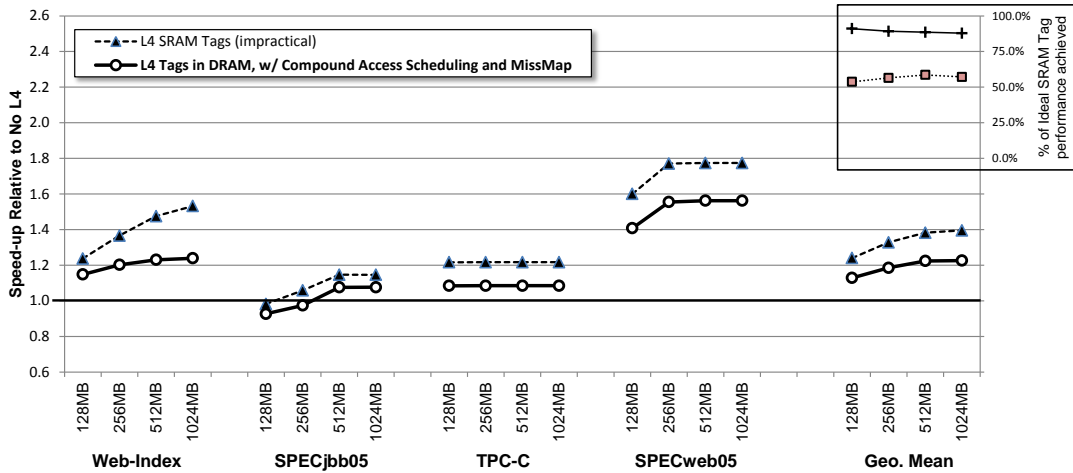


Figure 2: Performance results using the updated memory and system parameters shown in Table 1. All remaining simulation parameters are identical to those used in the original study.

approximately one-half of the performance benefit achievable with the impractical SRAM tags configuration. With the updated configuration, the benefit of the DRAM cache increases slightly due to the increased latency of off-chip DRAM when the off-chip interconnect overheads (15ns) are accounted for. This is offset by the fact that the primary parameters for the revised stacked DRAM (e.g., tCAS, tRCD) are not as fast as previously assumed ($1.72\times$ versus $2.0\times$).

Overall, it is slightly disappointing that the MissMap approach did not perform as well as we originally thought it did (i.e., for the 1GB DRAM cache configuration, MissMap provides 57.2% of the performance *benefit* of the SRAM tags case, whereas the MICRO'11 results suggested that the MissMap approach provided 82.2% of the benefit). However, from a research opportunity perspective, this is also encouraging as it shows that there is room for new ideas to successfully bridge the rest of the gap (or to perhaps even go further!).

3 Performance Deconstruction

The stacked DRAM technology only provides a slightly faster memory. There is a decent latency savings in avoiding the overheads of the off-chip interconnect interface ($\sim 15\text{ns}$). The raw command latencies only get a little faster ($\sim 1.7\times$ per command). When using the stacked DRAM in a tag-in-memory cache organization, however, the latency benefits end up being significantly reduced by the overhead of having to read the tags and data out of the DRAM. While Compound Access Scheduling makes this far more feasible than a naive command scheduling approach, accessing a cache line still requires:

1. Access the MissMap (assume it says the requested line is present)
2. Activate the target row (tRCD)
3. Issue three back-to-back-to-back read commands for the tag blocks (tCAS)
4. Receive the data for the blocks (burst $\times 3$)
5. Perform the tag check (1 cycle)
6. Issue another read command for the data block (tCAS)

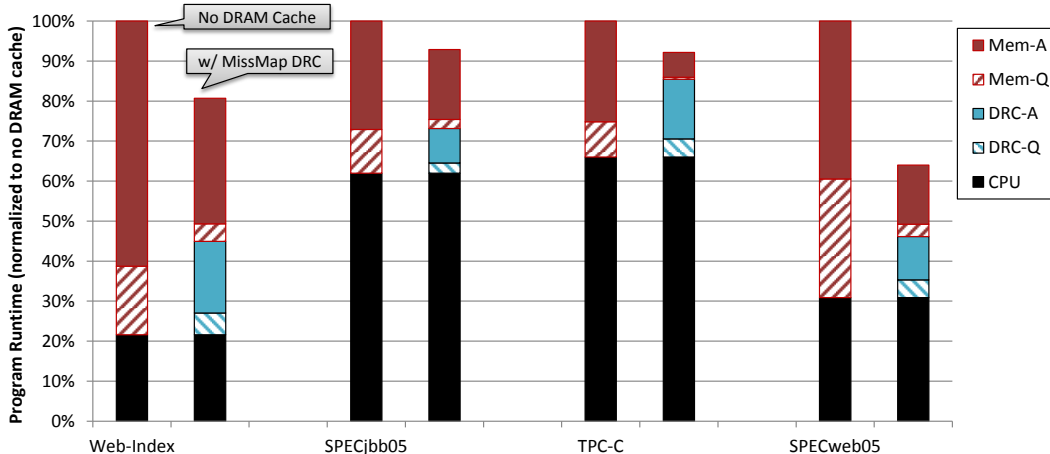


Figure 3: Break down of overall execution time. For each workload, the left bar shows the execution time for the baseline that does not use a DRAM cache, whereas the right bar shows the breakdown for a MissMap-based DRAM cache.

7. Receive the data (burst)

Contrast this to an off-chip memory access which only requires an activate (assuming no row conflict) and a read. So if the access latency of the stacked-DRAM cache is not significantly faster than that of the off-chip memory, where does the performance come from? Figure 3 shows for each workload, the breakdown of where the machine is spending its time in the baseline case with no DRAM cache (left bar per pair) and the case with the MissMap-based DRAM cache (right bar). The applications that observe the greatest benefits are, unsurprisingly, the ones that spend the most time in main memory (Web-Index and SPECweb05).

For each bar, the bottom black segment is the portion of time spent on compute. Above that, memory access time (whether to main memory (Mem) or to the DRAM cache (DRC)) is further sub-divided into queuing latency (Q) and time spent actually accessing the DRAM arrays (A)³. One source of the performance benefit comes from the reduction in the overall queuing delays (i.e., the sum of the two striped sections in the DRAM cache bar is less than the striped section of the baseline’s bar). The other major source of performance benefit, which was briefly discussed in the original MICRO’11 paper, comes from the increased row-buffer hit rates for the remaining off-chip memory accesses, which is illustrated in the figure by the reduction in height of the top-most bar segment. In the cases where the performance benefit is much smaller (SPECjbb05 and TPC-C), the reductions in queuing latency and the off-chip array latency are largely canceled out by the relatively large latency of having to access the die-stacked DRAM multiple times (even with compound access scheduling) when using the stacked DRAM as a cache.

4 Conclusions

While the magnitude of the performance results are not as great as was previously reported in the MICRO’11 paper, the general approach still provides performance benefit for heavily memory-intensive scenarios. Instead of closing the gap to the SRAM-tags configuration by 82%, the MissMap approach only provides 57% of the benefit (in the 1GB case under the most-recent parameters). From a research perspective, this updated result is actually exciting, as it indicates that there is still room for additional innovation and improvements to make effective use of die-stacked DRAM as a cache, and this should drive and encourage further research on the topic.

³The array latency is measured as the time from when a request first arrives until the first memory command (e.g., ACT) is issued. The array latency covers everything after that; note it is possible that the array latency ends up including some additional “queuing” latency from, e.g., having to wait for command or data bus availability. Also, because the simulator charges the interconnect latency at the end of handling the request, this also gets bundled together with the array latency in these statistics.