Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

A GENERALIZED MODEL FOR MODERN HIERARCHICAL MEMORY SYSTEM

Hamed Najafi
Jason Liu

Knight Foundation School of Computing
and Information Sciences
Florida International University
11200 SW 8th St
Miami, FL 33199, USA

Xiaoyang Lu
Xian-He Sun

Department of Computer Science
Illinois Institute of Technology
10 W 31th St
Chicago, IL 60616, USA

ABSTRACT

Memory system is critical to architecture design which can significantly impact application performance. Concurrent Average Memory Access Time (C-AMAT) is a model for analyzing and optimizing memory system performance using a recursive definition of the memory access latency along the memory hierarchy. The original C-AMAT model, however, does not provide the necessary granularity and flexibility for handling modern memory architectures with heterogeneous memory technologies and diverse system topology. We propose to augment C-AMAT to take into consideration the idiosyncrasies of individual cache/memory components as well as their topological arrangement in the memory architecture design. Through trace-based simulation, we validate the augmented model and examine the memory system performance with insight unavailable using the original C-AMAT model.

1 INTRODUCTION

We have recently witnessed considerable growth in data-intensive applications, which call for more effective memory system design. With more cores and processors in modern architectures to better exploit potential parallelism in applications, we have also noticed increased data access latency due to the more intensified memory system traffic hindering the performance of the applications. Researchers have proposed various solutions to rectify this problem, including more sophisticated cache hierarchies with deeper cache levels, more flexible topologies, as well as embracing heterogeneous memory technologies (including DRAM, NVRAM, and disaggregated memory) to gain better performance.

Studying the performance of memory architecture is becoming ever more important as many of today's popular applications, including big data processing, machine learning, cloud computing, and high-performance computing applications, are inherently data-intensive. Memory performance models have been developed to predict application performance, analyze and optimize the memory system design. The Concurrent Average Memory Access Time (C-AMAT) is one such model that allows us to capture the average memory access latency for applications (Sun and Wang 2014).

The C-AMAT model provides a more accurate representation of the memory access time than the conventional Average Memory Access Time (AMAT), as it considers concurrency in the memory access through the memory hierarchy. In a modern memory architecture with out-of-order instruction execution, multiple cache misses can be served concurrently. A memory stall happens only when a CPU cycle consists of only pure misses; that is, when the CPU cannot execute an instruction when the memory system fails to provide the necessary data for computing for the cycle. A detailed description of the unified C-AMAT memory performance model is provided by Liu et al. (2021). The authors used a memory-centric approach to establish a concrete mathematical foundation for model-driven performance analysis and optimization

978-1-6654-7661-4/22/$31.00 ©2022 IEEE

of memory systems, potentially featuring deep and diverse memory system hierarchies, heterogeneous memory devices, and complex data-intensive applications.

While C-AMAT provides an accurate representation of the average memory access time of the applications, it has limitations. First, the C-AMAT model calculates the memory access time top down through the cache memory hierarchy; it does not consider the contribution of individual memory units if there are multiple of them at the same level. For example, in a multi-core design, there are multiple memory units at the level-two cache, one for each processor core.In this paper, we refer to a memory unit generally as either a cache device or a memory device. There are also multiple memory banks in the memory system. Second, the current C-AMAT model considers all memory units at the same level as one undivided unit. This would be problematic when the memory accesses do not necessarily traverse the same top-down path through the cache hierarchy; different traffic leads to different performance results. Third, C-AMAT assumes that all memory units within the same cache/memory level use the same type of device and therefore present the same performance response. Such an assumption is untrue in a heterogeneous memory architecture, where, for example, the memory system consists of both DRAM and NVRAM devices sharing the memory space. These memory units deliver different performance with respect to latency and throughput.

Unraveling the complexity of modern memory system design requires more specific consideration of individual memory units in the memory performance model. This paper presents a generalized C-AMAT model for modern memory systems that consist of heterogeneous cache memory devices. In particular, the augmented C-AMAT model considers the general topology of the hierarchical memory systems, specifically for memory accesses branching and merging at a specific memory unit. Merging happens when memory accesses from different processors or cores may arrive at the same cache memory unit. Branching happens when a cache miss at a memory unit is served by different lower-layer caches or memory in accordance with its memory address. The augmented model calculates the concurrent average memory access time at individual memory units according to the idiosyncrasies of individual cache/memory components as well as their topological arrangement in the memory architecture design. Consequently, with the increased granularity, the generalized model can produce more accurate results and allows us to better analyze potential performance issues in the memory system.

The rest of the paper is organized as follows. In Section 2, we present related work and provide a quick introduction to the original C-AMAT model. In Section 3, we present the generalized C-AMAT model. For simplicity, we also present a special case where only merging occurs in the memory system topology. In Section 4, we conduct a validation study of the generalized model using trace-driven simulation to evaluate the memory performance of the different applications with different cache/memory configurations. Finally, in Section 5 we summarize our contributions and outline future work.

2 RELATED WORK

Concurrent Average Memory Access Time (C-AMAT) is a memory performance metric that accounts for both locality, concurrency, and overlapping in a single mathematical formulation (Sun and Wang 2014). C-AMAT extends the conventional Average Memory Access Time (AMAT) model (Wulf and McKee 1995), which is developed based on a CPU-centric view where cache hits and cache misses are accounted for by individual memory accesses when the CPU is executing instructions. Although seemingly intuitive, AMAT does not explain the overlap of memory accesses that may occur in the modern memory system. C-AMAT extends AMAT and incorporates memory concurrency in the calculation of memory access time:

$$C\text{-}AMAT = \frac{H}{C_H} + \rho_M \cdot \frac{pAMP}{C_M},$$

where H is the hit time (or the duration of a hit memory access), $pAMP$ is the average pure miss penalty (i.e., the average number of pure miss cycles for each pure miss memory access), ρ_M is the pure miss ratio, C_H and C_M are the average hit and pure miss concurrency. In the formulation, we consider pure miss memory accesses because only pure miss cycles contribute to memory stalls.

C-AMAT can also be recursively applied across the memory hierarchy. The definition of C-AMAT at cache level l can be derived from C-AMAT at the layer immediate below, i.e., at cache level $l+1$:

$$C\text{-}AMAT(l) = \frac{\rho_m(l)}{\mu(l)} \cdot C\text{-}AMAT(l+1),$$

where $\rho_m(l)$ is the cache miss ratio and $\mu(l)$ is the ratio of miss cycles over memory active cycles at cache level l. For more detailed description, a memory-centric view of the C-AMAT model can be found in Liu et al. (2021). Note that in the original formulation, we do not differentiate the contribution of the memory access time for individual memory units. The augmented model proposed in this paper refines the model and considers individual cache memory components and their relationship with one another in the memory system topology.

The C-AMAT model considers layered performance, which can be used for performance analysis and optimization. Similarly, Access Per Cycle (APC) is another performance metric for evaluating modern memory architectures (Wang and Sun 2013). It measures the number of memory accesses per cycle, which can be obtained using existing hardware counters. APC is a reciprocal of C-AMAT. The Layered Performance Matching (LPM) is yet another performance model related to C-AMAT (Liu and Sun 2019). LPM deals with cache performance based on data flow across the memory hierarchy. The intuition behind LPM is to match the data request rate from a higher memory layer with the data supply rate from the memory layer below so that the memory system performance is able to keep pace with the demand from the application. In particular, the LPM method transfers a global performance optimization problem to a series of local optimization problems at individual layers of the memory hierarchy. The augmented C-AMAT model presented in this paper considers individual memory units and can be applied directly to APC and LPM as in the case of the original C-AMAT model.

The C-AMAT model has been extended to reduce measurement complexity or increase accuracy. For example, Yu et al. (2018) proposed an analytical model, called FC-AMAT, that evaluates the effectiveness of different C-AMAT factors to reduce hardware overhead for measurement. Yu et al. (2017) introduced another memory metric, called Blocked C-AMAT or BC-AMAT, that takes blocked cycles into consideration in order to achieve better accuracy in memory assessment. Although we do not handle these extensions in this paper, it can be expected that the idea behind our augmented C-AMAT model can be applied relatively easily to the aforementioned extended C-AMAT models.

3 GENERALIZED MODEL

In this section, we first introduce the memory system topology and define the parameters to describe branching and merging in the memory system. We then present the generalized C-AMAT model in the recursive form. For simple exposition, we describe a special case for merging.

3.1 Memory System Topology

Let $\boldsymbol{p} = 1, 2 \cdots \mathscr{P}$ be the processing elements (cores), where $\mathscr{P}$ is the total number of processing elements in the entire system. Let $\boldsymbol{k} = 1, 2, \cdots, \mathscr{K}$ be the memory units, where $\mathscr{K}$ is the total number of memory units in the system. A memory unit can be L1, L2, L3 cache, DRAM, and NVRAM (used as far memory or persistence memory). The memory units may belong to different cores, processors, or machines connected via the bus (e.g., between CPU and main memory) or over the network (such as disaggregated memory).

The memory units are organized as a multi-tree, where the memory units are the nodes. A multi-tree is a direct acyclic graph in which a set of nodes reachable from any node induces a tree (Wikipedia 2022). Let $\mathbb{R}$ be the set of root nodes of the multi-tree and let $\mathbb{L}$ be the set of leaf nodes. The first-level caches are the root nodes of the multi-tree. We assume that there is an one-to-one mapping between the processing elements and the first-level caches. Note that in modern CPUs, the first-level cache typically has separate caches for handling instructions and data. In this study, we focus only on data as they dominate the overall

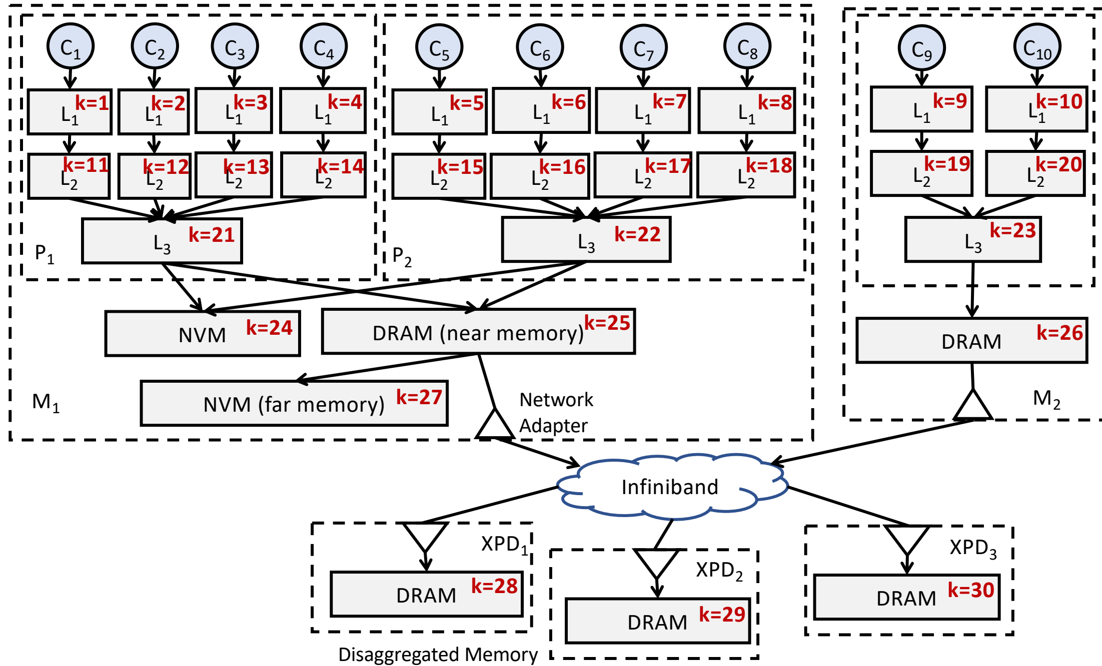

Figure 1: A memory system viewed as a multi-tree.

memory system performance. The leaf nodes are the memory units that eventually store the data. For convenience, we denote the first $\mathscr{P}$ memory units as the first-level caches corresponding to the processing elements. That is, $\mathbb{R} = \{1, 2 \cdots \mathscr{P}\}$.

A memory unit may be reachable by multiple processing elements. For example, an L3 cache may be shared by all cores within a processor; the main memory (DRAM) are shared by all cores in all processors on the same machine; a disaggregated memory device may be shared by all in the system. Similarly, multiple memory units may be reachable from a memory unit. Data is partitioned among leaf nodes; it is cached by interior nodes along the path to the corresponding processing element who accesses the data. A tree from a root node indicates the memory hierarchy with a unique path from the root that corresponds to the L1 cache of a processing element to a leaf node that corresponds to the final data location. When a processing element accesses the data (via read or write), the data is cached by the memory units along the path from the leaf node to the root node.

For any node k, let $\mathbb{P}(\boldsymbol{k})$ be the set of immediate predecessor nodes in the multi-tree (toward the direction of the root nodes). Let $\mathbb{S}(\boldsymbol{k})$ be the set of immediate successor nodes (in the direction away from the root nodes). In other words, $\mathbb{P}(k)$ and $\mathbb{S}(k)$ are the ingress and egress nodes of k in the multi-tree, respectively. Figure. 1 shows an example of a memory system with two machines and three disaggregated memory nodes. One machine has dual processors and each processor has four cores. The machine has three cache levels within each processor, a DRAM configured as the near memory, and two volatile memories, one as persistent memory and the other as the far memory. The other machine is a single-processor dual-core machine with three levels of cache and DRAM. Both machines are connected to three disaggregated memory nodes (Kove XPD) over an Infiniband interconnect. The memory system consists of 30 memory units in total. The first 10 memory units ($k = 1, 2 \cdots 10$) are first-level caches and they are the root nodes of the multi-tree. At node $k = 25$, for example, $\mathbb{P}(25) = \{21, 22\}$ and $\mathbb{S}(25) = \{27, 28, 29, 30\}$.

Let $\boldsymbol{\Lambda_k}$ be the set of addresses accessible by memory unit k. Since data is partitioned among the memory units at the leaf nodes, we have $\bigcap_{k \in \mathscr{L}} \Lambda_k = \phi$ and $\Lambda = \bigcup_{k \in \mathscr{L}} \Lambda_k$, where $\boldsymbol{\Lambda}$ is the entire memory space. For any memory unit j represented by an interior node, we have $\bigcap_{k \in \mathbb{S}(j)} \Lambda_k = \phi$, and $\Lambda_j = \bigcup_{k \in \mathbb{S}(j)} \Lambda_k$. In

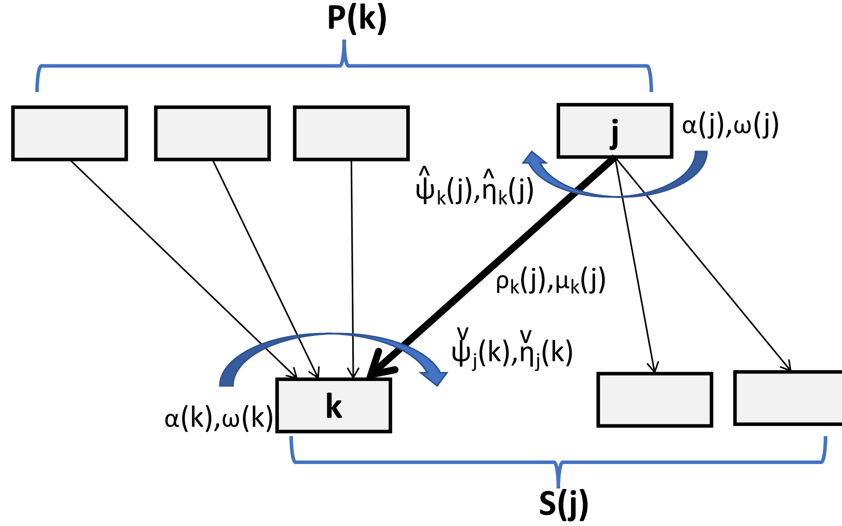

Figure 2: Branching and merging factors.

particular, $\Lambda_1, \Lambda_2 \cdots \Lambda_P$ are the memory spaces accessible by the corresponding processing elements. (We do not assume that all processing elements can access the entire memory space.) We say that a memory access at the memory unit j is *"targeted"* for the successor memory unit $k \in \mathbb{S}(j)$ if its address is in Λ_k.

3.2 Definitions for Branching and Merging

We first make some definitions for the generalized memory model described in the previous section. Figure. 2 shows the definition of parameters outlined in the rest of this section.

Let $\boldsymbol{\alpha(k)}$ be the number of memory accesses at memory unit k. If k is a first-level cache, that is, $k \in \mathbb{R}$, a memory access at k is directly issued from the corresponding processing element. Otherwise, a memory access at k is resulted from a miss at a predecessor memory unit of k. Let $\boldsymbol{\breve{\alpha}_j(k)}$ be the number of memory accesses arriving at memory unit k from memory unit j, where j is a predecessor memory unit of k; that is, $j \in \mathbb{P}(k)$. The total number of memory accesses at k is the sum of memory accesses from all its predecessor memory units: $\alpha(k) = \sum_{j \in \mathbb{P}(k)} \breve{\alpha}_j(k)$. Similarly, let $\boldsymbol{\hat{\alpha}_k(j)}$ be the number of memory accesses at memory unit j targeted for memory unit k, where k is a successor memory unit of j, that is, $k \in \mathbb{S}(j)$. The number of memory accesses at j is the sum of memory accesses targeted for all its successor memory units: $\alpha(j) = \sum_{k \in \mathbb{S}(j)} \hat{\alpha}_k(j)$. To be more specific, for any memory unit k and its predecessor memory unit j, i.e., $j \in \mathbb{P}(k)$, a memory access at k from j is resulted from a miss at j targeted for k. We have:

$$\breve{\alpha}_j(k) = \hat{\alpha}_k(j) \cdot \rho_k(j). \tag{1}$$

where $\boldsymbol{\rho_k(j)}$ is the miss ratio among all memory accesses at memory unit j targeted for memory unit k.

Let $\boldsymbol{\breve{\psi}_j(k)}$ be the ingress branching factor, defined as the proportion of memory accesses at memory unit k that come from memory unit j, where j is a predecessor memory unit of k, that is, $j \in \mathbb{P}(k)$:

$$\breve{\psi}_j(k) = \frac{\breve{\alpha}_j(k)}{\alpha(k)}. \tag{2}$$

We have $\sum_{j \in \mathbb{P}(k)} \breve{\psi}_j(k) = 1$. Similarly, we define $\boldsymbol{\hat{\psi}_k(j)}$ to be the egress branching factor for memory unit j targeted for memory unit k, where k is a successor memory unit of j, that is, $k \in \mathbb{S}(j)$:

$$\hat{\psi}_k(j) = \frac{\hat{\alpha}_k(j)}{\alpha(j)}. \tag{3}$$

Accordingly, $\sum_{k\in\mathbb{S}(j)} \hat{\psi}_k(j) = 1$. From the above definitions, we have:

$$\alpha(k) = \boxed{\frac{\breve{\alpha}_j(k)}{\breve{\psi}_j(k)}}_{(2)} = \frac{\boxed{\hat{\alpha}_k(j)\cdot\rho_k(j)}_{(1)}}{\breve{\psi}_j(k)} = \frac{\boxed{\alpha(j)\cdot\hat{\psi}_k(j)}_{(3)}\cdot\rho_k(j)}{\breve{\psi}_j(k)} = \frac{\hat{\psi}_k(j)\cdot\rho_k(j)}{\breve{\psi}_j(k)}\cdot\alpha(j). \quad (4)$$

The average duration of a memory access at memory unit k can be divided into two parts: the *hit time* for a memory access (regardless whether it's a hit or a miss) to "visit" the memory unit, and the *average miss penalty* to account for the additional time it takes for a miss memory access to fetch data from a successor memory unit. Note that only a miss memory access entails the second part. Memory accesses can overlap in time at a memory unit. Accordingly, the CPU cycles for a program's run time at any memory unit can be divided into four types: (1) *memory inactive cycles*, where no memory access occurs during these cycles, (2) *pure hit cycles*, which contain only the hit part of overlapped memory accesses, (3) *pure miss cycles*, which contain only the miss part of overlapped memory accesses, and (4) *mixed hit/miss cycles*, where both hit and miss parts of memory accesses overlap. Note that pure hit cycles, pure miss cycles, and mixed hit/miss cycles all contain outstanding memory accesses. Therefore, they are collectively called memory active cycles.

Let $\boldsymbol{\omega(k)}$ be the number of memory active cycles at memory unit k. To differentiate memory accesses from different source, let $\boldsymbol{\breve{\omega}_j(k)}$ be the number of memory active cycles at memory unit k from memory unit j, where j is a predecessor memory unit of k, that is, $j \in \mathbb{P}(k)$. Note that the definition of $\breve{\omega}_j(k)$ considers only the memory accesses from j. That is, the pure hit cycles, the pure miss cycles, and the mixed hit/miss cycles in this case are defined only for memory accesses from memory unit j. The classification of memory active cycles may be different among memory accesses from different predecessor memory units and therefore may be different from the overall classification (without considering the source of the memory accesses).

We define the ingress merging factor, $\boldsymbol{\breve{\eta}_j(k)}$, to be the proportion of the memory active cycles from j over all memory active cycles at k. That is,

$$\breve{\eta}_j(k) = \frac{\breve{\omega}_j(k)}{\omega(k)}. \quad (5)$$

The value of $\breve{\eta}_j(k)$ should be no larger than 1, since the total number of memory active cycles at k has to be between two extreme situations—where the memory accesses from all predecessor memory units overlap and where no overlap happens between the memory accesses from the predecessor memory units: $\max_{j\in\mathbb{P}(k)}\{\breve{\omega}_j(k)\} \leq \omega(k) \leq \sum_{j\in\mathbb{P}(k)} \breve{\omega}_j(k)$. The merging factor at a memory unit indicates the degree of overlap among the memory accesses from the predecessor memory units. It is closer to 1 when more overlap happens.

Similarly, let $\boldsymbol{\hat{\omega}_k(j)}$ be the number of memory active cycles at memory unit j targeted for memory unit k, where k is a successor memory unit of j, that is, $k \in \mathbb{S}(j)$. Here, the definition of $\hat{\omega}_k(j)$ considers only the memory accesses targeted for k. The pure hit cycles, the pure miss cycles, and the mixed hit/miss cycles in this case are defined only for memory accesses targeted for memory unit k. They may be different for different successor memory units and from the overall case (without considering the target of the memory accesses). Accordingly, we define the egress merging factor, $\boldsymbol{\hat{\eta}_k(j)}$, to be the proportion of the memory active cycles targeted for k over all memory active cycles at j. That is,

$$\hat{\eta}_k(j) = \frac{\hat{\omega}_k(j)}{\omega(j)}. \quad (6)$$

As with the ingress merging factor, the value of $\hat{\eta}_k(j)$ should be no larger than 1, which can be used to indicate the degree of overlap among the memory accesses targeted for the successor memory units. It is closer to 1 when more overlap happens.

As indicated earlier, for any memory unit k and its predecessor memory unit j, i.e., $j \in \mathbb{P}(k)$, a memory access at k from j is resulted from a miss at j targeted for k. Consequently, the memory active cycles at k from j coincide with the pure miss and mixed hit/miss cycles at j targeted for k. Let $\boldsymbol{\mu_k(j)}$ be the ratio of all miss cycles (including both pure miss cycles and mixed hit/miss cycles) over all memory active cycles at memory unit j targeted for successor memory unit k. We have:

$$\breve{\omega}_j(k) = \hat{\omega}_k(j) \cdot \mu_k(j). \tag{7}$$

From the above definitions, we have:

$$\omega(k) \quad = \quad \boxed{\frac{\breve{\omega}_j(k)}{\breve{\eta}_j(k)}}_{(5)} = \frac{\boxed{\hat{\omega}_k(j) \cdot \mu_k(j)}_{(7)}}{\breve{\eta}_j(k)} = \frac{\boxed{\omega(j) \cdot \hat{\eta}_k(j)}_{(6)} \cdot \mu_k(j)}{\breve{\eta}_j(k)} = \frac{\hat{\eta}_k(j) \cdot \mu_k(j)}{\breve{\eta}_j(k)} \cdot \omega(j). \tag{8}$$

3.3 Generalized Recursive C-AMAT for Branching and Merging

According to Sun and Wang (2014), we define ***C-AMAT(k)***, the concurrent average memory access time to be the number of memory active cycles divided by the number of memory accesses at memory unit k:

$$C\text{-}AMAT(k) = \frac{\omega(k)}{\alpha(k)}. \tag{9}$$

Theorem 1 For any memory units j and k, where j is a predecessor of k, i.e., $j \in \mathbb{P}(k)$, we have:

$$C\text{-}AMAT(j) = \frac{\hat{\psi}_k(j) \cdot \breve{\eta}_j(k) \cdot \rho_k(j)}{\breve{\psi}_j(k) \cdot \hat{\eta}_k(j) \cdot \mu_k(j)} \cdot C\text{-}AMAT(k).$$

This theorem can be easily derived from Equations (9), (4), and (8). □

The reciprocal of $C\text{-}AMAT(k)$ is ***APC(k)***, the accesses per cycle, which is the average number of memory accesses per memory active cycle at memory unit k: $APC(k) = C\text{-}AMAT(k)^{-1} = \frac{\alpha(k)}{\omega(k)}$. We have the following corollary:

Corollary 1.1 For any memory units j and k, where j is a predecessor of k, i.e., $j \in \mathbb{P}(k)$, we have:

$$APC(j) = \frac{\breve{\psi}_j(k) \cdot \hat{\eta}_k(j) \cdot \mu_k(j)}{\hat{\psi}_k(j) \cdot \breve{\eta}_j(k) \cdot \rho_k(j)} \cdot APC(k).$$

3.4 A Simplified Case for Merging

The previous generalized model can deal with arbitrary memory system topology. For simplicity, we deal with a special situation where a memory unit handles a confluence of memory accesses originated from multiple processor cores. A merging situation occurs at a memory unit k when $|\mathbb{P}(k)| \geq 1$.

While the generalized model described in the previous section can easily handle this special case, we can further simplify the model by introducing an assumption. In the generalized model, we define $\rho_k(j)$ to be the miss ratio among all memory accesses at memory unit j targeted for memory unit k. In practice, it would be rather expensive to keep track of the individual miss ratio for each subsequent memory unit. To make it easier, one can instead use the overall miss ratio, $\boldsymbol{\rho_m(j)}$, for all memory accesses at memory unit j. This is to assume that the cache behavior of memory accesses at memory unit j is similar across the partitioned memory space among its successors.

Since we need not consider branching in this case, that is, the memory unit k is the only immediate successor node for the memory unit j (although k may have more than one immediate predecessor nodes, including j). Therefore, the egress branching factor for memory unit j targeted for the successor memory

.

Table 1: Configuration of the Simulated System.

Processor	4 cores, 4GHz, 8-issue width, 256-entry ROB
L1 Cache	private, split 32KB I/D-cache/core, 64B line, 8-way, 4-cycle latency, 8-entry MSHR
L2 Cache	private, 256KB/core, 64B line. 8-way, 10-cycle latency, 32-entry MSHR
L3 Cache	shared, 2MB/core, 64B line, 16-way, 20-cycle latency, 64-entry MSHR
DRAM	4GB/8GB/16GB, 64-bit channel, 2400MT/s, tRP=15ns, tRCD=15ns, tCAS=12.5ns
NVM	4GB/8GB/16GB, 64-bit channel, 1600MT/s, tRP=22ns, tRCD=22ns, tCAS=15ns

unit k, $\hat{\psi}_k(j) = 1$. Similarly, the egress merging factor, which is defined to be the proportion of the memory active cycles targeted for k over all memory active cycles at j, $\hat{\eta}_k(j) = 1$.

Considering the merge-only scenario, we can express C-AMAT recursively for the memory hierarchy (i.e., on a path from any memory unit to a root node of the multi-tree) using the following theorem.

Theorem 2 (Merging Case Only) For any memory units j and k, where j is a predecessor of k, i.e., $j \in \mathbb{P}(k)$, we have:

$$C\text{-}AMAT(j) = \frac{\rho_m(j) \cdot \breve{\eta}_j(k)}{\mu(j) \cdot \breve{\psi}_j(k)} \cdot C\text{-}AMAT(k).$$

The theorem can be derived from Theorem 1 by replacing $\rho_k(j)$ with $\rho_m(j)$ and setting both $\hat{\psi}_k(j)$ and $\hat{\eta}_k(j)$ to 1. □

4 EXPERIMENTS

In this section, we describe the experiments we have conducted to validate the generalized memory model. We implemented the model using the ChampSim simulator (Kim 2017). Unlike the timing simulators, ChampSim does not natively run an executable/assembly program but uses an instruction trace extracted from an application to capture the representative phases of workloads. ChampSim is a cycle-accurate simulator that adopts a detailed out-of-sequence CPU model to achieve accurate simulation results.

We conducted experiments of a four-core system. We model a per-core private L1 cache, a per-core private L2 cache, and a shared LLC cache. We also extend the ChampSim simulator to model the performance characteristics of our memory system, which closely matches the JEDEC Non-volatile Dual In-line Memory Module (NVDIMM) specifications (Lalam and Shen 2019). The default configurations are shown in Table 1. We statically allocate pages of both core 0 and core 1 to DRAM, and pages of both core 2 and core 3 to NVM.

The simulations are conducted with 8 workloads from SPEC CPU2006 suite (Spradling 2007) and SPEC CPU2017 suite. We select 4 memory-intensive workloads, which have at least 1 LLC miss per kilo instructions (MPKI). We also select 4 CPU-intensive workloads that have less than 1 LLC MPKI. Table 2 shows LLC MPKI for the experiment workloads in our study. For SPEC workloads, the traces are collected with SimPoint (Perelman et al. 2003). We use mixed workloads to simulate a multi-programmed system. For a 4-core mixed workload, we select 4 different benchmarks from 8 SPEC benchmarks and run one trace in each core. Each trace is warmed up with 10M instructions, and simulation results are collected over the next 50M instructions. For each mixed workload, if a benchmark finishes early, it is replayed until each benchmark has finished running 50M instructions. We generate 3 mixed workloads. We select 4 different memory-intensive benchmarks to compose MIX-1. For MIX-2, we replace the benchmark assigned to core 0 in MIX-1 with a CPU-intensive one and keep the rest the same as MIX-1. For MIX-3, we replace the benchmarks assigned to cores 1, 2, and 3 in MIX-1 with different CPU-intensive benchmarks, and the benchmark assigned to core 0 remains the same as MIX-1.

Table 2: Experiment Workloads.

	Workload	MPKI	Workload	MPKI	Workload	MPKI	Workload	MPKI
Memory-intensive	403.gcc	25.55	429.mcf-22B	26.28	623.xalancbmk	24.26	654.roms	24.23
CPU-intensive	400.perlbench	0.09	435.gromacs	0.06	444.namd	0.06	641.leela	0.03
MIX-1: 403, 429, 623, 654; MIX-2: 400, 429, 623, 654; MIX-3: 403, 435, 444, 641								

Table 3: IPC with varying mem size (MIX-1).

D/N Size	Core 0	Core 1	Core 2	Core 3
4/4GB	0.2517	0.2718	0.1602	0.2983
8/8GB	0.2603	0.2914	0.1688	0.3187
16/16GB	0.2660	0.3055	0.1763	0.3371

Table 4: C-AMAT with varying mem size (MIX-1).

D/N Size	Core 0	Core 1	Core 2	Core 3
4/4GB	67.6121	130.9021	158.9072	170.1020
8/8GB	63.3539	121.7010	147.0258	159.2506
16/16GB	60.2883	114.8905	137.7589	150.9292

Table 5: IPC for MIX-1 and MIX-2.

D/N Size	Core 0	Core 1	Core 2	Core 3
4/4GB	0.2517	0.2718	0.1602	0.2983
4/4GB	0.5629	0.3564	0.1601	0.3010
8/8GB	0.2603	0.2914	0.1688	0.3187
8/8GB	0.5635	0.3722	0.1687	0.3229
16/16GB	0.2660	0.3055	0.1763	0.3371
16/16GB	0.5639	0.3865	0.1755	0.3387

Table 6: Pure misses for MIX-1 and MIX-2.

D/N Size	Core 0	Core 1	Core 2	Core 3
4/4GB	1281991	1199660	1226297	1257263
4/4GB	4613	1169636	1225475	1235936
8/8GB	1282252	1200073	1226633	1257411
8/8GB	4616	1168373	1225627	1235441
16/16GB	1282378	1201410	1226743	1257026
16/16GB	4618	1167864	1225834	1235709

4.1 Impact of Main Memory Configuration

The main memory configuration will affect the data access latency and change the data access concurrency as well. Table 3 shows the IPC (Instructions Per Cycle) value of each core when we have 4/8/16 GB DRAM and NVM. We observe that for a mixed workload with 4 memory-intensive benchmarks (MIX-1), the IPC value of each core increases with the increase of the DRAM and NVM size. Table 4 shows the changes in LLC C-AMAT values with different DRAM/NVM sizes. We observe that with larger DRAM/NVM, each memory-intensive application has a lower concurrent average memory access time in LLC. When the size of the main memory increases, the LLC C-AMAT values decrease due to the decrease of interference in the main memory. This explains why IPCs are better with larger memory in Table 3.

4.2 Impact of Memory-intensive vs. CPU-intensive Applications

Memory-intensive applications have a considerable number of miss accesses in LLC. As a result, memory-intensive applications have worse performance in terms of IPC. Tables 5 and 6 show the difference when we change the application running on core 0 from a memory-intensive application to a CPU-intensive one, which generates fewer miss accesses at LLC. We observe that IPC for core 0 grows, and LLC pure miss drops significantly. Even though we only change the application on CPU 0, we see IPC grows at CPU 1 as well. It is because core 0 has fewer accesses in DRAM, which leaves core 1 with less congestion at DRAM, resulting in a better IPC. IPC in CPU 2 and CPU 3 does not change as much because they are using a different memory device (NVM). Moreover, the CPU-intensive application causes less LLC interference; core 1 benefits a lot from it in this case. When the size of both DRAM and NVM is 4GB; in MIX-2, core 1 eliminates 2.5% LLC pure misses compared with MIX-1, which improves 31.11% IPC. We also observe a similar trend as DRAM and NVM increase in size. In these two tables, the odd rows belong to MIX-1 and the even rows belong to MIX-2.

Table 7: IPC for MIX-1 and MIX-3.

D/N Size	Core 0	Core 1	Core 2	Core 3
4/4GB	0.2517	0.2718	0.1602	0.2983
4/4GB	0.2840	0.9647	1.3086	0.3855

Table 8: LLC Performance for MIX-1 and MIX-3.

D/N Size	LLC Pure Misses	LLC C-AMAT
4/4GB	1281991	67.6121
4/4GB	1234424	42.6608

4.3 Impact of Interference

We further examine the interference. We switch the benchmarks assigned to cores 1, 2, and 3 in MIX-1 from memory-intensive benchmarks to CPU-intensive benchmarks (MIX-3). In Table 7 we observe that, although the application assigned to core 0 does not change in MIX-3 compared to MIX-1, the IPC of core 0 increases significantly in MIX-3. This can be justified if we take a look at LLC pure miss at core 0 in Table 8. We observe that core 0's LLC pure misses are reduced due to less interference from other cores. Furthermore, fewer LLC pure misses on core 0 cause lower LLC C-AMAT value. When we change the benchmarks assigned to core 1, core 2, and core 3 to CPU-intensive applications, there is less LLC interference leading to a reduction in core 0's pure misses and C-AMAT and improvement in core 0's IPC. In Table 8 we also notice a considerable drop in LLC's C-AMAT that proves our point. In these two tables, first row belongs to MIX-1 and the second row belongs to MIX-3.

5 CONCLUSION

We propose an augmented C-AMAT model that considers the idiosyncrasies of cache memory devices and the interconnecting topology of the memory system. In doing so, the generalized model allows us to examine the performance of the individual cache memory devices and be able to analyze their contributions to the overall performance of the memory system design. We conduct validation experiments using trace-driven simulations, which confirm that the augmented C-AMAT model matches the expected performance for applications running on a multi-core architecture. The augmented C-AMAT model is expected to provide better understanding and more insight into the memory system performance and its impact on applications.

For future work, we plan to apply this model to examine the impact of recent architectures, including GPU and disaggregated memory, on different types of real applications, including machine learning and data processing applications. We also plan to seek the possibility of extending this model for studying cache policies (including cache bypassing) and other system resource scheduling decisions.

REFERENCES

Jinchun Kim 2017. "The Champsim Simulator". https://github.com/ChampSim/ChampSim. Last accessed on 12nd July 2022.

Lalam, A., and A. C. Shen. 2019, February 5. "Non-volatile Dual In-line Memory Module (NVDIMM) Multichip Package". Google Patents. US Patent 10,199,364.

Liu, J., P. Espina, and X.-H. Sun. 2021. "A Study on Modeling and Optimization of Memory Systems". *Journal of Computer Science and Technology* 36(1):71–89.

Liu, Y., and X. Sun. 2019, Nov. "LPM: A Systematic Methodology for Concurrent Data Access Pattern Optimization from a Matching Perspective". *IEEE Transactions on Parallel and Distributed Systems* 30(11):2478–2493.

Perelman, E., G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder. 2003. "Using SimPoint for Accurate and Efficient Simulation". *ACM SIGMETRICS Performance Evaluation Review* 31(1):318–319.

Spradling, C. D. 2007. "SPEC CPU2006 Benchmark Tools". *ACM SIGARCH Computer Architecture News* 35(1):130–134.

Sun, X., and D. Wang. 2014, May. "Concurrent Average Memory Access Time". *Computer* 47(05):74–80.

Wang, D., and X.-H. Sun. 2013. "APC: A Novel Memory Metric and Measurement Methodology for Modern Memory Systems". *IEEE Transactions on Computers* 63(7):1626–1639.

Wikipedia 2022. "Multitree". https://en.wikipedia.org/wiki/Multitree, Last accessed 15$^{\text{nd}}$ July 2022.

Wulf, W. A., and S. A. McKee. 1995. "Hitting the Memory Wall: Implications of the Obvious". *ACM SIGARCH computer architecture news* 23(1):20–24.

Yu, Q., L. Huang, C. Qian, J. Ma, and Z. Wang. 2018. "FC-AMAT: Factor-based C-AMAT Analysis in Memory System Measurement". *Innovations in Systems and Software Engineering* 14(2):143–156.

Yu, Q., L. Huang, C. Qian, and Z. Wang. 2017. "BC-AMAT: Considering Blocked Time in Memory System Measurement". In *Proceedings of the Computing Frontiers Conference*, 230–236.

ACKNOWLEDGMENTS

The authors would like thank the anonymous reviewers for their constructive comments. This project is partially supported by the National Science Foundation Awards CCF-2008000 and CCF-2008907, and the Department of Homeland Security grant 2017-ST-062-000002.

AUTHOR BIOGRAPHIES

HAMED NAJAFI is a Ph.D. student at the Knight Foundation School of Computing and Information Sciences at Florida International University. His research interests include artificial intelligence, computer memory architecture design, and performance model and simulation. His email address is hnaja002@fiu.edu.

XIAOYANG LU is a Ph.D. student at the Department of Computer Science, Illinois Institute of Technology (IIT). His research interests include cache/memory performance optimizations, AI-assisted design for computer architecture, and data-centric design for computer architecture. His email address is xlu40@hawk.iit.edu.

JASON LIU is a Professor at the Knight Foundation School of Computing and Information Sciences, Florida International University. His research focuses on parallel simulation and high-performance modeling of computer systems and communication networks. His email address is liux@cis.fiu.edu.

XIAN-HE SUN is a Professor in the Department of Computer Science at Illinois Institute of Technology (IIT). His research interests include data-intensive high-performance computing, memory and I/O systems, software system for big data applications, and performance evaluation and optimization. His email address is sun@iit.edu.