

Estimating TLB Misses in BLIS-like GEMM

Girlana Souza¹, Lucas Wanner¹, Isaías Felzmann¹

¹Institute of Computing – Universidade Estadual de Campinas (UNICAMP)

g295743@dac.unicamp.br, {wanner, ibf}@unicamp.br

Abstract. *Virtual memory introduces address-translation overhead that can affect the performance of highly optimized kernels such as General Matrix Multiplication (GEMM). While GEMM implementations such as BLIS employ hierarchical blocking and data packing to improve data locality, characterizing their impact on Translation Lookaside Buffer (TLB) behavior typically requires empirical hardware measurements. This work proposes an analytical model to estimate L1 and L2 TLB misses in BLIS-like GEMM implementations without requiring execution profiling. The model captures the matrix memory layout, loop hierarchy, and TLB configuration to derive virtual-page footprints, translation working sets, and reuse distances for both packed and unpacked BLIS-like GEMM variants. Experimental validation using hardware performance counters across varying workloads and page sizes (4 KiB and 2 MiB) shows that the model consistently captures the overall trends in translation pressure and provides an optimistic estimate of the absolute number of misses. Despite its structural simplifications, the model characterizes how GEMM configurations and packing strategies affect TLB behavior, including the trade-off between increased translation demand for small working sets and improved L1 TLB spatial locality for larger workloads. These results provide an analytical basis for studying translation behavior in GEMM and for guiding optimization decisions without requiring runtime performance-counter measurements.*

1. Introduction

Virtual memory is fundamental to modern computer systems, but address translation introduces an additional cost to every memory access. The Translation Lookaside Buffer (TLB) mitigates this overhead by caching recently used virtual-to-physical address translations. However, when the active set of pages exceeds TLB reach or when access patterns exhibit large strides, translation misses require accesses to lower TLB levels or page-table walks, potentially affecting execution performance.

This effect is particularly relevant for General Matrix Multiplication (GEMM), whose optimized implementations rely heavily on data locality. Libraries and frameworks such as GotoBLAS and BLIS employ hierarchical blocking and data packing to improve memory access behavior [Goto and van de Geijn 2008, Van Zee and Van De Geijn 2015]. Besides their effect on the cache hierarchy, these transformations change the number,

This work was supported by Instituto de Pesquisas Eldorado; by the National Council for Scientific and Technological Development (CNPq, in part by grant #306314/2025-8); by the Coordination for the Improvement of Higher Education Personnel (CAPES) - Finance Code 001; by the Brazilian Ministry of Science, Technology and Innovation (MCTI), with resources from Law No. 8,248, of October 23, 1991, under process no. 01245.002675/2024-69; and by FAEPEX-UNICAMP grants #3770/25 and #2622/25.

organization, and reuse pattern of virtual pages accessed during execution. Packing, for example, replaces strided accesses with compact buffers, but also introduces additional memory accesses required to copy matrix panels.

Although TLB coverage has been considered in the design of high-performance GEMM implementations, existing approaches mainly use it as a constraint for blocking or packing decisions. They do not explicitly characterize how the BLIS loop hierarchy, matrix layout, blocking parameters, and packing operations translate into misses across multiple TLB levels. Consequently, evaluating translation pressure generally requires executing the kernel and collecting hardware performance counters.

This work proposes an analytical model for estimating TLB misses in BLIS-like GEMM implementations. Given the matrix algorithm structure, memory layout, page size, and TLB configuration, the model derives the virtual-page footprint of matrix regions, constructs translation working sets, and uses reuse distances to estimate translation residency. The formulation considers both packed and unpacked variants and distinguishes compulsory and reuse misses propagated through the L1 and L2 TLB hierarchy.

The main contributions of this work are an analytical formulation of the page footprints and translation working sets for packed and unpacked BLIS-like GEMM; a reuse-distance model that captures translation reuse across the GEMM loop structure; an analytical estimation of L1 and L2 TLB misses that can be used to characterize TLB behavior across GEMM configurations; and an experimental validation using hardware performance counters across configurations with different levels of TLB pressure.

2. Background

This section introduces the concepts underlying our address-translation model: TLBs, data locality and storage layouts, and GEMM tiling and packing.

2.1. Translation Lookaside Buffer

The *Translation Lookaside Buffer* (TLB) is a small cache that stores recent virtual-to-physical address translations. It reduces the overhead of virtual memory, a technique that provides each process with its own address space, independently of the available physical memory [Denning 1970]. TLB entries are typically organized as fully or set-associative structures with only from tens of entries (L1) to a few thousand (L2), which makes a TLB lookup fast enough to complete within a single processor cycle [Hennessy et al. 2012].

As part of the *Memory Management Unit* (MMU), the TLB accelerates address translation. When a memory access occurs, the MMU extracts a Virtual Page Number (VPN) and queries the TLB to find the corresponding Physical Page Number (PPN). If the mapping is cached (a TLB hit), it is combined with the page offset to form the physical address. Otherwise (a TLB miss), the MMU must perform a costly *page-table walk* through the operating system’s page tables. Once the mapping is located, the new translation is loaded into the TLB, and the memory access proceeds [Hennessy et al. 2012].

Modern TLBs are typically structured in a two-level hierarchy. The L1 TLB is smaller and faster, often split into separate instruction and data TLBs, whereas the unified L2 TLB is larger and provides greater translation reach. An L1 miss triggers an L2 lookup, and a page-table walk occurs only if both levels miss [Agredo et al. 2025]. Be-

cause traversing the page table requires multiple slow memory accesses, maintaining a high TLB hit rate is critical for overall system performance.

2.2. Locality

Both caches and TLBs rely on spatial and temporal locality to minimize memory access latency [Denning 2005]. For matrix operations, locality depends on the data layout, which determines how elements of a two-dimensional matrix are mapped to the linear memory address space. In row-major order, elements of each row are stored contiguously, whereas in column-major order, elements of each column are stored contiguously.

Matrix accesses can follow either row-wise or column-wise traversal. Spatial locality is maximized when the access pattern matches the storage layout, such as row-wise traversal for row-major matrices and column-wise traversal for column-major matrices. Conversely, a mismatch between traversal and storage order produces strided accesses, potentially increasing cache and TLB miss rates.

2.3. General Matrix Multiplication

General Matrix Multiplication (GEMM) is a core subroutine in numerical computing and machine learning, defined as $C = \alpha AB + \beta C$, where A , B , and C are dense matrices, and α and β are floating-point scalar coefficients [Dongarra et al. 1990]. GEMM is highly optimized in linear algebra libraries such as OpenBLAS [OpenMathLib 2026].

To reduce memory access costs, GEMM implementations use hierarchical tiling (or blocking), which partitions matrices into blocks that fit in different levels of the cache hierarchy and can be reused. This organization typically uses large *macroblocks* sized for the L2 and L3 caches and smaller register blocks processed by a *microkernel*.

Data packing complements tiling by converting strided accesses into contiguous ones. It copies panels or macroblocks of A and B into buffers arranged in the layout expected by the microkernel [Zhang et al. 2025]. This improves cache-line utilization and can reduce conflict misses and TLB pressure by limiting the number of active memory pages during inner-loop computation [Salvador Rohwedder et al. 2023].

3. Related Work

This section reviews prior work on GEMM optimization, memory hierarchy management, and cache and address-translation modeling.

3.1. GEMM Optimizations

Goto and van de Geijn [Goto and van de Geijn 2008] show that high-performance GEMM must account for both cache capacity and TLB coverage. Since submatrices may span non-contiguous pages, their approach combines hierarchical blocking with packing panels into contiguous buffers, reducing translation pressure and improving data locality.

BLIS [Van Zee and Van De Geijn 2015] extends this organization through portable macro-kernels and architecture-specific micro-kernels. Panels of A and B are packed into contiguous buffers (A_c and B_c), while (m_c, n_c, k_c) define cache-level blocking and (m_r, n_r) the register-level micro-kernel. Together, these parameters determine both cache reuse and TLB pressure, while avoiding non-unit strides during the inner computation.

Our work adopts the blocking and packing organization of GotoBLAS and BLIS, but uses it to model address-translation behavior. Rather than using TLB coverage only as a blocking constraint, the proposed model estimates page footprints, reuse distances, and L1/L2 TLB misses for both packed and unpacked BLIS-like GEMM organizations.

3.2. Models for Parameter Selection

Low et al. [Low et al. 2016] present an analytical model for determining the blocking parameters of the GEMM operation in BLIS without relying on empirical searches. By exploiting the modular structure of BLIS, their model relates these blocking parameters to architectural characteristics. The authors validate the derived values across different architectures and show that they achieve performance close to that of manually tuned implementations. These results support the use of an analytical formulation in this work.

Whereas the model by Low et al. selects BLIS blocking parameters primarily based on cache and architectural constraints, our approach takes the GEMM configuration as input and estimates its TLB behavior. Thus, the focus shifts from parameter selection for cache efficiency to predicting translation pressure across the TLB hierarchy.

3.3. Packing Analysis

GPAT, proposed by Salvador Rohwedder et al. [Salvador Rohwedder et al. 2023], is a packing transformation for affine loop nests. It filters candidates based on data reuse and cache hierarchy, retaining those that reduce access stride or TLB pressure. GPAT estimates the number of TLB entries required with and without packing and considers packing beneficial when the demand fits within the L1 dTLB capacity. Candidate selection uses the reduction in TLB entries as the benefit and the packed-buffer footprint as the cost.

Rather than using cache and TLB criteria to dynamically select packing transformations as GPAT does, our work evaluates predefined packed and unpacked BLIS-like GEMM variants. We model their page footprints, reuse distances, and resulting TLB misses, including the additional translation demand introduced by packing.

3.4. Address translation and TLB

McKellar and Coffman [McKellar and Coffman Jr 1969] developed analytical models for the paging cost of matrix algorithms, relating memory organization and operation ordering to page faults. More recently, Basu et al. [Basu et al. 2013] quantified address-translation overhead in big-memory workloads using hardware performance counters, relating TLB-miss overhead to limited TLB reach.

Unlike McKellar and Coffman, who model page faults and transfers, our approach assumes resident pages and focuses on translation misses within the TLB hierarchy. In contrast to Basu et al., who quantify translation overhead experimentally, our model predicts L1 and L2 TLB misses analytically and uses hardware counters only for validation.

4. An Analytical Model for TLB Performance Estimation

The model estimates, for each TLB level, the number of data-TLB misses produced by a BLIS-like GEMM. It proceeds in five steps, each described in one subsection: it (i) maps matrix regions to virtual pages and computes their page footprints (Section 4.3); (ii) combines these footprints along the loop hierarchy into translation working sets (Section

4.4); (iii) derives from the working sets the reuse distance of each class of translation reuse (Section 4.5); (iv) decides whether each reuse hits by comparing its distance with the TLB capacity (Section 4.6); and (v) adds compulsory and reuse misses and propagates them from L1 to L2 (Section 4.7). Table 1 summarizes the notation.

Table 1. Main notation used by the model.

Symbol	Meaning
M, N, K	Dimensions of A ($M \times K$), B ($K \times N$), C ($M \times N$)
m_c, n_c, k_c	Cache-level blocking parameters
m_r, n_r	Micro-tile dimensions
i, j, p	Macro-block indices along M, N, K
I, J, Q	Numbers of macro-blocks along M, N, K
u, v	Micro-tile indices along M, N
U_i, V_j	Numbers of micro-tiles of macro-block (i, j) along M, N
M_i, N_j, K_p	Effective dimensions of macro-block (i, j, p)
P	Page size (bytes)
b_X, ld_X, s_X	Base address, leading dimension, element size of region X
δ_X	Offset of packed buffer X within its first page
E_l	Number of entries of TLB level l
$F_X(r, c)$	Page footprint of an $r \times c$ tile of X
f_A, f_B, f_C	Footprints of the micro-panels used by one micro-kernel call
W_x	Working set of one complete execution of loop x
$\bar{\Delta}_X^{(x)}$	Average new pages of X per iteration of loop x
$D_X^{(x)}$	Reuse distance of X across iterations of loop x
$D_{\min}, D_{\max}$	Minimum and maximum reuse distance of a reuse class
$H(D, E_l)$	Hit indicator for reuse distance D at TLB level l
$\rho_{c,l}$	Residency of reuse class c at TLB level l
N_{ref}	Total number of page-level translation references
N_{new}	Number of compulsory misses (distinct pages touched)
R_c	Number of reuses in class c
Q_c	Reuses of class c that miss at L1 and reach L2
M_l	Estimated misses at TLB level l

4.1. Inputs and Assumptions

The model evaluates two GEMM variants, $v \in \{\text{unpacked}, \text{packed}\}$. Both compute $C+ = A \cdot B$ (i.e., GEMM with $\alpha = \beta = 1$) with $A \in \mathbb{R}^{M \times K}$, $B \in \mathbb{R}^{K \times N}$ and $C \in \mathbb{R}^{M \times N}$, all stored in row-major order. The unpacked variant accesses the original matrices directly, whereas the packed variant copies panels of A and B into contiguous buffers A_c and B_c . Each variant is described by $\theta_v = (\theta_{\text{gemm}}, \theta_{\text{mem}}, \theta_{\text{tlb}})$, where:

- $\theta_{\text{gemm}} = (M, N, K, m_c, n_c, k_c, m_r, n_r)$, contains the matrix dimensions, the cache-level blocking parameters, and the micro-tile dimensions.
- $\theta_{\text{mem}} = (P, \{b_X, ld_X, s_X\}_{X \in \mathcal{D}_v})$ contains the page size P and, for each memory region X , its base virtual address b_X , leading dimension ld_X , and element size s_X in bytes. The regions are $\mathcal{D}_{\text{unpacked}} = \{A, B, C\}$ and $\mathcal{D}_{\text{packed}} = \{A, B, C, A_c, B_c\}$.

- $\theta_{\text{tlb}} = \{E_l\}_{l=1}^L$ contains the number of entries E_l of each of the L TLB levels.

The model represents each TLB level as a fully associative structure with E_l entries. Given θ_v , the model returns $\{M_l\}_{l=1}^L$, the estimated number of misses at each TLB level.

Assumptions: The model assumes single-threaded execution, fixed-size pages, and an initially empty TLB. It considers only the data accesses of the GEMM operation, excluding instruction fetches, operating-system activity, and interference from other processes. TLB level $l+1$ is accessed only after a miss at level l , and distinct memory regions occupy disjoint virtual pages.

4.2. GEMM Execution Phases

The model follows the BLIS loop order [Van Zee and Van De Geijn 2015], shown below with matrix elements indexed as $X[\text{row}, \text{column}]$:

```

for jc = 0 .. N-1 step nc:           // index j
  for pc = 0 .. K-1 step kc:         // index p
    pack B[pc:pc+Kp, jc:jc+Nj] into Bc // packed variant only
    for ic = 0 .. M-1 step mc:       // index i
      pack A[ic:ic+Mi, pc:pc+Kp] into Ac // packed variant only
      for jr = 0 .. Nj-1 step nr:    // index v
        for ir = 0 .. Mi-1 step mr:  // index u
          microkernel(A[ic+ir, pc], B[pc, jc+jr], C[ic+ir, jc+jr])

```

In the packed variant, the micro-kernel reads A_c and B_c instead of A and B . The numbers of macro-blocks along M , N and K are

$$I = \lceil M/m_c \rceil, \quad J = \lceil N/n_c \rceil, \quad Q = \lceil K/k_c \rceil,$$

indexed by $0 \leq i < I$, $0 \leq j < J$ and $0 \leq p < Q$. The matrix sizes need not be multiples of the blocking parameters, so the effective dimensions of macro-block (i, j, p) are:

$$M_i = \min(m_c, M - im_c), \quad N_j = \min(n_c, N - jn_c), \quad K_p = \min(k_c, K - pk_c).$$

Macro-block (i, j, p) operates on three regions: $A_{ip}(M_i \times K_p)$, $B_{pj}(K_p \times N_j)$ and $C_{ij}(M_i \times N_j)$. Its output block is divided into $U_i \times V_j$ micro-tiles, with $U_i = \lceil M_i/m_r \rceil$ and $V_j = \lceil N_j/n_r \rceil$, indexed by $0 \leq u < U_i$ and $0 \leq v < V_j$.

4.3. Page Mapping and Page Footprints

For a region X , element (r, c) has virtual address $\text{addr}_X(r, c) = b_X + s_X(r \cdot ld_X + c)$ and lies in virtual page $\lfloor \text{addr}_X(r, c)/P \rfloor$. The page footprint $F_X(r, c)$ of an $r \times c$ tile of X is the number of distinct virtual pages that the tile occupies.

Unpacked tiles A tile starting at (i_0, j_0) consists of r rows, and each row is a contiguous segment of $c s_X$ bytes. Row t ($0 \leq t < r$) starts at $a_t = b_X + s_X((i_0 + t)ld_X + j_0)$ and spans pages S_t to E_t .

$$S_t = \left\lfloor \frac{a_t}{P} \right\rfloor, \quad E_t = \left\lfloor \frac{a_t + c s_X - 1}{P} \right\rfloor.$$

Because a_t increases with t , the footprint can be obtained by counting only the pages that each row adds to the previous one: For the first segment: $\boxed{\eta_0 = E_0 - S_0 + 1}$. For

each subsequent segment $t > 0$: $\eta_t = \max(0, E_t - \max(S_t, E_{t-1} + 1) + 1)$. The page footprint of the unpacked tile is:

$$F_X(r, c) = \sum_{t=0}^{r-1} \eta_t.$$

Packed buffers Packing removes the stride between rows, so the $r \times c$ elements form a single contiguous block. Let $\delta_X = b_X \bmod P$ be the offset of the buffer within its first page. Then:

$$F_X(r, c) = \left\lceil \frac{\delta_X + r \cdot c \cdot s_X}{P} \right\rceil, \quad X \in \{A_c, B_c\}.$$

Unlike the unpacked footprint, this value does not depend on ld_X .

Micro-panel footprints One micro-kernel call accesses an $m_r \times K_p$ panel of A , a $K_p \times n_r$ panel of B , and an $m_r \times n_r$ tile of C . Their footprints are denoted f_A , f_B and f_C . In the packed variant f_A and f_B are computed over A_c and B_c .

4.4. Translation Working Sets

The *translation working set* of an execution phase is the number of distinct virtual pages accessed by its active regions. Since regions are disjoint, it equals the sum of their footprints. Let W_x denote the working set of one complete execution of loop $x \in \{u, v, i, p\}$ for fixed values of the enclosing indices:

$$\begin{aligned} W_u &= F_A(M_i, K_p) + f_B + F_C(M_i, n_r), \\ W_v &= F_A(M_i, K_p) + F_B(K_p, N_j) + F_C(M_i, N_j), \\ W_i &= F_A(M, K_p) + F_B(K_p, N_j) + F_C(M, N_j), \\ W_p &= F_A(M, K) + F_B(K, N_j) + F_C(M, N_j). \end{aligned}$$

In the packed variant, the A and B terms of W_u and W_v refer to A_c and B_c , while W_i and W_p additionally include the footprints of A_c and B_c and of the original regions read during packing. The total number of distinct pages touched by the computation is:

$$W_{tot} = F_A(M, K) + F_B(K, N) + F_C(M, N),$$

plus the footprints of A_c and B_c in the packed variant.

4.5. Reuse Distances

The *reuse distance* of a translation is the number of distinct virtual pages accessed between two consecutive references to it. The model groups reuses into five structural classes, plus a spatial class described in Section 4.7, each named after the reused operand and the loop that carries the reuse. For example, $A^{(v)}$ denotes the reuse of an A micro-panel across consecutive iterations of loop v .

Between two such references, the other operands advance and introduce new pages. The model approximates this growth by an average increment. If the footprint of X operand grows from F_{first} to F_{full} over n iterations of loop x , then

$$\bar{\Delta}_X^{(x)} = \frac{F_{full} - F_{first}}{n - 1} \quad (n > 1), \quad \bar{\Delta}_X^{(x)} = 0 \quad (n \leq 1).$$

Table 2 gives the distance of each class. For $B^{(u)}$, the distance counts the pages of the A and C micro-tiles and the other pages of the same B micro-panel. For the other classes, it combines the working set of the inner loops with the average increments of the other operands.

Table 2. Reuse classes and their reuse distances.

Class	Reuse loop	Reuse distance
$B^{(u)}$	u	$D_B^{(u)} = f_A + f_C + \max(0, f_B - 1)$
$A^{(v)}$	v	$D_A^{(v)} = W_u - f_A + \bar{\Delta}_B^{(v)} + \bar{\Delta}_C^{(v)}$
$B^{(i)}$	i	$D_B^{(i)} = W_v - f_B + \bar{\Delta}_A^{(i)} + \bar{\Delta}_C^{(i)}$
$C^{(p)}$	p	$D_C^{(p)} = W_i - f_C + \bar{\Delta}_A^{(p)} + \bar{\Delta}_B^{(p)}$
$A^{(j)}$	j	$D_A^{(j)} = W_p - f_A + \bar{\Delta}_B^{(j)} + \bar{\Delta}_C^{(j)}$

4.6. TLB Residency

Each TLB level is approximated as a fully associative structure with E_l entries and LRU (*Least Recently Used*) replacement. Under this abstraction, a reused translation hits if and only if its reuse distance is smaller than the capacity:

$$H(D, E_l) = \begin{cases} 1, & D < E_l, \\ 0, & D \geq E_l. \end{cases}$$

Working sets and increments are computed from representative regions and averages, so the distance of a class varies across iterations. For each class, the model therefore computes a minimum $D_{\min}$ and a maximum $D_{\max}$, the latter being the value in Table 2. For most classes, $D_{\min}$ is obtained by summing the average number of new pages that the other operands bring into the TLB at each iteration of the loop that carries the reuse: for class $X^{(x)}$, $D_{\min} = \sum_{Y \neq X} \bar{\Delta}_Y^{(x)}$. For $B^{(u)}$, which is reused within the micro-kernel loop, and for $A^{(j)}$, whose reuse spans a complete execution of the inner loop chain, all pages in the distance are necessarily accessed between two uses, so $D_{\min} = D_{\max}$. In the packed variant, the same holds for $C^{(p)}$.

The estimator uses $D_{\min}$, which corresponds to the smallest possible interference between accesses. Together with full associativity, this assumption ignores set-conflict effects and favors TLB residency, thereby biasing the estimate toward fewer TLB misses. The residency of class c at level l is $\rho_{c,l} = H(D_{\min,c}, E_l)$.

4.7. TLB Miss Estimation

The model separates translation misses into two components: *compulsory misses* from the first access to a virtual page, and *reuse misses* when a previously accessed translation has been evicted. Set-conflict misses are not modeled. Instead, each TLB level is approximated as fully associative with the same number of entries as the corresponding hardware TLB, yielding an optimistic estimate that abstracts away set-placement effects.

Compulsory Misses Starting from an empty TLB, the first access to each of the W_{tot} pages misses at every level: $N_{\text{new}} = W_{\text{tot}}$.

Reuse (Capacity) misses Let N_{ref} be the total number of page-level translation references issued by the micro-kernel, $N_{ref} = J Q I V U \cdot W_\mu$, where $W_\mu = f_A + f_B + f_C$ is the number of pages referenced by one micro-kernel call, $U = \lceil m_c / m_r \rceil$, and $V = \lceil n_c / n_r \rceil$. Since the first reference to each page is a compulsory miss, the number of reuses is $N_{reuse} = \max(0, N_{ref} - N_{new})$.

These reuses are distributed among the five structural classes $\mathcal{C} = \{B^{(u)}, A^{(v)}, B^{(i)}, C^{(p)}, A^{(j)}\}$. The number of occurrences R_c of class c is the product of the enclosing trip counts, the carrying loop's trip count minus one, the inner trip counts that index the reused operand, and the operand's footprint. The remaining R_{sp} reuses form a spatial class: they are consecutive references to the same page and always hit.

In the packed variant, three adjustments apply. First, N_{ref} also includes the page references of the packing routines, $J Q I (F_A(M_i, K_p) + F_{A_c}(M_i, K_p)) + J Q (F_B(K_p, N_j) + F_{B_c}(K_p, N_j))$. Second, each R_c is capped by the number of distinct buffer pages touched between consecutive iterations of the carrying loop, and the remaining reuses form a local class with reuse distance W_μ instead of the spatial class. Third, $D_B^{(i)}$ uses $F_A(M_i, K_p)$, the A block read by the packing routine, instead of $\bar{\Delta}_A^{(i)}$.

With $\mathcal{C}$ extended by the local class in the packed variant, the L1 misses are:

$$M_1^{reuse} = \sum_{c \in \mathcal{C}} R_c (1 - \rho_{c,1}), \quad M_1 = N_{new} + M_1^{reuse}. \quad (1)$$

Propagation to L2 Only references that miss at L1 reach L2. For each class, the demand that reaches L2 is $Q_c = R_c (1 - \rho_{c,1})$, and

$$M_2^{reuse} = \sum_{c \in \mathcal{C}} Q_c (1 - \rho_{c,2}), \quad M_2 = N_{new} + M_2^{reuse}. \quad (2)$$

The spatial class does not contribute to L2, since it always hits at L1.

5. Experimental Evaluation and Validation

We validate the model by comparing its L1 and L2 TLB miss estimates with hardware performance counters, assessing whether it exhibits the expected optimistic behavior and whether it captures trends in translation pressure as the GEMM configuration varies.

5.1. Methodology and Test Environment

Experiments run single-threaded on an AMD Ryzen 7 5700. Matrices use row-major, double-precision (8 B) elements backed by 4 KiB or 2 MiB pages. The processor features a 64-entry fully-associative L1 dTLB and a 2048-entry L2 dTLB, 6-way associative for 4 KiB pages and 4-way associative for 2 MiB pages. Hardware events are measured in user space using Linux `perf_event_open` [Linux man-pages project 2026] (event=0x45). To compare hardware counters (M_1^{hw} , M_2^{hw}) against our model predictions (M_1 , M_2), we track the following for both page sizes:

- **L1 miss / L2 hit** (umask=0x01 for 4 KiB, 0x04 for 2 MiB): corresponds to the analytical quantity $(M_1 - M_2)$.
- **L1 miss / L2 miss** (umask=0x10 for 4 KiB, 0x40 for 2 MiB): corresponds to M_2 .

Reported values are medians of ten runs. Measurements exclude allocation and initialization, but include the packing overhead for the packed variant. We evaluate 15 scenarios by varying the matrix dimension N and tile size. All matrices are square ($M = N = K$), and the tile size sets the cache-level blocking parameters ($m_c = n_c = k_c$), with $m_r = 8$ and $n_r = 8$. In Scenario 1, the micro-tile is clipped to the 4×4 block. The model uses $E_1 = 64$ and $E_2 = 2048$ entries for both page sizes. 2 MiB pages are allocated through mmap with the MAP_HUGETLB | MAP_HUGE_2MB flags, backed by a pre-reserved HugeTLB pool (vm.nr_hugepages).

5.2. Results and Analysis

Table 3 summarizes the fifteen experimental scenarios, defined by the matrix size N , tile size, and page size. These configurations represent different memory hierarchy stress levels and optimizations: L1 capacity limits (Scenarios 1–4), tile size effects (Scenarios 5–7), L2 saturation (Scenarios 8–9), high translation pressure (Scenarios 10–11), and 2 MiB huge-page behavior (Scenarios 12–15).

Table 3. Experimental scenarios and corresponding parameters.

Id	Scenario	N	Tile	Page	Id	Scenario	N	Tile	Page
1	tiny	8	4	4 KiB	9	l2_above	608	32	4 KiB
2	l1_fit	64	16	4 KiB	10	pressure16	1024	16	4 KiB
3	l1_near	96	16	4 KiB	11	pressure64	1024	64	4 KiB
4	l1_over	128	16	4 KiB	12	huge_tiny	1024	256	2 MiB
5	tile16	256	16	4 KiB	13	huge_2048	2048	256	2 MiB
6	tile32	256	32	4 KiB	14	huge_2560	2560	256	2 MiB
7	tile64	256	64	4 KiB	15	huge_4096	4096	512	2 MiB
8	l2_below	576	32	4 KiB					

To evaluate the model’s performance and limitations, we compare its predicted L1 (M_1) and L2 (M_2) TLB misses with hardware measurements across different workload sizes, as shown in Figure 1. In Scenarios 1–4, the model predictions remain close to the hardware measurements while the workload fits within or approaches the L1 TLB capacity. When this capacity is exceeded, as in Scenario 4, the model captures the increase in translation pressure but underestimates its magnitude, partly because its fully associative LRU abstraction does not account for set-conflict effects.

Scenarios 5–7 show that the model captures the effect of tile size. For the unpacked variant, hardware measurements show a reduction in M1 misses when the tile size increases from 16 to 32, followed by an increase at 64. The model reproduces this trend and identifies Scenario 6 as the configuration with the lowest translation pressure. For the packed variant, M1 misses decrease monotonically with the tile size in both the measurements and the model estimates.

The largest differences occur in Scenarios 8–11, where the working set places high pressure on the L2 TLB. The model captures the overall increase in translation demand, but its error becomes larger. For example, in Scenario 8 (unpacked), the model predicts 883.2K M1 misses, compared with 5.3M measured in hardware. Scenarios 8 and 9 show particularly large differences as they approach the L2 TLB capacity, making them more sensitive to external interference. As the working set grows, set conflicts

and interference between reuse classes become more important, but these effects are not captured by the model.

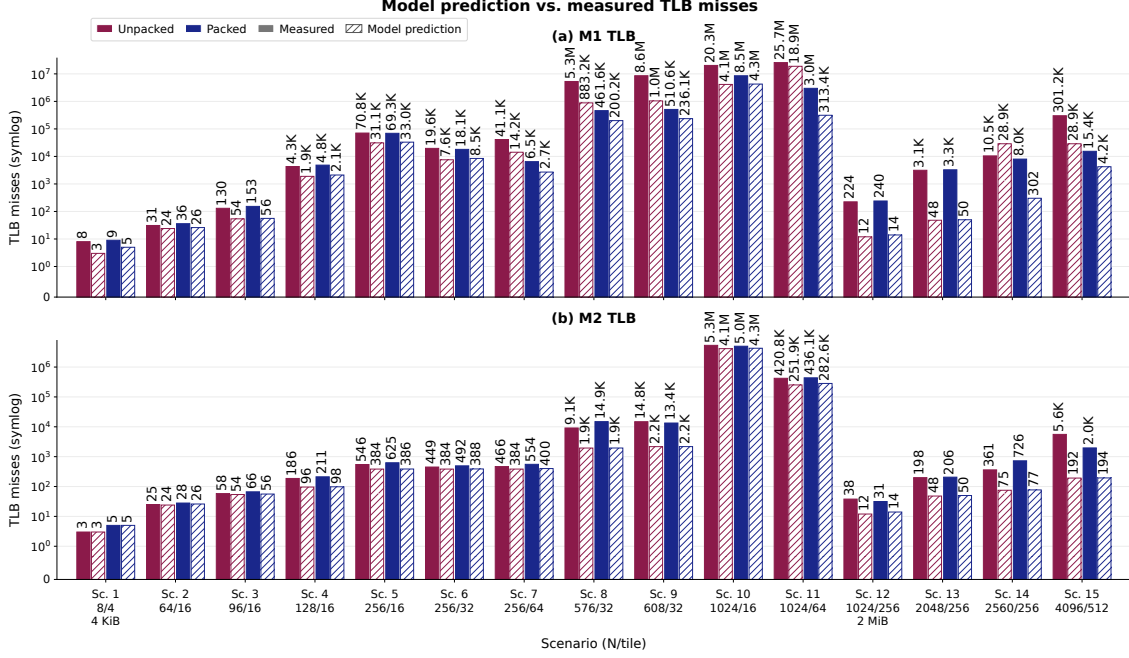


Figure 1. Comparison of analytical model estimations and hardware-measured TLB misses across all scenarios.

Scenarios 12–15, using 2 MiB pages, show that huge pages reduce TLB pressure, as fewer translations cover the working set. As a result, GEMM-related misses become small, making external interference more significant in the hardware measurements. In huge_tiny and huge_2048, short reuse distances keep translations inside the L1 TLB. In huge_2560, exceeding this capacity causes an abrupt change in the model estimate due to its binary residency rule. This transition also produces the only case in which the model overestimates the measured misses (Scenario 14, unpacked). Overall, these scenarios show that the model captures the reduction in translation pressure provided by huge pages, but becomes less accurate when this pressure is low due to external interference.

The results show that the model is useful for qualitative and comparative analysis of TLB behavior. The comparison also exposes the trade-off introduced by packing. For small workloads (Scenarios 1–4), the packed variant incurs slightly more M1 misses than the unpacked one (e.g., 153 vs. 130 in Scenario 3), since the buffers A_c and B_c add pages to the translation working set. As the workload grows, packing substantially reduces M1 misses (461.6K vs. 5.3M in Scenario 8 and 3.0M vs. 25.7M in Scenario 11), because the micro-kernel accesses contiguous buffers instead of strided rows. The model reproduces both effects. The model also captures the effects of page footprint, reuse, working-set growth, tile size, and page size, while its accuracy decreases as TLB pressure increases. This behavior is consistent with its optimistic formulation and the exclusion of set conflicts, external interference, and other effects related to TLB associativity and packing.

6. Limitations

The analytical model is intentionally optimistic, generally underestimating absolute TLB misses due to the following structural simplifications:

- **Linear footprint deltas:** The model assumes linear footprint growth between iterations, but actual growth may be sublinear or superlinear due to page overlap and alignment. This approximation can propagate to demand and reuse estimates.
- **Limited reuse classes:** Representing reuse using only five classes does not fully capture intra-panel reuse, cross-reuse, and other dependencies between iterations.
- **Simplified TLB residency:** Residency is approximated in an essentially binary manner, classifying access as a hit or a miss based on a distance threshold.
- **No set-conflict modeling:** The model does not capture conflict misses caused by TLB set mapping and associativity.
- **No external interference:** The model considers only GEMM data accesses, excluding other sources that may affect TLB state and introduce additional misses.
- **Simplified packing interference:** The model adds the page references of the packing routines, but does not track how they evict individual A_c and B_c translations.

The model’s optimistic nature does not imply a strict lower bound for all TLB organizations and access sequences. A fully associative LRU TLB may incur more misses than a set-associative TLB due to differences in eviction behavior. Thus, full associativity serves as an idealized baseline that eliminates set-placement constraints and isolates the effects of page footprints and reuse distances. Despite these simplifications, the model captures key TLB behavioral trends and supports comparative analysis.

7. Conclusion

This work presented an analytical model for estimating address translation behavior in BLIS-like GEMM implementations. The model combines page footprints, working sets, and reuse distances to estimate TLB misses in packed and unpacked variants, providing a means to analyze translation-hierarchy pressure without relying exclusively on empirical hardware measurements.

Experimental validation using hardware counters showed that the model underestimates the absolute number of TLB misses in nearly all evaluated scenarios. This underestimation is primarily attributed to the idealized fully associative TLB assumption, the independent treatment of reuse classes, and the simplified treatment of packing-induced evictions. Despite these simplifications, the model captures the overall trends in translation pressure across the evaluated scenarios. In particular, it captures the trade-offs introduced by data packing: packing may increase translation demand for small working sets while substantially improving L1 TLB spatial locality as the workload grows.

These results demonstrate that the proposed analytical model can be used to analyze TLB pressure comparatively and qualitatively across different GEMM configurations, reducing the reliance on empirical hardware measurements during early-stage performance analysis. Future work will focus on refining the model by incorporating set-associativity, inter-class interference, and packing-induced evictions. A further goal is to translate the estimated TLB misses into execution costs, enabling the model to provide more direct performance predictions and to better guide GEMM optimization decisions.

References

- Agredo, C., Koranek, D. F., Schubert Kabban, C. M., Gutierrez Del Arroyo, J. A., Langehaug, T. J., and Graham, S. R. (2025). Exploring the translation lookaside buffer (tlb) for low-level task differentiation and classification. *IEEE Access*, 13:111199–111216.
- Basu, A., Gandhi, J., Chang, J., Hill, M. D., and Swift, M. M. (2013). Efficient virtual memory for big memory servers. *ACM SIGARCH Computer Architecture News*, 41(3):237–248.
- Denning, P. J. (1970). Virtual memory. *ACM Computing Surveys (CSUR)*, 2(3):153–189.
- Denning, P. J. (2005). The locality principle. *Commun. ACM*, 48(7):19–24.
- Dongarra, J. J., Du Croz, J., Hammarling, S., and Duff, I. S. (1990). A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):1–17.
- Goto, K. and van de Geijn, R. A. (2008). Anatomy of high-performance matrix multiplication. *ACM Transactions on Mathematical Software (TOMS)*, 34(3):1–25.
- Hennessy, J., Patterson, D., and Asanović, K. (2012). *Computer Architecture: A Quantitative Approach*. Computer Architecture: A Quantitative Approach. Kaufmann.
- Linux man-pages project (2026). `perf_event_open(2)` — linux manual page. https://man7.org/linux/man-pages/man2/perf_event_open.2.html. System Calls Manual. Version 6.18.
- Low, T. M., Igual, F. D., Smith, T. M., and Quintana-Orti, E. S. (2016). Analytical modeling is enough for high-performance blis. *ACM Trans. Math. Softw.*, 43(2).
- McKellar, A. and Coffman Jr, E. G. (1969). Organizing matrices and matrix operations for paged memory systems. *Communications of the ACM*, 12(3):153–165.
- OpenMathLib (2026). Github - openmathlib/openblas: Openblas is an optimized blas library based on gotoblas2 1.13 bsd version. <https://github.com/OpenMathLib/OpenBLAS>. Accessed: 30 jul. 2026.
- Salvador Rohwedder, C., Henderson, N., De Carvalho, J. P., Chen, Y., and Amaral, J. N. (2023). To pack or not to pack: A generalized packing analysis and transformation. In *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*, pages 14–27.
- Van Zee, F. G. and Van De Geijn, R. A. (2015). Blis: A framework for rapidly instantiating blas functionality. *ACM Transactions on Mathematical Software (TOMS)*, 41(3):1–33.
- Zhang, J., Yang, W., Fang, J., Dong, D., and Chen, X. (2025). Flashgemm: Optimizing sequences of matrix multiplication by exploiting data reuse on cpus. *ACM Transactions on Architecture and Code Optimization*, 22(4):1–25.