RESEARCH FEATURE

Concurrent Average Memory Access Time

Xian-He Sun and Dawei Wang,
Illinois Institute of Technology

Traditional memory performance metrics, such as average memory access time (AMAT), are designed for sequential data accesses and can prove misleading for contemporary cache technologies that increasingly rely on access concurrency. C-AMAT, a new performance metric, accounts for concurrency at both the component and system levels for modern memory design.

Growing disparities between processor and memory speeds have resulted in a so-called "memory wall,"[1,2] an ever-widening gap between CPU and chip-memory performance. Cache hierarchies currently serve to mitigate delays in off-chip main memory accesses, but as the memory wall looms larger, long delay times will penetrate into cache hierarchies as well. Intel's Nehalem CPU architecture offers a good example: here, each L1 data cache has a 4-cycle hit latency, while each L2 cache has a 10-cycle hit latency. Similarly, IBM's Power 6 has a 4-cycle L1 cache hit latency and a 24-cycle L2 cache hit latency. Modern multicore processor last-level latencies can exceed 100 cycles.

Research to alleviate these performance gaps has focused on improving memory system concurrency—that is, methods that support multiple requests simultaneously. Advanced designs utilizing multiport, multibanked, and pipelined cache,[3] for example, improve cache hit concurrency; complementary techniques, such as nonblocking cache,[4] improve cache miss concurrency; and processor instruction-level parallelism (ILP) techniques, such as out-of-order execution, multiple-issue pipelining, simultaneous multithreading, and chip multiprocessing, improve both cache hit and miss concurrency. These advanced cache optimizations and processor ILP techniques mean that tens or even hundreds of cache/memory accesses can coexist in the memory hierarchy simultaneously. Thus, a single cache miss is no longer a determining factor for overall memory system performance.

However, existing memory metrics—miss rate (MR) and average miss penalty (AMP) as calculations of average memory access time (AMAT)—still measure hits and misses based primarily on sequential single-access activity, and so are inadequate for purposes of measuring concurrent cache memory access activity. In addition, concurrency effectiveness is application and implementation dependent: accurate data storage and access measurements can vary. Highly data-intensive applications, in particular, require an alternative to existing memory metrics in order to take full account of the various concurrency methods available for modern memory systems.

To this end, we propose concurrent average memory access time (C-AMAT), an extension of AMAT that we developed as a more accurate metric for concurrent memory systems. C-AMAT formulates memory access delay as a summation of total access delays across cache hierarchies. It introduces two new average concurrency parameters—hit concurrency and miss concurrency—at each level of the memory hierarchy and, in so doing, posits that hit concurrency will always improve performance, while a cache miss may or may not reduce overall memory system performance, depending on the corresponding hit concurrency. C-AMAT allows for more accurate evaluation of concurrent memory behavior at both the component and the system levels than traditional memory metrics, which can provide misleading measurements when concurrency is a factor. C-AMAT is a powerful tool for determining architecture design choices, from hit–miss ratio to hit–miss concurrency.

C-AMAT FORMULATION

C-AMAT is formulated like AMAT, but takes into consideration concurrent hit and concurrent miss accesses. Quantitatively, C-AMAT is defined as the total memory access cycles (that is, the total number of cycles executed in which there is at least one outstanding memory reference), represented as T_{MemCycle}, divided by the total number of memory accesses, represented as C_{MemAcc}:

$$\text{C-AMAT} = \frac{T_{\text{MemCycle}}}{C_{\text{MemAcc}}}. \tag{1}$$

Concurrency is implicit in Equation 1, which assumes that the memory cycles are parallel: when several memory accesses coexist during the same cycle, T_{MemCycle} increases by only one. In other words, in counting memory access cycles, we adopt an *overlapping* mode to account for advanced modern cache and memory system structures—pipelined cache, multiported cache, nonblocking cache, and the like—in which multiple hit accesses and miss accesses can overlap one another. T_{MemCycle} assumes this concurrency.

Another important aspect of T_{MemCycle} is that it includes only those clock cycles that have memory access activities; cycles without memory references are excluded. This is based on earlier work[5] where we described memory access per cycle (APC) or, more accurately, memory access per memory active cycle (APMAC), a new performance metric designed to measure concurrent memory system performance as verified using the statistical variable *correlate coefficient*. C-AMAT is the reciprocal of APC, and based on this APC measurement methodology, we obtain the following value for C-AMAT:

$$\text{C-AMAT} = \frac{1}{\text{APC}} = \frac{T_{\text{MemCycle}}}{C_{\text{MemAcc}}}. \tag{2}$$

To use C-AMAT as an analytic tool, like AMAT, we must first derive a component-based, parameterized formula for C-AMAT, which we do by extending AMAT with the addition of concurrency.

Extending the AMAT formulation to include concurrency

Traditional AMAT is calculated as HitCycle + MR × AMP, where HitCycle (H) is the hit time of memory accesses, MR is the miss rate of cache accesses, and AMP is the average miss penalty. AMP is calculated as the sum of all single-miss access latency divided by the total number of miss accesses.

AMAT does not consider the concurrency of memory accesses in terms of either hits or misses, based on the assumption that memory accesses are sequential, one after another; further, AMAT does not take into account that with concurrent accesses, hits and misses may coexist in the same cycle. The sequential assumption governing AMAT worked well in the past, but applies less accurately for modern processor architectures and memory systems where concurrency is paramount. To properly analyze concurrency, we extend AMAT with concurrency parameters for hit and miss accesses, and propose a new counting method for MR and AMP that considers the relation between concurrent hits and misses:

$$\frac{H}{C_H} + \text{MR} \times \frac{\text{AMP}}{C_M}. \tag{3}$$

The parameter C_H represents hit concurrency, which results from multiport, multibanked, and pipelined cache structures, while the parameter C_M represents miss concurrency, which results from nonblocking cache structures; these parameters can also represent both hit concurrency and miss concurrency that result from processor ILP design techniques, such as out-of-order execution and multiple-issue pipelining.

Equation 3 redefines MR as the number of pure misses—with a pure miss assumed to contain at least one miss cycle with no hit access activity—divided by the total number of accesses. When measuring private caches—for example, the L1 data cache—for multicore microprocessors, pure misses are measured based on a "per-core" mode: every core has its own detecting logic, and that logic measures only that core's private cache accesses. When a miss occurs without a hit access inside the private cache, the corresponding cycle is measured as a "pure miss cycle" for that core. For shared caches—that is, L2 caches, L3 caches, and so on—pure miss cycles are measured based on an "all-core" mode: if there is no cache hit access from any core or cores, then a miss cycle is counted as a "pure miss cycle." Equation 3 also redefines AMP as the average number of pure miss cycles per miss access.

C-AMAT can be calculated using hit and miss concurrency factors for architecture design choices. The critical question is how to obtain an accurate average C_H and C_M. Here we apply a weighted method to calculate the average value.

If C_H is the average hit cache concurrency, by definition, it is equal to

$$C_H = \sum_{i=0}^{N} C_i \times \frac{t_i}{T_H}, \tag{4}$$

where N represents the total number of cache hit phases. For each hit phase, the value of C_i remains constant; C_i is the hit concurrency during phase i; t_i is the number of cycles of phase i. Note these hit access phases include only cache cycles containing at least one cache hit activity; clock cycles without any hit accesses cannot be counted in a hit access phase. T_H is the total hit cycles in the overlapping mode; therefore

$$T_H = \sum_{i=0}^{N} t_i \ . \tag{5}$$

Similarly, the average miss cache concurrency, C_M, can be defined as follows:

$$C_M = \sum_{j=0}^{M} C_j \times \frac{t_j}{T_M} \ , \tag{6}$$

where M is the total number of pure cache miss phases. In each miss phase, the value of C_j does not change; C_j is the miss concurrency during phase j; t_j is the number of cycles of phase j. Note that the pure miss phases only include the cache cycles that contain at least one pure cache miss activity. If one clock cycle contains a miss access as well as a hit access or does not contain any miss access, this cycle is not counted in pure miss phases. T_M is the total number of pure miss cycles in the overlapping mode; therefore

$$T_M = \sum_{j=0}^{M} t_j \ . \tag{7}$$

Based on these initial analyses and definitions, we further prove that the formula for C-AMAT in Equation 1 is equal to Equation 3 and show how the C-AMAT formula established in Equation 3 can be extended recursively from L1 cache memory and apply equally to lower levels of cache memory.

Proof of equality for the C-AMAT formula

H is the number of hit cycles when accessing the current cache layer. Every cache access needs to spend H cycles to determine whether this is a hit or a miss access. Note H is a constant value in our cache model.

MR (miss rate) for our purposes is an extended version of the traditional miss rate definition with the consideration of concurrency. Only when a miss access has no overlap with any hit accesses is this miss access considered a pure miss access. Thus,

$$\text{MR} = \frac{C_{\text{MemPMiss}}}{C_{\text{MemAcc}}} \ . \tag{8}$$

In this formulation, C_{MemPMiss} is the total number of pure misses.

AMP is the average miss penalty, considering only pure miss accesses:

$$\text{AMP} = \frac{T_{\text{MemPMiss}}}{C_{\text{MemPMiss}}} \ , \tag{9}$$

where T_{MemPMiss} is the sum of total pure miss cycles. The pure miss cycles are the cache miss access cycles without any hit access. Thus,

$$\begin{aligned}
&\frac{H}{C_H} + \text{MR} \times \frac{\text{AMP}}{C_M} = \frac{H}{\sum_{i=0}^{N} C_i \times \frac{t_i}{T_H}} \\
&+ \frac{C_{\text{MemPMiss}}}{C_{\text{MemAcc}}} \times \frac{T_{\text{MemPMiss}}}{C_{\text{MemPMiss}}} \times \frac{1}{\sum_{j=0}^{M} C_j \times \frac{t_j}{T_M}} \\
&= \frac{H \times T_H}{\sum_{i=0}^{N} C_i \times t_i} + \frac{T_{\text{MemPMiss}}}{C_{\text{MemAcc}}} \times \frac{T_M}{\sum_{j=0}^{M} C_j \times t_j} \ .
\end{aligned} \tag{10}$$

Further, because

$$\sum_{i=0}^{N} C_i \times t_i = C_{\text{MemAcc}} \times H \tag{11}$$

and

$$\sum_{i=0}^{M} C_j \times t_j = T_{\text{MemPMiss}} \ , \tag{12}$$

it follows that Equation 10 is

$$\begin{aligned}
&\frac{H \square T_H}{C_{\text{MemAcc}} \square H} + \frac{T_{\text{MemPMiss}}}{C_{\text{MemAcc}}} \square \frac{T_M}{T_{\text{MemPMiss}}} = \frac{T_H + T_M}{C_{\text{MemAcc}}} \\
&= \frac{T_{\text{MemCycle}}}{C_{\text{MemAcc}}} = \text{C-AMAT}.
\end{aligned} \tag{13}$$

Thus,

$$\text{C-AMAT} = \frac{H}{C_H} + \text{MR} \times \frac{\text{AMP}}{C_M} \ . \tag{14}$$

Equation 14, then, reveals five important parameters that determine overall memory performance: hit latency (H), hit concurrency (C_H), miss rate (MR), AMP, and miss concurrency (C_M). Though the concepts of MR and AMP in C-AMAT are similar to their counterparts in AMAT, C-AMAT excludes concurrent hit and miss accesses for overlapping cycles.

As we will see, different processor and cache design choices will affect one or more C-AMAT parameters in different ways—sometimes benefiting a specific factor while acting to the detriment of others. These parameters in turn affect overall data access time, and therefore determine underlying design choices. Only by considering various factors comprehensively can we determine the most appropriate configuration given a particular memory system.

CACHE CONCURRENCY MEASUREMENT

Equation 14 introduces two concurrencies existing at each memory hierarchy layer: hit concurrency and miss concurrency. Hit concurrency reflects parallelism in cache tag query and cache data access. It does not matter whether a cache access ultimately results in a hit or a miss; the cache access in itself requires the system to spend a certain fixed cycle performing the cache tag query. In advanced cache designs such as multiport and pipelined cache, the maximum hit cache concurrency is (#cache port × #cache pipeline stage).

For example, the Advanced Micro Devices Opteron CPU has a two-port L1 data cache and a three-cycle pipeline stage for cache access;[6] thus, the maximum hit concurrency is 2 × 3 = 6. The miss concurrency is usually determined by the number of miss status holding register (MSHR) entries. The maximum miss concurrency is equal to the number of outstanding cache misses MSHR can support.

Compared with the traditional AMAT miss rate measurement, C-AMAT does not include any miss cycles that overlap with a hit cycle because when a hit occurs, the memory does not block CPU performance. Only pure miss access cycles cause the CPU to discontinue execution. Thus, the challenge of measuring miss concurrency lies in eliminating overlapping cycles that contain hit accesses. In other words, the miss concurrency detector needs to simultaneously be aware of both cache hit accesses and miss accesses. Only pure miss cycles, defined as miss cycles that do not overlap with hit cycles, are counted in C-AMAT as miss cycles. Figure 1 presents a structure for detecting cache hit and miss concurrency using the C-AMAT metric.

The hit concurrency detector (HCD) counts the total hit cycles and records each hit phase in order to calculate the average hit concurrency. The hit cycles are the clock cycles containing at least one hit access activity. The HCD also tells the miss concurrency detector (MCD) whether a current cycle has a hit access or not.

The MCD is a monitoring unit that counts the total number of pure miss cycles and records each pure miss phase in order to calculate the average miss concurrency, pure miss rate, and pure miss penalty. With the information provided by the HCD, the MCD is able to tell whether a cycle is a pure miss cycle, and whether a miss is a pure miss. Further, the pure miss rate and average pure miss penalty can be calculated based on total miss data. Finally, C-AMAT can be measured based on the five parameters on the right side of Equation 14. As with AMAT, the AMP in Equation 14 can be further extended in a straightforward fashion as a composition of hit and miss for each next cache level in the memory hierarchy.

Table 1. Default processor and cache configuration parameters for simulated testing of C-AMAT.

Device/parameter	Value

Processor
Function units
ROB (reorder buffer) and LSQ (load/store queue) size
	1 core, 4 GHz, 4-issue width
	6 IntALU: 1 cycle; 1 IntMul: 3 cycles;
	2 FPAdd: 2 cycles; 1 FPCmp: 2 cycles;
	1 FPCvt: 2 cycles;
	1 FPMul: 4 cycles; 1 FPDiv: 12 cycles
	ROB 64, LQ 48, SQ 24

L1 caches	32 KB inst/32KB data, 2-way, 64B line, 4-cycle hit latency, inst/4-cycle data, ICache 8 MSHR entry, DCache 8 MSHR entry
L2 cache	512 KB, 16-way, 64B line, 24-cycle hit latency, 16 MSHR entry
DRAM latency/width	240-cycle access latency/64 bits

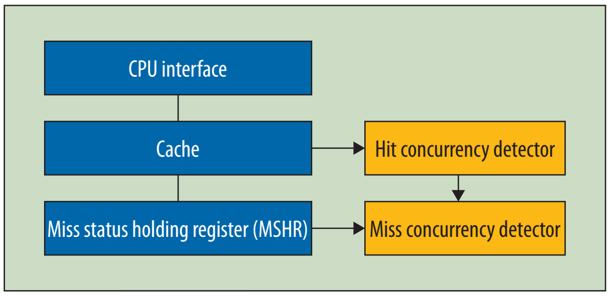
CPU interface
Cache
Hit concurrency detector
Miss status holding register (MSHR)
Miss concurrency detector

Figure 1. Structure for detecting cache hit concurrency and cache miss concurrency using the C-AMAT metric.

EXPERIMENTAL DESIGN AND RESULTS

To establish the feasibility of incorporating C-AMAT in making cache/memory architecture design choices that take concurrency into account, we first adopted a detailed CPU model in the GEM5 simulator,[7] which supports out-of-order, speculative, superscalar, and multithreading execution for a single core, as well as complicated cache hierarchies for multicores with different cache coherency protocols. Unless stated otherwise, our simulations assume the default processor and cache configuration parameters shown in Table 1. It is a common, general-purpose computing system configuration.

We conducted our simulations based on 24 benchmarks from the SPEC CPU2006 suite[8] (some benchmarks were omitted because of simulator compatibility issues), compiled using GCC 4.3.2 with -O2 optimization; for all benchmarks, we adopted the suite-provided reference input sizes. For each of the 24 benchmarks, we collected statistics based on 10 million simulated instructions. The memory performance results we present are the average value for all 24 benchmarks.

Having previously verified APC's accuracy as a measurement,[5] in this study we focused on establishing C-AMAT's usefulness in evaluating three widely implemented advanced technologies: *multiple-issue pipelining*, which increases both cache hit and miss concurrency; *nonblocking cache*, which increases cache miss concurrency; and *multicore technologies*, which also increase both cache hit and cache miss concurrency. (We discuss AMAT and C-AMAT performance comparisons more extensively elsewhere.[9])

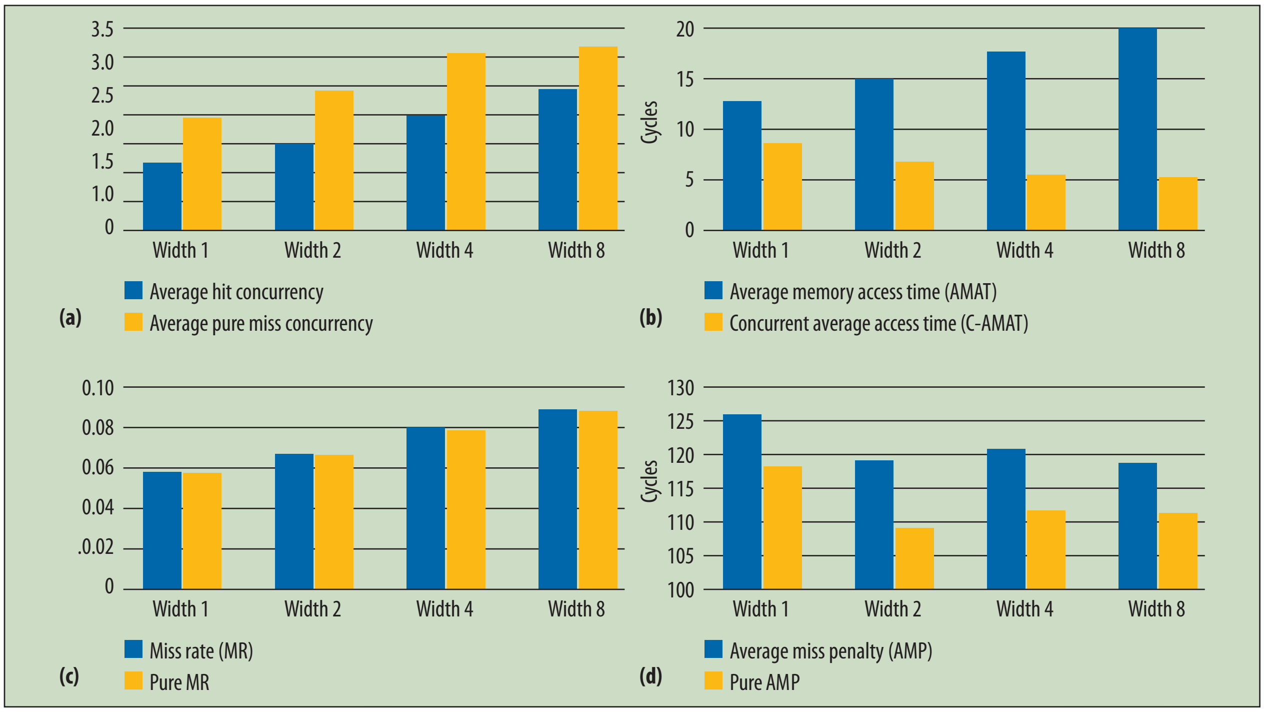

Figure 2. **Memory performance measured as multiple-issue pipeline width increases. (a) Average hit concurrency and average pure miss concurrency of L1 data cache as pipeline issue width increases from 1 to 8, showing memory improvement. (b) Average memory access time (AMAT) and concurrent average memory access time (C-AMAT) measurements as pipeline width increases: AMAT shows decrease in memory performance, while C-AMAT accurately reflects an increase. (c) Miss rate (MR) measurement and pure MR as pipeline width increases. (d) Average miss penalty (AMP) measurement and pure AMP as pipeline width increases.**

Based on our initial default configuration, we modified only one or two parameters in each simulation to show the influence of the specific memory design choice on cache concurrency and C-AMAT. We used the same variation for other common memory performance metrics (MR, AMP, and AMAT) to establish a base level of accuracy for each, assuming that the winner among these would correlate to the targeted processor design.

The results of our simulations comparing AMAT, traditional MR, and traditional AMP with C-AMAT for purposes of measuring concurrent cache hits and misses show that only C-AMAT consistently matches the actual design choices for modern processors.

Hit concurrency results based on multiple-issue pipeline width increases

Multiple-issue pipelining represents an important improvement in processor microarchitecture design to increase ILP. It allows multiple instructions to be fetched, decoded, issued, executed, and committed within the same cycle. Because of algorithm data dependency, the high penalty paid for branch misprediction, load data constraints, and general power-wall limitations, increasing pipeline width is not a viable option, so most commercial processors adopt a 4- to 8-width pipeline per core at different processing stages.[3] For simplicity, our study assumes the same pipeline width at each stage.

As the graphs in Figure 2 demonstrate, only the memory performance reflected by C-AMAT consistently matches the performance improvement trend indicated by this ILP technique.

Figure 2a shows the average hit concurrency and the average pure miss concurrency of an L1 data cache as pipeline issue width increases from 1 to 8. As is clear, the average hit concurrency increases at a greater rate than average pure miss concurrency when the pipeline width is larger than 4, representing an improvement in memory.

But, as Figure 2b shows, traditional AMAT increases as pipeline width increases, which indicates a decrease in memory performance—a finding contrary to the fact that application performance should improve with increased pipeline issue width.[3] C-AMAT, on the other hand, not only correctly describes the overall memory performance as increasing, but also correctly reflects the diminishing returns that come with a pipeline issue width greater than 4. Also interesting to note, this C-AMAT result confirms current designs that select issue widths of between 4 and 6 as optimal for general-purpose processors, further demonstrating C-AMAT'S usefulness in practice.

Figures 2c and 2d reveal that traditional MR and AMP are also relatively ineffective at measuring memory performance as pipeline issue width increases. Only the comprehensive memory metric C-AMAT, which considers hit and

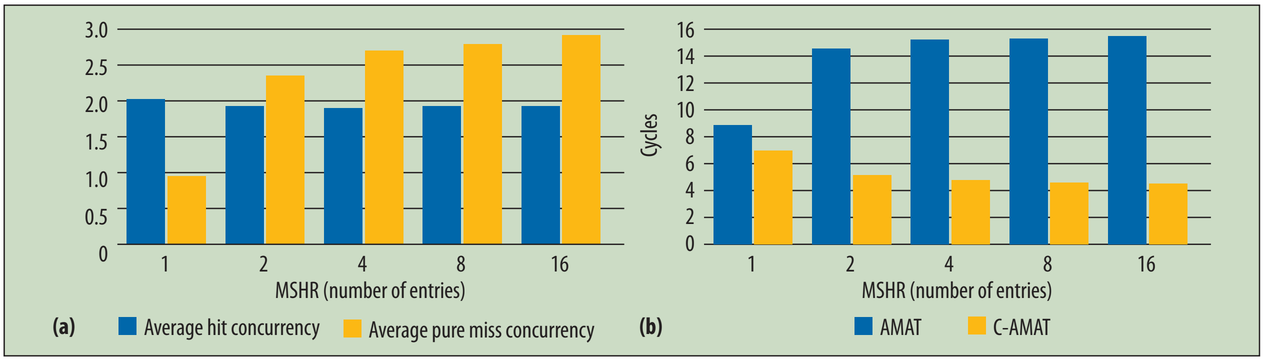

Figure 3. Memory performance measured as a function of miss status holding register (MSHR) entries adopted in nonblocking cache technologies. (a) As the number of MSHR entries increases, average hit concurrency remains the same, while average miss concurrency increases, resulting in better memory performance. (b) C-AMAT reflects this smaller miss concurrency gain, showing a decrease, while AMAT inaccurately increases, suggesting a decrease in memory performance.

miss access delay, proportion, and concurrency, can correctly reflect overall memory performance. (Admittedly, the differences reflected in Figures 2c and 2d are very small—between MR and pure MR the average difference is 1.2 percent, while between AMP and pure AMP it is 7.1 percent; the finer accuracy of C-AMAT is attributable mainly to the concurrency of memory accesses—a reasonable measure, given that changing pipeline issue width changes concurrency.)

Miss concurrency results based on MSHR size in nonblocking cache

To accommodate new processor microarchitectures based on out-of-order execution and speculation as well as multiple-issue, multithread, and multicore technologies, modern CPUs like the Intel Core and Itanium and IBM's Power series employ nonblocking cache heavily at each level of a memory hierarchy in order to enhance memory access parallelism. Nonblocking caches can continue supplying data under a certain number of cache misses by adopting a miss status holding register (MSHR).[4]

The MSHR is a structured table that records cache miss information such as access type (load/store), access address, return register, and so forth. When the MSHR table is empty, no outstanding cache misses remain. When the MSHR is attached to the last-level cache (LLC) and empty, no outstanding main memory accesses remain. When the MSHR table is full, the cache cannot afford more cache accesses, which blocks the CPU's memory accesses or next-level memory accesses. Therefore, the number of MSHR entries can directly determine miss access concurrency.

However, even though MSHR table size can directly determine maximum miss concurrency, it does not have a direct impact on hit concurrency, as Figure 3 shows. In Figure 3a, for example, as the number of MSHR entries increases, the average hit concurrency remains approximately the same, but the average pure miss concurrency increases steadily. C-AMAT reflects this reality that the larger the MSHR table, the smaller the miss concurrency gain; as Figure 3b demonstrates, when the number of MSHR entries increases, C-AMAT decreases. Under the same circumstances, however, AMAT increases, which inaccurately describes a memory performance decrease. Only C-AMAT accurately portrays the performance improvement resulting from the wide adoption of nonblocking cache techniques in modern processor design; AMAT fails completely in describing concurrency variations involved with MSHR.

Hit and miss concurrency results based on multicore technologies

Faced with a looming power wall resulting from limited ILP technology gains, researchers see an increase in the number of processor cores per die as the most effective and efficient means to increase total system performance. In multicore processors, different cores share an LLC; this greater number of cores obviously increases LLC hit and miss concurrency, which AMAT cannot measure correctly but C-AMAT can measure accurately, as Figure 4 demonstrates.

We assume that L2 is the LLC. Positing an increase in the number of cores by a factor of 2 ($1 \rightarrow 2 \rightarrow 4 \rightarrow 8$), L2 cache size increases accordingly (512 KB $\rightarrow$ 1 MB $\rightarrow$ 2 MB $\rightarrow$ 4 MB), as do L2 cache concurrencies, as shown in Figure 4a. Miss rates also increase because of the increased number of conflicts (Figure 4b). Figure 4c shows AMAT's failure as a metric when concurrency is a factor: the AMAT metric suggests a decrease for overall L2 cache memory performance as the number of cores increases, but this is obviously wrong. Using C-AMAT for the metric here, however, shows an appropriate increase in L2 cache memory performance.

While AMAT is widely used as a tool for architecture analysis and design, it does not accurately take into account access concurrency, an increasingly vital factor in measuring memory performance. C-AMAT extends AMAT to consider data access concurrency in

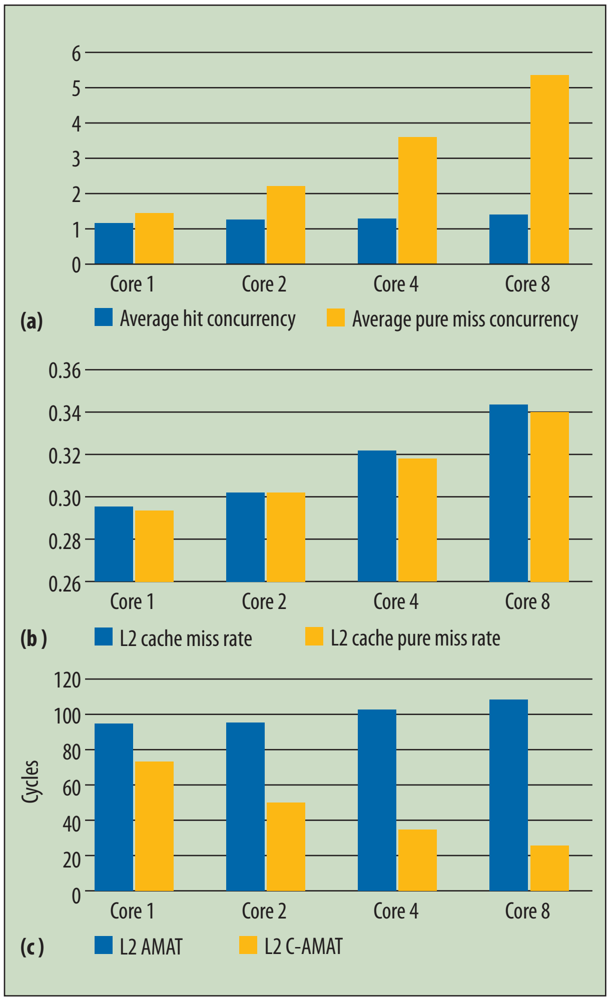

Figure 4. Memory performance measured for multiple cores. (a) As the number of cores increases, LLC (here L2 cache) concurrencies also increase. (b) Miss rates increase accordingly. (c) AMAT increases with the number of cores, showing an inaccurate decrease in overall memory performance; C-AMAT decreases, which accurately reflects overall memory improvement.

modern memory systems. Extensive simulations confirm C-AMAT's effectiveness for evaluating modern memory system design and architecture configuration. When access concurrency remains unchanged, C-AMAT produces results identical to those of AMAT. However, when memory access improvement results from concurrence—whether through ILP or other advanced cache design techniques or as a result of multicore technologies—AMAT cannot correctly reflect these memory performance changes, and so provides misleading information. C-AMAT takes into account memory access improvements that result from concurrency.

AMAT's attraction as a metric is its simplicity. But it is designed to characterize memory hierarchy, not concurrency. C-AMAT offers a practical, accurate alternative for modern memory architecture analysis and design, both now and in the future.

Acknowledgments

This research was supported in part by the National Science Foundation under grants CCF-0621435, CNS-0751200, and CCF-0937877.

References

1. X.-H. Sun and L.M. Ni, "Another View on Parallel Speedup," *Proc. 1990 ACM/IEEE Conf. Supercomputing* (SC 90), 1990, pp. 324–333.
2. W.A. Wulf and S.A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, no. 1, 1995, pp. 20–24.
3. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed., Morgan Kaufmann, 2011.
4. D. Kroft, "Lockup-free Instruction Fetch/Prefetch Cache Organization," *Proc. 8th Ann. Symp. Computer Architecture* (ISCA 81), 1981, pp. 81–87.
5. X.-H. Sun and D. Wang, "APC: A Performance Metric of Memory Systems," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 40, no. 2, 2012, pp. 125–130.
6. H. de Vries, *Understanding the Detailed Architecture of AMD's 64-bit Core*, 21 Sept. 2003; http://chip-architect.com/news/2003_09_21_Detailed_Architecture_of_AMDs_64bit_Core.html.
7. N. Binkert et al., "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, no. 4, 2006, pp. 52–60.
8. C.D. Spradling, "SPEC CPU2006 Benchmark Tools," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 1, 2007, pp. 130–134.
9. D. Wang and X.-H. Sun, *Concurrent Average Memory Access Time*, tech. report IIT/CS-SCS-2012-05, Dept. Computer Science, Illinois Inst. Technology, 2012.

***Xian-He Sun** is a distinguished professor of computer science and chair of the Department of Computer Science at the Illinois Institute of Technology, where he also directs the Scalable Computing Software Laboratory. He is also a guest faculty member with the Argonne National Laboratory's Mathematics and Computer Science Division. Sun's research interests include parallel and distributed processing, and performance evaluation and optimization. He received a PhD in computer science from Michigan State University. Sun is an IEEE Fellow. Contact him at sun@iit.edu.*

***Dawei Wang** is an ASIC design engineer with Juniper Networks and was previously a postdoctoral researcher in the Scalable Computing Software Laboratory at the Illinois Institute of Technology. His research interests include computer architecture, large-scale interconnection networks, and architectural simulation and emulation. Wang received a PhD in computer science from the Institute of Computing Technology, Chinese Academy of Sciences. Contact him at david.albert.wang@gmail.com.*