The Combinatorics of Cache Misses during Matrix Multiplication*

Philip J. Hanlon† Dean Chung‡ Siddhartha Chatterjee§ Daniela Genius§
Alvin R. Lebeck¶ Erin Parker §

March 16, 2000

AMS(MOS) subject classifications: 68P20

*This work was supported in part by DARPA Grant DABT63-98-1-0001, NSF Grants EIA-97-26370 and CDA-95-12356, NSF Career Award MIP-97-02547, The University of North Carolina at Chapel Hill, Duke University, and an equipment donation through Intel Corporation's Technology for Education 2000 Program. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

†Department of Mathematics, University of Michigan, Ann Arbor, MI

‡Amazon.com, Inc., Seattle, Washington

§Department of Computer Science, The University of North Carolina, Chapel Hill, NC

¶Department of Computer Science, Duke University, Durham, NC

Abstract

In this paper we construct an analytic model of cache misses during matrix multiplication. The analysis in this paper applies to square matrices of size 2^m where the array layout function is given in terms of a function Θ that interleaves the bits in the binary expansions of the row and column indices. We first analyze the number of cache misses for direct-mapped caches and then indicate how to extend this analysis to $\mathbb{A}$-way associative caches.

The work in this paper accomplishes two things. First, we construct fast algorithms to estimate the number of cache misses. Second, we develop theoretical understanding of cache misses that will allow us, in subsequent work, to approach the problem of minimizing cache misses by appropriately choosing the bit interleaving function that goes into the array layout function.

1 Introduction

As the gap between processor cycle time and main memory access time continues to widen, effective use of the memory hierarchy becomes ever more critical to overall program performance. Caches can help alleviate the CPU-memory gap by satisfying most memory references at close to processor speed (1 to 3 cycles). Unfortunately, programs that do not exhibit good memory reference locality cannot exploit the potential benefits of caches.

For scientific computations that repeatedly access large data sets, good locality of reference is essential at the algorithm level for high performance. Such locality can either be *temporal*, in which a single data item is reused repeatedly, or *spatial*, in which a group of data items "adjacent" in space are used in temporal proximity. High-performance dense linear algebra codes rely on good spatial and temporal locality of reference for their performance. In this paper, we focus on an analysis of matrix multiplication, the workhorse of modern linear algebraic algorithms.

Our previous studies demonstrated an intimate relationship between the layout of the arrays in memory and the performance of the routine [1, 2]. This early work experimentally showed the benefits of using array layout functions based on interleaving the bits in the binary expansions of the row and column indices of arrays. This paper complements our earlier empirical studies by providing an analytical framework for analyzing the cache behavior of matrix multiplication in the presence of such array layout functions. Future work will use this framework in an optimization context, to determine array layouts that minimize the number of cache misses.

The remainder of this section provides the background of the cache analysis problem. Section 1.1 provides a brief overview of cache memory basics. Section 1.2 describes our analysis framework—both the similarities to earlier work and the critical differences that require us to use completely different techniques. Section 1.3 discusses array layout functions based on bit interleaving. Section 1.4 reiterates the goals of our analysis and provides a roadmap of the remainder of the paper.

1.1 Basics of cache memory

We assume a simplified memory hierarchy that processes one memory access at a time, with no distinction between memory reads and writes.

The structure of a single level of a memory hierarchy—called a *cache*—is generally characterized by three parameters: **A**ssociativity, **B**lock size, and **C**apacity. Capacity and block size are in units of the minimum memory access size (usually one byte). A cache can hold a maximum of C bytes. However, due to physical constraints, the cache is divided into *cache frames* of size B that contain B contiguous bytes of memory—called a *memory block*. The associativity A specifies the number of different frames in which a memory block can reside. If a block can reside in any frame (*i.e.*, $A = \frac{C}{B}$), the cache is said to be *fully associative*; if $A = 1$, the cache is *direct-mapped*; otherwise, the cache is A-*way set associative*.

For a given memory access, the hardware inspects the cache to determine if the corresponding memory element is resident in the cache. This is accomplished by using an indexing function to locate the appropriate set of cache frames that may contain the memory block. If the memory block is resident, a *cache hit* is said to occur, and the cache satisfies the access after its *access latency*. If the memory block is not resident, a *cache miss* is said to occur.

From an architectural standpoint, cache misses fall into one of three classes [7].

- A *compulsory miss* is one that is caused by referencing a previously unreferenced memory block. Eliminating a compulsory miss requires prefetching the data, either by an explicit prefetch operation or by placing more data items in a single memory block.
- A reference that is not a compulsory miss but misses in a fully-associative cache with LRU replacement is classified as a *capacity miss*. Capacity misses are caused by referencing more memory blocks than can fit in the cache. Restructuring the program to re-use blocks while they are in cache can reduce capacity misses.
- A reference that hits in a fully-associative cache but misses in an A-way set-associative cache is classified as a *conflict miss* (or *interference miss*). A conflict miss to block X indicates that block X has been referenced in the recent past, since it is contained in the fully-associative cache, but at least A other memory blocks that map to the same cache set have been accessed since the last reference to block X. Eliminating conflict misses requires transforming the program to change either the memory allocation and/or layout of the two arrays (so that contemporaneous accesses do not compete for the same sets) or the manner in which the arrays are accessed.

  At the program source level, interference misses can be further subdivided based on whether the interfering blocks come from different parts of a single array, or from different arrays. The miss is called a *self-interference miss* in the former case and a *cross-interference miss* in the latter case [8].

1.2 An analysis framework

Our general model for counting cache misses follows the framework used in previous work [5], with one significant difference. We first explain the common framework, then highlight the key difference in our version of the problem that necessitates entirely new solution techniques.

The program fragment whose cache behavior we are trying to analyze is a perfectly nested normalized loop with d levels of nesting, numbered 1 through d from outermost to innermost. The lower and upper bounds of ι_j, the loop control variable (LCV) for loop j, are affine functions of the LCVs ι_1 through ι_{j-1}. The iteration space $\mathcal{I}$is the set of all valid combinations of LCV values that are within the bounds of the loop nest. The notation $\iota = [\ell_1, \ldots, \ell_{d-1}]^T$ denotes a generic point in the iteration space $\mathcal{I}$. The iteration space is also equipped with a total order $<$, which is the lexicographic ordering on ι. The order specifies the temporal order in which the iteration points in the iteration space are executed.

The loop accesses elements of arrays $A^{(1)}$ through $A^{(m)}$. Array variable $A^{(i)}$ has d_i dimensions, with n_j being the extent of the array in the jth dimension. The data index space $\mathcal{D}_i$ corresponding to array $A^{(i)}$ is the Cartesian product $[0, n_1 - 1] \times \cdots \times [0, n_{d_i} - 1]$.

The statements in the loop body make k references to array variables. We denote these references R_1 through R_k. A reference R_i has two components: N_i, the name of the array referenced (so that $N_i = A^{(j)}$ for $1 \leqslant j \leqslant m$); and F_i, the index expression of the reference, which identifies the coordinates of the array element accessed by this reference at iteration point ι. The index expression F_i is constrained to be an affine function of ι in each of its components. Thus, F_i is a function from the iteration space $\mathcal{I}$to the data index space $\mathcal{D}_{N_i}$. We also assume that R_i is the ith array reference made at iteration point ι.

Array A_i has an associated layout function $\mathcal{L}_i$, which is a 1-1 map from $\mathcal{D}_i$ to the memory address space $\mathbb{Z}_0^+$. Applying this map to an element of an array produces the byte address of that array element.

We assume a two-level memory hierarchy, with a direct-mapped cache with block size of B bytes and total capacity of C bytes (and therefore $p = C/B$ sets). The quantities B and C are always powers of two for technological reasons, so we will assume that $p = 2^\rho$. We also assume that main memory is large enough to hold all the data referenced by the program. The function $\mathcal{B}$converts a memory byte address address into a memory block address (with $\mathcal{B}(a) = \lfloor a/B \rfloor$). The function $\mathcal{S}$converts a memory block address to the cache set to which it maps (thus, $\mathcal{S}(b) = b \bmod S$).

Putting all of this notation together, we have the following table of objects of interest and their mathematical representations.

Object	**Mathematical representation**
An iteration point	ι
The ith array reference at that iteration	$R_i = (A_j, F_i)$
The array element accessed by R_i at ι	$e_i = A_j[F_i(\iota)]$
The byte address of e_i	$m_i = \mathcal{L}_j(F_i(\iota))$
The block address of m_i	$b_i = \mathcal{B}(\mathcal{L}_j(F_i(\iota)))$
The cache set to which b_i maps	$s_i = \mathcal{S}(\mathcal{B}(\mathcal{L}_j(F_i(\iota))))$

Example 1 Consider the following loop nest for matrix multiplication (the so-called ikj variant), which will be the specific computation whose cache behavior we analyze in the remainder of this paper.

```
for (i = 0; i < n; i++)
  for (k = 0; k < n; k++)
    for (j = 0; j < n; j++)
      C[i][j] = C[i][j] + A[i][k]*B[k][j];
```

This loop nest has depth $d = 3$. The LCVs are $\iota_1 = i$, $\iota_2 = k$, and $\iota_3 = j$. The loop nest accesses three arrays: $A^{(1)} = A$, $A^{(2)} = B$, and $A^{(3)} = C$. Each array is two-dimensional, so that $\mathcal{D}_1 = \mathcal{D}_2 = \mathcal{D}_3 = [0, n-1] \times [0, n-1]$. There are four array references: $R_1 = A[i][k]$, $R_2 = B[k][j]$, $R_3 = C[i][j]$ (the read access), and $R_4 = C[i][j]$ (the write access). The index expressions of the four references are $F_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \iota$, $F_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \iota$, and $F_3 = F_4 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \iota$. We defer the discussion of the layout functions of the three arrays to later in this section.

In the remainder of the paper we will work in units of array elements rather than bytes. Given that 32 bytes is a popular block size for first-level caches in many modern machines, and that double-precision numbers are represented with eight bytes, we will assume in this paper that memory blocks and cache blocks hold four array elements.

The goal of cache analysis is to efficiently estimate the number of capacity and conflict misses of a given code fragment, given the numerical value of the loop bounds, a cache configuration, and the layout functions of the arrays. To formulate the conditions under which the reference $R_i = (A, F_i)$ misses at iteration point w because it was replaced by reference $R_j = (B, F_j)$, let $u = \mathit{Last}(w)$ be the most recent iteration point that accessed $b_A = \mathcal{B}(\mathcal{L}_A(F_i(w)))$, the block being accessed by reference R_i at iteration point w. Let v be an iteration point satisfying $u < v < w$ at which the memory block $b_B = \mathcal{B}(\mathcal{L}_B(F_j(v)))$ accessed by reference R_j displaced block b_A from cache. This condition is satisfied iff

$$\mathcal{S}(\mathcal{B}(\mathcal{L}_A(F_i(w)))) = \mathcal{S}(\mathcal{B}(\mathcal{L}_B(F_j(v))). \tag{1}$$

Equation (1) captures both *capacity* and *conflict* misses, but does not distinguish between the two. (Discriminating between these miss classes would require the additional ability to ascertain the hit/miss status of the reference in a fully-associative cache.) It does not capture *compulsory* misses, as such misses correspond to iterations w for which *Last*(w) is not defined. We use the term *replacement misses* to encompass capacity and conflict misses. We omit compulsory misses from the scope of this paper for two reasons: they are unavoidable misses that cannot be reduced by optimization techniques, and they need to be formulated completely differently. It is clear that a simple strategy to count misses is through simulation of the code. This is exactly what cache simulators do. The main drawback of simulation is its slowness: it takes time proportional to the actual execution of the code, usually with a significant multiplicative factor ($10 - 100$ is typical). In the matrix multiplication example of Example 1, this time is $\Theta(n^3)$. Our interest is in much faster algorithms, whose existence is suggested by the regularity of the array access patterns and the limited number of cache sets to which they map. By using these regularities to tame the potential combinatorial explosion of cases, we will in fact demonstrate algorithms that accurately compute the number of cache misses for the matrx multiplication example in $O(\max(\log n, \log(C/B)))$ time.

Previous work [5] at this point introduces two additional constraints to make the problem tractable. First, it assumes that the layout functions are row- or column-major, which is affine in the array co-ordinates. We will subsequently use the term *canonical layout* to refer to these two layout functions. Second, it assumes that *Last*(w) can be obtained through *reuse vectors*, which occurs when the array index expressions are uniformly generated in addition to being affine in the LCVs. These two conditions keep everything within the polyhedral model [3], which has been well-studied and for which counting algorithms are well-known [9]. It is at this point that our work diverges from previous work.

Prior empirical evidence [4, 1, 2] suggests that alternative array layout functions such as Morton order [2] provide better cache behavior than canonical layout functions for many dense linear algebra codes. Such layout functions are described in terms of interleavings of the bits in the binary expansions of the array co-ordinates rather than as affine functions of the numerical values of these quantities. This single change puts our version of the problem beyond the scope of the solution techniques for the polyhedral model. We will therefore need to investigate different techniques for counting the number of solutions to equations such as equation (1).

1.3 Array layouts based on bit interleaving

In developing this model of alternative array layouts, we assume that $n = 2^m$, so that the bit representation of an array index will have m bits, with the least significant bit (LSB) numbered 0 and the most significant bit (MSB) numbered $m - 1$. We identify the binary sequence $s_{m-1} \dots s_0$ with the non-negative integer $s = \sum_{i=0}^{m-1} s_i 2^i$. We denote by B_m the set of all binary sequences of length m, and extend the above identification to identify B_m with interval $[0, 2^m - 1]$.

We will describe a family of nonlinear layout functions parameterized by a single parameter σ, as follows. An (m, m)*-interleaving*, σ, is a $2m$-bit binary sequence containing m 0s and m 1s. It describes the order in which bits from the two array coordinates are interleaved to linearize the array in memory. Given σ, define its *characteristic sequence* χ_σ to be the sequence with entries f_i and s_i defined by replacing the $(i+1)^{\text{st}}$ 0 from the right in σ by f_i and the $(i+1)^{\text{st}}$ 1 from the right in σ by s_i. (The letters f and s are chosen for mnemonic reasons: they are the initial letters of the words "first" and "second".)

Example 2 Let $m = 4$ and let $\sigma = 10110010$. Then $\chi_\sigma = s_3 f_3 s_2 s_1 f_2 f_1 s_0 f_0$. Next, let $m = 3$ and let $\sigma = 010011$. In this case, $\chi_\sigma = f_2 s_2 f_1 f_0 s_1 s_0$.

Given an (m, m)-interleaving σ, define a map

$$\Theta : B_m \times B_m \to B_{2m}$$

in the following way. If $a = a_{m-1} \dots a_1 a_0 \in B_m$ and $b = b_{m-1} \dots b_1 b_0 \in B_m$, then $\Theta(a, b)$ is the sequence obtained by replacing each f_i in χ_σ by a_i and each s_i in χ_σ by b_i. We extend this notation to consider Θ as a map from $[0, 2^m - 1] \times [0, 2^m - 1] \to [0, 2^{2m} - 1]$ by identifying non-negative integers and their binary expansions. We call Θ the *mixing function* indexed by σ. Note that $\Theta(0, 0) = 0$ for any σ.

Example 3 Let $m = 4$ and let $\sigma = 01101001$ so that $\chi_\sigma = f_3 s_3 s_2 f_2 s_1 f_1 f_0 s_0$. Then

$$\Theta(12, 5) = \Theta(1100, 0101) = 10110001 = 128 + 32 + 16 + 1 = 177.$$

Next, let $\sigma = 10110010$ so that $\chi_\sigma = s_3 f_3 s_2 s_1 f_2 f_1 s_0 f_0$. In this case,

$$\Theta(9, 6) = \Theta(1001, 0110) = 01110001 = 64 + 32 + 16 + 1 = 113.$$

Many popular layout functions fall into this class. For example, row-major layout corresponds to the signature $\sigma = \overbrace{0 \dots 0}^{n}\overbrace{1 \dots 1}^{n}$; column-major layout corresponds to the signature $\sigma = \overbrace{1 \dots 1}^{n}\overbrace{0 \dots 0}^{n}$; pure Morton layout corresponds to the signature $\sigma = \overbrace{01 \dots 01}^{2n}$; a combination of Morton layout with $2^k \times 2^k$ tiles arranged in row-major order corresponds to the signature $\sigma = \overbrace{01 \dots 01}^{2(n-k)}\overbrace{0 \dots 0}^{k}\overbrace{1 \dots 1}^{k}$; and so on.

We are now ready to discuss the matter of the layout functions of the three arrays in our matrix multiplication example. Given an arbitrary array element indexed (r, c), the quantity $\Theta(r, c)$ gives the position of the element (r, c) relative to the starting position of the array in memory. We use the generic notation μ to denote this starting address. Specifically, we assume the following forms of layout functions for A, B, and C:

$$\begin{aligned} \mathcal{L}_A(r, c) &= \mu_1 + \Theta(r, c) \\ \mathcal{L}_B(r, c) &= \mu_2 + \Theta(r, c) \\ \mathcal{L}_C(r, c) &= \mu_3 + \Theta(r, c). \end{aligned}$$

1.4 Goals and structure of the paper

Our overall goal, to be studied in a subsequent paper, is to find the layout functions of the form shown above that minimize cache misses. In this paper, we create an analytic model of cache misses using layout functions of this form, and we use this model to estimate the number of cache misses in the matrix multiplication example. These results will form the basis for the analysis in future work.

The counting of cache misses for the matrix multiplication example is, in the end, a giant case analysis of all possible patterns of interference among the various arrays. Fortunately, this analysis ultimately reduces to solving two enumeration problems, which are then adapted and augmented in diverse ways, and finally combined using inclusion-exclusion. We first discuss the two enumeration problems and their solutions in an abstract setting in Section 2. We then adapt these algorithms to the cache model in Section 3 and to the problem of counting cache misses in Section 4. We extend our analysis to set-associative caches in Section 5, and conclude in Section 6.

2 Two Enumeration Problems

In this section we study a pair of counting problems which together form the foundation for our enumeration of cache misses. We will not attempt to determine closed-form expressions for these numbers—almost certainly the answers to these questions cannot be put in elegant closed forms. Instead, our goal will be to describe efficient algorithms to determine the number of solutions.

We will let $n = 2^m$ and $p = 2^\rho$ be as in the last section. For any positive integer q we will let B_q denote the number of binary sequences $e = e_{q-1} \dots e_1 e_0$ of length q. When convenient, we will treat e as a non-negative integer in the range 0 to $2^q - 1$ using the usual notion of binary representation.

2.1 Algorithm AB(d)

Given an (m, m)-interleaving σ, an integer d with a ρ-bit binary expansion, and an initial carry $k_0 \in \{0, 1\}$, we want to determine **AB(d)**, the number of triples $(a, b, c) \in B_m^3$ such that

$$\Theta(a, b) = \Theta(b, c) + d + k_0 \bmod 2^\rho \tag{2}$$

under the condition that $2m \leqslant \rho$. A correct but inefficient algorithm would enumerate all possible triples (a, b, c) and check satisfiability of equation (2) for each triple. Such an algorithm would have time complexity of $O(2^m + \rho)$. The basic technique that we will use to derive an efficient algorithm of time complexity $O(\max(m, \rho))$ is to reason about individual bits of the terms on either side of the equation in terms of whether they *propagate* or *generate* carry bits. We will denote by k_i the carry input at bit position i (or, equivalently, the carry output at position $i - 1$). Note that k_0, the carry input at the least significant bit, is supplied.

The first observation is that we can simplify the problem based on the values of bits $d_{\rho-1}$ through d_{2m}.

Definition 2.1 (Consistency of d)
Let σ be an (m, m)-interleaving and let $d = d_{\rho-1} \dots d_0 \in B_\rho$. Let $r = [u, \dots, v]$ be a subsequence of $P = [0, \dots, \rho - 1]$. We say that d is *ϵ-consistent on* r if $d_j = \epsilon$ for all $j \in r$. We say that d is *inconsistent* on r if it is neither 0-consistent not 1-consistent on r.

Lemma 2.1 *Equation (2) has no solutions if d is inconsistent on* $[2m, \dots, \rho - 1]$. *For $\epsilon \in \{0, 1\}$, if d is ϵ-consistent on* $[2m, \dots, \rho - 1]$, *then equation (2) has solutions iff $k_{2m} = k_\rho = \epsilon$.*

Proof: By case analysis on bits $d_{\rho-1}$ through d_{2m}. □

This reduces the original problem to that of counting the number of solutions to a reduced system E of $2m$ bit-equations, and separating the solutions of E based on the value of k_{2m} that they produce. Let n_ϵ be the number of solutions of E that produce $k_{2m} = \epsilon$, for $\epsilon \in \{0, 1\}$. Then we have the following expression for **AB(d)**:

$$\textbf{AB(d)} = \begin{cases} n_0, & \text{if } d_{\rho-1} = \cdots = d_{2m} = 0 \\ n_1, & \text{if } d_{\rho-1} = \cdots = d_{2m} = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

We will now give an algorithm to determine the pair (n_0, n_1).

Let us label the $2m$ components of E with the numbers 0 through $2m - 1$, with t being the label of the equation corresponding to bit position t. Bit equation t has one of two forms:

$$b_i = c_i + d_t \tag{4}$$
$$a_i = b_i + d_t \tag{5}$$

where $0 \leqslant i < m$. For any fixed i, there is exactly one equation of form (4) and one equation of form (5) (of course, with different values of t). Of these, call the equation with larger value of t the *major* i-equation, an the equation with smaller value of t the *minor* i-equation.

The $+$ in the above equations is to be interpreted as binary addition, with hidden carry bits. To make this explicit, we rewrite the component equations in a more elaborate form, using the operations *exclusive-or* (denoted $\oplus$) and *majority* (denoted MAJ). For equation (4) we get

$$\begin{aligned} b_i &= c_i \oplus d_t \oplus k_t && (6) \\ k_{t+1} &= \text{MAJ}(c_i, d_t, k_t) && (7) \end{aligned}$$

while for equation (5) we get

$$\begin{aligned} a_i &= b_i \oplus d_t \oplus k_t && (8) \\ k_{t+1} &= \text{MAJ}(b_i, d_t, k_t) && (9) \end{aligned}$$

Our interest is not so much in specific values of the bits a_i, b_i, and c_i, but rather on the terminal carry k_{2m} that any particular assignment of bits produces. As b is the only variable that occurs on both sides of component equations, a particular choice of b uniquely determines values of a and c. We will therefore use the bits of b to collect solution triples that generate a common terminal carry. Looking at the behavior of component equation t for a specific choice of b_i, we observe that it has three possible *modes*.

1. $k_{t+1} = k_t$. We call this mode *Propagate*, or P for short.
2. $k_{t+1} = 0$, independent of k_t. We call this mode *0-Generate*, or G_0 for short.
3. $k_{t+1} = 1$, independent of k_t. We call this mode *1-Generate*, or G_1 for short.

The following lemma relates these modes to the choice of value b_i.

Lemma 2.2 *Equation (4) behaves in mode P if we set $b_i = d_t$ and in mode G_{d_t} if $b_i = \bar{d}_t$. Equation (5) behaves in mode G_{d_t} if we set $b_i = d_t$ and in mode P if $b_i = \bar{d}_t$.*

Proof: Simple case analysis based on possible values of b_i and of k_t. □

The key idea in the algorithm is to capitalize on the G_0 and G_1 modes. Consider b_{m-1}, the most significant bit of b. The major $(m-1)$-equation occurs at position $2m-1$, and the minor $(m-1)$-equation occus at some position s with $s < 2m-1$. Depending on the form of equation $2m-1$ and the value of d_{2m-1}, one of the two choices for b_{m-1} will lead to a G_ϵ-mode (with $\epsilon \in \{0, 1\}$). This means that no matter what values we assign to bits b_{m-2} through b_0, they will all contribute to n_ϵ. We can therefore increment n_ϵ by 2^{m-1}. The other choice of b_{m-1} will lead to P-mode for the major equation (and some mode for the minor equation as determined by Lemma 2.2). In this case, we need to explore further the assignment of values to lower-order bits of b to separate those assignments that contribute to n_0 from those that contribute to n_1. To do this, we will symbolically *reduce* the major and minor $(m-1)$-equations to their modes for this choice of b_{m-1}, and proceed to the equations involving b_{m-2}.

The reason behind the reduction of component equations to behavior modes becomes clear if we consider the situation when we are considering how the assignment of values to b_i, with $0 \leqslant i < m-1$, affects the counts n_0 and n_1. The fact that we are reasoning about b_i means:

- that we have already considered the bits b_{m-1} through b_{i+1};
- that we have identified the unique assignment of values of these bits that leads to P-modes for the major $(m-1)$-equation through the major $(i+1)$-equation;
- that we have reduced all of these major and minor equations to their appropriate behavior modes for these assignments of values to bits b_{m-1} through b_{i+1}.

If component equation t is the major i-equation, then this means that component equations $m-1$ through $t+1$ have been reduced. (Some of the component equations $t-1$ through 0 may also have been reduced; this does not concern us yet, because carries move from lower-order to higher-order bits.) In any case, one of the two choices of b_i will lead to a G_ϵ mode for component equation t. However, we cannot at this point simply increment n_ϵ by 2^{i-1}, since the generated carry k_{t+1} may be altered as it travels through the reduced component equations $t+1$ through $m-1$. What we need to do is to determine the value $k_{2m} = \delta$ that emerges at the other end of this process, and increment n_δ by 2^{i-1}. The representation of component equations as modes facilitates the determination of k_{2m}.

One final observation about the algebraic structure of modes allows us to calculate the terminal carry k_{2m} in a constant number of operations. It is easily seen that the mode set $\{P, G_0, G_1\}$ is a monoid under composition, with P as the identity element. Composition is defined by the following table.

	P	G_0	G_1
P	P	G_0	G_1
G_0	G_0	G_0	G_0
G_1	G_1	G_1	G_1

In trying to interpret this "composition table", remember that carries move from right to left. Thus, G_1P means that an input carry first passes through a P-mode and then through a G_1 mode. This is equivalent to a G_1 mode. Thus, instead of maintaining individual modes for reduced component equations $i+1$ through $2m-1$ and laboriously propagating k_{t+1} through them to obtain k_{2m}, we can keep a compact description of the combined effect of these modes and obtain k_{2m} from k_{t+1} in a single step. Furthermore, we can incrementally update this description as we move to lower-numbered component equations.

We are now ready to present the complete algorithm to determine (n_0, n_1).

$n_0 \leftarrow 0$
$n_1 \leftarrow 0$
mode $\leftarrow P$
$i \leftarrow m-1$
for $t = 2m-1$ **downto** 0 **do**
    if component equation t has been reduced to mode M **then**
        mode $\leftarrow$ COMPOSE(mode, M) /* Use composition table */
    else /* This is the major i-equation */
        $v \leftarrow$ value of b_i that makes this equation behave in mode G_ϵ, from Lemma 2.2
        $\delta \leftarrow$ APPLY(mode, ϵ)
        $n_\delta \leftarrow n_\delta + 2^i$
        Locate the minor i-equation and reduce it to the mode resulting from setting $b_i = \bar{v}$
        $i \leftarrow i-1$
    endif
enddo
$\delta \leftarrow$ APPLY(mode, k_0)
$n_\delta \leftarrow n_\delta + 1$

Theorem 2.1 *The above program correctly computes n_0 and n_1 and runs in $O(m)$ steps.*

Proof: Immediate from Lemmas 2.1 and 2.2. □

Example 4 Let $\sigma = 001110$, let $d_{2m-1}\cdots d_0 = 011000$ and let $k_0 = 1$. In this case, the equations are:

$$E_0: \qquad a_0 = b_0 + 0$$

$$\begin{array}{ll} E_1: & b_0 = c_0 + 0 \\ E_2: & b_1 = c_1 + 0 \\ E_3: & b_2 = c_2 + 1 \\ E_4: & a_1 = b_1 + 1 \\ E_5: & a_2 = b_2 + 0 \end{array}$$

The system of equations and (n_0, n_1) evolve in the following way as we go through the steps of the algorithm:
$\underline{t=5}$: Now $i = 2$. Set $v = 0$ because $b_2 = 0$ makes E_5 behave in mode G_0. Then $\delta = 0$ because mode $= P$ and $\epsilon = 0$ (i.e., a carry of 0 propagates through the reduced component equations). Update $n_0 = 0 + 4$. E_3 is the minor 2-equation and gets reduced to P-mode. This leaves:

$$\begin{array}{lll} E_0: & a_0 = b_0 + 0 & \\ E_1: & b_0 = c_0 + 0 & \\ E_2: & b_1 = c_1 + 0 & \\ E_3: & b_2 = c_2 + 1 & P\text{-mode} \\ E_4: & a_1 = b_1 + 1 & \\ E_5: & a_2 = b_2 + 0 & P\text{-mode} \end{array}$$

$\underline{t=4}$: Now $i = 1$. Set $v = 1$ and $\delta = 1$. Update $n_1 = 0 + 2$. E_2 is the minor 1-equation and gets reduced to P-mode. This leaves:

$$\begin{array}{lll} E_0: & a_0 = b_0 + 0 & \\ E_1: & b_0 = c_0 + 0 & \\ E_2: & b_1 = c_1 + 0 & P\text{-mode} \\ E_3: & b_2 = c_2 + 1 & P\text{-mode} \\ E_4: & a_1 = b_1 + 1 & P\text{-mode} \\ E_5: & a_2 = b_2 + 0 & P\text{-mode} \end{array}$$

$\underline{t=3}$: mode $= P$ because the previous value of mode, P, composed with the mode of E_3, P, is P.
$\underline{t=2}$: mode $= P$.
$\underline{t=1}$: Now $i = 0$. Set $v = 1$ and $\delta = 0$. Update $n_0 = 4 + 1$. E_0 is the minor 0-equation and gets reduced to G_0-mode. This leaves:

$$\begin{array}{lll} E_0: & a_0 = b_0 + 0 & G_0\text{-mode} \\ E_1: & b_0 = c_0 + 0 & P\text{-mode} \\ E_2: & b_1 = c_1 + 0 & P\text{-mode} \\ E_3: & b_2 = c_2 + 1 & P\text{-mode} \\ E_4: & a_1 = b_1 + 1 & P\text{-mode} \\ E_5: & a_2 = b_2 + 0 & P\text{-mode} \end{array}$$

$\underline{t = 0}$: mode $= G_0$. Set $\delta = 0$ and update $n_0 = 5 + 1$.

The final values for (n_0, n_1) are $(6, 2)$ which agrees with the answer obtained by explicit generation of all solutions. So, if $d_{\rho-1} = \cdots = d_{2m} = 0$ then **AB(d)** = 6, whereas if $d_{\rho-1} = \cdots = d_{2m} = 1$ then **AB(d)** = 2.

2.2 Algorithm AC(d)

We now investigate the following problem: Given an (m, m)-interleaving σ, an integer d with a ρ-bit binary expansion, determine **AC(d)**, the number of triples $(a, b, c) \in B_m^3$ such that

$$\Theta(a, b) = \Theta(a, c) + d \bmod 2^\rho \tag{10}$$

under the condition that $2m \leqslant \rho$. This problem is superficially similar to equation (2), with one small but critical difference: the variable that occurs on both sides of equation (10) occurs in the 0-positions of σ on both sides of the equation, whereas the variable that occurs on both sides of equation (2) occurs in the 0-position of σ on one side of the equation and the 1-position of σ on the other side of the equation. This difference makes the combinatorics of equation (10) radically different from the combinatorics of equation (2), leading in the end to a conceptually simpler algorithm to compute **AC(d)**.

If we write out equation (10) in terms of component bit-equations as we did for equation (2), we see that component equation t (for $0 \leqslant t < 2m$) has one of two forms: $a_i = a_i + d_t$ if $\sigma_t = 0$, and $b_i = c_i + d_t$ if $\sigma_t = 1$. The decoupling of the bits of a from the bits of b and c indicates that the a-component of any solution of equation (10) can be chosen independent of the b- and c-components. The decoupling also suggests that we need to look at the distribution of 0s and 1s in σ. Based on these observations, we start with a few definitions.

Definition 2.2 (Runs of σ)
Let σ be an (m, m)-interleaving, and let P be the sequence $[0, \ldots, \rho - 1]$. For $\epsilon \in \{0, 1\}$, an ϵ-*run* of σ is a maximal-length subsequence $[u, \ldots, v]$ of P such that $\sigma_u = \cdots = \sigma_v = \epsilon$, where σ_{2m} through $\sigma_{\rho-1}$ are declared to be 0. Order ϵ-runs in increasing order of u, and denote the ith ϵ-run of σ by $R_i^{(\epsilon)}$.

For technical reasons that will soon become evident, we will always want the "lowest" run to be a 0-run. This is a problem only when $\sigma_0 = 1$. In this case, we will create a special empty 0-run $R_1^{(0)}$ and label the non-empty 0-runs from $R_2^{(0)}$ onwards. Thus, $R_i^{(1)}$ is sandwiched between $R_i^{(0)}$ and $R_{i+1}^{(0)}$. Note also that the 0-runs constrain possible choices of a, while the 1-runs constrain possible choices of b and c.

We obtain strong conditions on the (non-)existence of solutions of equation (10) by considering the restrictions of d to the 0-runs of σ. The intuition behind the following lemma and its proof are small variations of Lemma 2.1.

Lemma 2.3 *Equation (10) has no solutions if d is inconsistent on any 0-run of σ. For $\epsilon \in \{0, 1\}$, if d is ϵ-consistent on $R_i^{(0)} = [u, \ldots, v]$, then equation (10) has solutions iff $k_u = k_{v+1} = \epsilon$. If $R_1^{(0)}$ is empty, then every d is declared to be 0-consistent on it.* □

Lemma 2.3 has two important consequences. First, it provides an early termination test for the algorithm. Second, if d is indeed consistent on all 0-runs of σ, then it simplifies the counting of the number of choices of a in the following way. Note that each of the component equations is of the form $a_i = a_i + d_t$. Since the same element of a appears on both sides of the equation, there is in fact *no* constraint on a! Thus, for every possible choice of b and c that we discover by examining the 1-runs (which we will do shortly), any of the 2^m choices of a will work.

Consider $R_i^{(1)} = [u, \ldots, v]$, the ith 1-run of σ. Recall that this run is sandwiched between runs $R_i^{(0)}$ and $R_{i+1}^{(0)}$. Let d be z_u-consistent on $R_u^{(0)}$. Let $t = |R_1^{(1)}| + \cdots + |R_{i-1}^{(1)}|$. Then the component equations in $R_i^{(1)}$ are as follows.

$$
\begin{aligned}
b_t &= c_t \oplus d_u \oplus k_u \\
k_{u+1} &= \text{MAJ}(c_t, d_u, k_u) \\
b_{t+1} &= c_{t+1} \oplus d_{u+1} \oplus k_{u+1} \\
k_{u+2} &= \text{MAJ}(c_{t+1}, d_{u+1}, k_{u+1}) \\
&\cdots \\
b_{t+v-u} &= c_{t+v-u} \oplus d_v \oplus k_v \\
k_{v+1} &= \text{MAJ}(c_{t+v-u}, d_v, k_v)
\end{aligned}
$$

By Lemma 2.3, we know that $k_u = z_i$ and $k_{v+1} = z_{i+1}$. Thus we are constrained by being given the values of both the initial and terminal carries of the 1-run, and must determine how many choices of bit values for b and c honor these constraints. It turns out that the easiest way to count the possibilities is to reason about the bit patterns as non-negative integers. To this end, define $\delta_i = z_u + \sum_{j=u}^{v} d_j 2^{j-u}$. That is, δ_i is the integer corresponding to the bit pattern $d_v \cdots d_u$, with the initial carry value absorbed into it. Also, let $\Delta_i = 2^{v-u+1} - \delta_i$. We then get the following result by case analysis on the value of k_{v+1}.

Theorem 2.2 *Let σ be an (m, m)-interleaving and let $d \in B_\rho$ be consistent on all 0-runs of σ. Then the number of solutions to equation (10) is $2^m \cdot \prod_{i=1}^{\ell} F_i$, where ℓ is the number of 1-runs of σ and*

$$
F_i = \begin{cases} \Delta_i, & \textit{if } d \textit{ is } 0\textit{-consistent on } R_{i+1}^{(0)} \\ \delta_i, & \textit{if } d \textit{ is } 1\textit{-consistent on } R_{i+1}^{(0)}. \end{cases}
$$

Proof: By equating the coefficients of the distinct powers of 2 on the two sides of (10) we arrive at a set of restrictions on the sequences a, b, c. Lemma 2.3 describes restrictions that result from equating coefficients of powers 2^τ where τ is in a 0-run of σ. The elements of a appear in these equations, with the same element of a appearing on both sides. This gives no restrictions on a and so there are $2^m = n$ choices for a. This accounts for the factor of 2^m that appears in the formula. The remaining factors will count the number of choices we have for b and c.

Consider restrictions on b and c that result from equating coefficients of powers 2^τ for τ in a particular 1-run $R_i^{(1)}$. Define β_i and γ_i by $\beta_i = \sum_{j=t}^{t+v-u} b_j 2^{j-t}$ and $\gamma_i = \sum_{j=t}^{t+v-u} c_j 2^{j-t}$.

<u>*Case 1*</u>: Suppose $z_{i+1} = 0$. Then the component equations on $R_i^{(1)}$ are equivalent to

$$
\beta_i - \gamma_i = \delta_i \tag{11}
$$

where we have equality of integers in equation (11). So, the number of choices we have for b_j, c_ℓ satisfying the component equation on $R_i^{(1)}$ is equal to the number of integers β_i, γ_i with $0 \leq \beta_i < 2^{v-u+1}$, $0 \leq \gamma_i < 2^{v-u+1}$ that satisfy equation (11). For each β_i with $\delta_i \leq \beta_i < 2^{v-u+1}$ there is exactly one choice of γ_i such that β_i, γ_i satisfy equation (11). For $0 \leq \beta_i < \delta_i$ there are no choices of γ_i such that β_i, γ_i satisfy equation (11). So the number of solutions to equation (11) is

$$
2^{v-u+1} - \delta_i = \Delta_i
$$

which is the i^{th} factor in the product in the statement of the theorem.

<u>*Case 2*</u>: Suppose $z_{i+1} = 1$. Then the component equations on $R_i^{(1)}$ are equivalent to:

$$
\beta_i + 2^{v-u+1} = \delta_i + \gamma_i
$$

which can be rewritten as:

$$\gamma_i - \beta_i = 2^{v-u+1} - \delta_i. \tag{12}$$

By the same reasoning as above, the number of solutions to equation (12) is δ_i, which is the i^{th} factor in the product in the statement of the theorem. $\square$

Example 5 Let $m = 5$, $\rho = 12$, $\sigma = 1000110110$ and $d = 111101001111$. We will use **Algorithm AC** to compute **AC(d)**.

In *Step 1* we compute the runs and the consistency values z_i:

$$\begin{aligned}
R_1^{(0)} &= [], z_1 = 0 \\
R_1^{(1)} &= [0] \\
R_2^{(0)} &= [1, 2, 3], z_2 = 1 \\
R_2^{(1)} &= [4, 5] \\
R_3^{(0)} &= [6], z_3 = 0 \\
R_3^{(1)} &= [7, 8] \\
R_4^{(0)} &= [9, 10, 11], z_4 = 1
\end{aligned}$$

In *Step 2* we compute the factors F_i and use them to determine **AC(d)**:

i	δ_i	Δ_i	F_i
1	1	1	1
2	3	1	1
3	2	2	2

So **AC(d)** = $32 \cdot 1 \cdot 1 \cdot 2 = 64$.

2.3 Counting joint solutions

The last problem we will consider in this section is to count those triples $a, b, c \in B_m$ which satisfy the two equations:

$$\Theta(a, b) = \Theta(b, c) + d \tag{13}$$

and

$$\Theta(a, b) = \Theta(a, c) + e \tag{14}$$

simultaneously. It is instructive to consider an example.

Example 6 Let $m = 5$, $\rho = 11$, $\sigma = 0110001011$, $d = 00010101111$, and $e = 00110001101$. Recalling the *characteristic sequence* notation from Section 1.3, $\chi_\sigma = f_4 s_4 s_3 f_3 f_2 f_1 s_2 f_0 s_1 s_0$. Then the simultaneous equations that must be satisfied are:

$$\Theta(a, b) = \Theta(b, c) + d \quad \Theta(a, b) = \Theta(a, c) + e$$

$$\begin{array}{rrr}
b_0 = c_0 + 1 & b_0 = c_0 + 1 & s_0 \\
b_1 = c_1 + 1 + k_0 & b_1 = c_1 + 0 + \ell_0 & s_1 \\
a_0 = b_0 + 1 + k_1 & a_0 = a_0 + 1 + \ell_1 & f_0
\end{array}$$

$$
\begin{array}{rrl}
b_2 = c_2 + 1 + k_2 & b_2 = c_2 + 1 + \ell_2 & s_2 \\
a_1 = b_1 + 0 + k_3 & a_1 = a_1 + 0 + \ell_3 & f_1 \\
a_2 = b_2 + 1 + k_4 & a_2 = a_2 + 0 + \ell_4 & f_2 \\
a_3 = b_3 + 0 + k_5 & a_3 = a_3 + 0 + \ell_5 & f_3 \\
b_3 = c_3 + 1 + k_6 & b_3 = c_3 + 1 + \ell_6 & s_3 \\
b_4 = c_4 + 0 + k_7 & b_4 = c_4 + 1 + \ell_7 & s_4 \\
a_4 = b_4 + 0 + k_8 & a_4 = a_4 + 0 + \ell_8 & f_4 \\
0 = 0 + k_9 & 0 = 0 + \ell_9 &
\end{array}
$$

In the above set of equations, k_t is the carry from the t^{th} to $(t+1)^{st}$ equation in $\Theta(a,b) = \Theta(b,c) + d$ whereas ℓ_t is the carry from the t^{th} to the $(t+1)^{st}$ equation in $\Theta(a,b) = \Theta(a,c) + e$. We will refer to these two sets of equations as the *d-system* and the *e-system*. Also, we will let B denote the number of f_i-equations in the system above. Note that $B = m$ if $2m \leq \rho$.

As we will see, it is seldom the case that there are any simultaneous solutions to equations (13) and (14). The next result states that even if there are simultaneous solutions, there are not very many.

Theorem 2.3 *The number of simultaneous solutions to equations (13) and (14) is less than or equal to* 2^{-B} *times the number of solutions to equation (14).*

Proof: Suppose there is a simultaneous solution to equations (13) and (14). Then the s_i-equations determine the values of $b_0, b_1, \ldots, b_{B-1}$. To this simultaneous solution of equations (13) and (14) we can correspond 2^B solutions to equation (14) which have the same b_i and c_i but where the choices of $a_0, a_1, \ldots, a_{B-1}$ range over all possibilities. □

One might ask whether there are instances in which the number of simultaneous solutions to equations (13) and (14) is exactly 2^{-B} times the number of solutions to equation (14). The next result tells us that this the case when $d = e$.

Definition 2.3
Let S denote the set of solutions to equation (14). We say two solutions $(a^{(1)}, b^{(1)}, c^{(1)})$ and $(a^{(2)}, b^{(2)}, c^{(2)}) \in S$ are *equivalent* if $b^{(1)} = b^{(2)}$ and $c^{(1)} = c^{(2)}$.

It is straightforward to see that every equivalence class has size n and that equivalence classes are indexed by pairs $b, c \in B_m$.

Theorem 2.4 *If* $d = e$*, then there is exactly one solution to equation (13) in every equivalence class of solutions to equation (14).*

Proof: Consider the equivalence class indexed by the pair b, c. It is clear that there is at most one solution (a, b, c) to equation (13) in that equivalence class because a_i is determined by equation f_i. It remains to show that there is at least one solution.

Consider the process of solving for the a_i and the carries k_t in equations of type (13) starting with b, c which gives (along with any a) a solution to equation (14). The thing we need to check is that the carries k_t we get in the equations of type (13) are identical to the carries ℓ_t we get in equations of type (14). We see this by induction on t.

Assume that $k_{t-1} = \ell_{t-1}$. There are two cases to consider. First assume equation t is labelled s_i so that in system (13) the t^{th} equation is $b_i = c_i + d_t + k_{t-1}$ and in system (14) the t^{th} equation is

$b_i = c_i + d_t + \ell_{t-1}$. In this situation it is clear that k_t will be equal to ℓ_t. Next, assume that equation t is labelled f_i. In this case the t^{th} equation in system (13) is $a_i = b_i + d_t + k_{t-1}$ whereas the t^{th} equation in system (14) is $a_i = a_i + d_t + \ell_{t-1}$. By Lemma 2.3 we have $\ell_t = \ell_{t-1} = d_t$. By our induction hypothesis, $k_{t-1} = \ell_{t-1} = d_t$. Because $k_{t-1} = d_t$ we have $k_t = d_t$ so $k_t = \ell_t$ which completes the induction step and finishes the proof. □

Corollary 2.5 *Let notation be as in Theorem 2.3. Then:*

1. *The number of triples (a, b, c) which are simultaneous solutions to $\Theta(a, b) = \Theta(b, c) + d$ and $\Theta(a, b) = \Theta(a, c) + d$ is $\prod F_i$.*
2. *The number of simultaneous solutions can be computed in $O(\rho)$ steps.*

The above results show that there are not very many simultaneous solutions of equations (13) and (14). The next results indicate that in most instances there are no simultaneous solutions.

Suppose there exist simultaneous solutions to equations (13) and (14). From our previous analysis, we know a number of things.

a) e must be consistent on 0-runs of σ.

b) In equation (14) the carry into any 0-run and carry out of that 0-run must both match the value of e on that run.

As a first test to whether there exist simultaneous solutions to equations (13) and (14), conditions **a)** and **b)** can be checked in $O(\rho)$ steps. We are now going to focus on 1-runs.

Suppose that equations $u, u + 1, \ldots, u + j - 1$ constitute a 1-run and that these equations are labeled $s_i, s_{i+1}, \ldots, s_{i+j-1}$. Let $\beta, \gamma, \delta, \varepsilon$ be the numbers with binary expansions given below:

$\beta = b_i b_{i+1} \cdots b_{i+j-1}$

$\gamma = c_i c_{i+1} \cdots c_{i+j-1}$

$\delta = d_u d_{u+1} \cdots d_{u+j-1}$

$\varepsilon = e_u e_{u+1} \cdots e_{u+j-1}$.

By comparing equations $s_i, \ldots, s_{i+j-1}$ in equations (13) and (14) we see that:

$$\beta = \gamma + \delta + k_{u-1} = \gamma + \varepsilon + \ell_{u-1}. \tag{15}$$

Also ℓ_{u-1} is specified to be the consistent value of e on preceding 0-run and γ must be chosen so that $\gamma + \varepsilon + \ell_{u-1}$ is less than 2^j iff the value on e on the subsequent 0-run is 0. From equation (15), the following result follows immediately.

Theorem 2.6 *Let δ and ε be the numbers whose binary expansions are given by the binary digits of d and e on a 1-run of σ as above. If there are simultaneous solutions to equations (13) and (14) then δ and ε must differ by no more than 1.*

More precisely, we must have one of the following four cases:

1. $\delta = \varepsilon + \ell_{u-1}$.

   Note: In this case we also must have $k_{u-1} = 0$.

2. $\delta = \varepsilon + \ell_{u-1} - 1$.

   Note: In this case we also must have $k_{u-1} = 1$.

3. $\delta = 0, \varepsilon = 2^j - 1, \ell_{u-1} = 1$.

   Note: In this case, we also must have that $k_{u-1} = 0$ and that e is consistently 1 on the next 0-run.

4. $\delta = 2^j - 1, \varepsilon = \ell_{u-1} = 0$.

   Note: In this case, we also must have that $k_{u-1} = 1$ and that e is consistently 0 on the next 0-run.

Theorem 2.6 gives another $O(\rho)$ test which can determine that there are no simultaneous solutions to equations (13) and (14). Note that if we assume d and e are chosen randomly, then Theorem 2.6 together with condition **a)** show that the probability that there exist simultaneous solutions to equations (13) and (14) is no more than $2^{-(B+U-R)}$, where B is the number of f_i-equations, U is the number of s_i-equations and R is the number of runs of σ. Note that $B + U = min\{2m, \rho\}$. Alternatively, if we have some freedom to choose d, e and σ, then the conditions given in **a)** and Theorem 2.6 can be used to insure that there are no simultaneous solutions to equations (13) and (14). We will return to this important point in our later paper on minimizing the number of cache misses.

It seems unlikely to us that there exists an algorithm which is polynomial in m or linear in ρ which determines the exact number of simultaneous solutions to equations (13) and (14). Just to conclude, we examine the case given in Example 6 just to point out some of the complexities of this problem.

Turning to the set of equations given in Example 6, we first examine whether the conditions set out in **a)** and **b)** hold. It can be seen that e is consistent on 0-runs with value 1 on f_0, value 0 on f_1, f_2, f_3 and value 1 on f_4. Condition **b)** thus implies that $\ell_1 = \ell_2 = 1, \ell_3 = \ell_4 = \ell_5 = \ell_6 = 0$ and $\ell_8 = \ell_9 = 0$.

To now consider the constraints given by Theorem 2.6, we must look at 1-runs. For the 1-run s_0, s_1, we have $\delta = \varepsilon = 3$ and $\ell_{-1} = 0$. So we are in Case 1. This implies that $k_{-1} = 0$ and that $\ell_1 = 1$. This gives a constraint on $\gamma = c_0c_1$ i.e., $\gamma + \varepsilon \geq 4$. This constraint on γ, which comes from consideration of the e equations, which implies that $k_1 = 1$.

Moving now to the 1-run s_2, we have that $\delta = 1, \varepsilon = 0$ and $l_2 = 1$. So we are again in Case 1 which implies that $k_2 = 0$. But now we have an inconsistency: it is impossible to have $k_1 = 1$ and $k_2 = 0$. So there are no simultaneous solutions to equations (13) and (14) in the case given in Example 6.

This particular example gives a flavor for the complex interplay that can take place between the constraints imposed by the d equations and those imposed by the e equations. At this time, we do not know a fast algorithm to determine the number of simultaneous solutions exactly.

3 Incorporating Cache Block Size

In the last section, we devised fast algorithms to compute the number of solutions to systems of equations of the form

$$\Theta(a, b) = \Theta(b, c) + d \mod 2^\rho$$

and

$$\Theta(a, b) = \Theta(a, c) + d \mod 2^\rho.$$

In practice, we will need to extend these algorithms to enumerate solutions to a slightly different pair of equations. Usually 2^λ memory locations fit into a cache block, represented by the denominator in the following equations.

Thus, the equation has to be taken $\mathrm{mod} 2^{\rho-\lambda}$. In practice, λ often equals 2 or 4. We show the case $\lambda = 2$ to provide the case distinction in full detail; the extension for $\lambda \in \mathbb{N}_0^+$ is straightforward, but requires consideration of more cases for $\lambda > 2$.

$$\lfloor \frac{\Theta(a,b)+\alpha}{4} \rfloor = \lfloor \frac{\Theta(b,c)+\beta}{4} \rfloor \mod 2^{\rho-2} \tag{16}$$

and

$$\lfloor \frac{\Theta(a,b)+\alpha}{4} \rfloor = \lfloor \frac{\Theta(a,c)+\beta}{4} \rfloor \mod 2^{\rho-2} \tag{17}$$

where $\alpha, \beta \in B_\rho$.

In this section we sketch methods, based on the ideas and algorithms developed in Section 2, to compute the number of solutions to equations (16) and (17). We will take the two equations in turn, starting with equation (17) because much of what we find there can later be reused for the treatment of equation (16).

3.1 Computing the number of solutions to equation (17)

To begin we will write out the digits in the binary expansions of $\Theta(a,b)+\alpha$ and $\Theta(a,c)+\beta$. Equating these expressions gives a system of equations $\underline{E} = E_0, E_1, \ldots, E_{\rho-1}$ where equation (17) imposes the requirement that equations $E_2, E_3, \ldots E_{\rho-1}$ must be satisfied mod 2. In order to satisfy equation (17), E_0 or E_1 need not hold mod 2. Consider E_0 and E_1. They look like one of the following:

$$\begin{aligned} a_0 + \alpha_0 &= a_0 + \beta_0 \\ a_1 + \alpha_1 + k_0 &= a_1 + \beta_1 + \ell_0 \end{aligned} \tag{18}$$

or

$$\begin{aligned} a_0 + \alpha_0 &= a_0 + \beta_0 \\ b_0 + \alpha_1 + k_0 &= c_0 + \beta_1 + \ell_0 \end{aligned} \tag{19}$$

or

$$\begin{aligned} b_0 + \alpha_0 &= c_0 + \beta_0 \\ a_0 + \alpha_1 + k_0 &= a_0 + \beta_1 + \ell_0 \end{aligned} \tag{20}$$

or

$$\begin{aligned} b_0 + \alpha_0 &= c_0 + \beta_0 \\ b_1 + \alpha_1 + k_0 &= c_1 + \beta_1 + \ell_0 \end{aligned} \tag{21}$$

where k_0 is the carry from the left side of E_0, and l_0 is the carry from the right side of E_0. Case (18) occurs when $\sigma_1\sigma_0 = 00$, (19) when $\sigma_1\sigma_0 = 10$, (20) when $\sigma_1\sigma_0 = 01$, and (21) when $\sigma_1\sigma_0 = 11$. The key observation is that the variables which appear in these equations do not appear in any of the later equations $E_2, E_3, \ldots, E_{\rho-1}$, because only a_i can occur more than once for each i and both instances of a_i are in equation E_i.

Our algorithm for enumerating the solutions to equation (17) begins with a loop over all possible choices of values for the variables that occur in E_0 and E_1. So, this outer loop runs through $4, 8, 8$, or 16 possibilities depending on whether we are in case (18), (19), (20), or (21) respectively.

Once values for these variables have been chosen, we compute the carries k_1 and ℓ_1 that are added to the left and right sides of E_2. Let

$$\alpha' = k_1 + \sum_{i=2}^{\rho-1} \alpha_i 2^{i-2}$$

and

$$\beta' = \ell_1 + \sum_{i=2}^{\rho-1} \beta_i 2^{i-2}.$$

Then the number of solutions to equation (17) with the chosen values for the variables in E_0 and E_1 is equal to the number of solutions to

$$\Theta'(a', b') + \alpha' = \Theta'(a', c') + \beta' \mod 2^{\rho-2} \tag{22}$$

where a', b', c' each come from B_{m-2}, B_{m-1} or B_m depending on whether $\sigma_1\sigma_0 = 00, 10, 01$ or 11. Here Θ' is the mixing function based on the interleaving σ' obtained from σ by deleting σ_0 and σ_1. Let

$$d = \begin{cases} \beta' - \alpha' \ \ if\ \beta' \geq \alpha' \\ 2^{\rho-2} + \beta' - \alpha' \ \ if\ \beta' < \alpha' \end{cases}$$

Then the number of solutions to equation (22) is equal to the number of solutions to

$$\Theta'(a', b') = \Theta'(a', c') + d$$

which can be computed using the AC algorithm in $O(m + \rho)$ steps.

With this generalization we call the algorithm the **extended AC Algorithm**.

3.2 Computing the number of solutions to equation (16)

As in Section 3.1 we will begin by writing out expressions for the digits in the binary expansions of $\Theta(a, b) + \alpha$ and $\Theta(b, c) + \beta$. This gives a system of equations $E_0, E_1, \ldots, E_{\rho-1}$ where the requirement of equation (16) is that $E_2, \ldots, E_{\rho-1}$ must be satisfied mod 2 (E_0 and E_1 need not hold mod 2).

Again, we will look at E_0, E_1 and find that they have one of four possible forms:

$$a_0 + \alpha_0 = b_0 + \beta_0$$

$$a_1 + \alpha_1 + k_0 = b_1 + \beta_1 + \ell_0 \tag{23}$$

or

$$a_0 + \alpha_0 = b_0 + \beta_0$$

$$b_0 + \alpha_1 + k_0 = c_0 + \beta_1 + \ell_0 \tag{24}$$

or

$$b_0 + \alpha_0 = c_0 + \beta_0$$

$$a_0 + \alpha_1 + k_0 = b_0 + \beta_1 + \ell_0 \tag{25}$$

or

$$b_0 + \alpha_0 = c_0 + \beta_0$$

$$b_1 + \alpha_1 + k_0 = c_1 + \beta_1 + \ell_0 \tag{26}$$

where k_0 is the carry from the left side of E_0, and l_0 is the carry from the right side of E_0. Case (23) occurs when $\sigma_1\sigma_0 = 00$, (24) when $\sigma_1\sigma_0 = 10$, (25) when $\sigma_1\sigma_0 = 01$, and (26) when $\sigma_1\sigma_0 = 11$. Note that in (24) and (25) the variables which occur in equations E_0 and E_1 do not occur in $E_2, E_3, \ldots, E_{\rho-1}$, because only b_i can occur more than once for each i and both instances of b_0 occur in E_0 and E_1. Therefore, we can use the same method we used in Section 3.1 to devise fast algorithms to compute the number of solutions.

Cases (23) and (26) are slightly different because b_0 and b_1 may occur later in $E_2, E_3, \ldots, E_{\rho-1}$. In case (23), our algorithm has an outside loop over the four possible choices of values for a_0 and a_1. For each choice of these values, we compute the carry k_1 which is added to the left-hand side of E_2. We let

$$\alpha' = 4k_1 + \sum_{i=2}^{\rho-1} \alpha_i 2^i$$

and we apply the **AB Algorithm** to count the number of solutions to

$$\Theta(a,b) = \Theta(b,c) + d$$

where

$$d = \begin{cases} \beta - \alpha' \;\; if \; \beta \geq \alpha' \\ 2^\rho + \beta - \alpha' \;\; if \; \beta < \alpha'. \end{cases}$$

This number is equal to the number of solutions to equation (16) in which a_0 and a_1 have the specified values.

We handle case (26) in a way quite similar to (23). We loop over the four possible choices of values for c_0 and c_1. For each of these values, we compute the carry l_1 which is added to the right-hand side of E_2 and let

$$\beta' = 4l_1 + \sum_{i=2}^{\rho-1} \beta_i 2^i$$

and we apply the **AB Algorithm** to count the number of solutions to

$$\Theta(a,b) = \Theta(b,c) + d$$

where

$$d = \begin{cases} \beta' - \alpha \;\; if \; \beta' \geq \alpha \\ 2^\rho + \beta' - \alpha \;\; if \; \beta' < \alpha. \end{cases}$$

This number is equal to the number of solutions to equation (16) in which c_0 and c_1 have the specified values.

With this generalization we call the algorithm the **extended AB Algorithm**.

4 Calculating the Number of Cache Misses

In this section we return to the problem of counting cache misses. Recall that we are analyzing the data layout function defined in terms of an m, m interleaving $\sigma = \sigma_{2m-1} \ldots \sigma_1 \sigma_0$ by

$A_{i,k}$ maps to $\mu_1 + \Theta(i,k)$

$B_{k,j}$ maps to $\mu_2 + \Theta(k,j)$

$C_{i,j}$ maps to $\mu_3 + \Theta(i,j)$.

and that we use the following suggestive notation:

A miss is the number of cache misses when accessing an element of A,

A-B miss is the number of cache misses which occur when an element of A is accessed which was in cache but was removed because an element of B took its place,

A-BC miss is the number of cache misses which occur when an element of A is accessed which was previously in cache and such that both an element of B and an element of C have taken its place in cache since it was most recently there, and so on.

Considering the inclusion-exclusion property of set intersections, the task is to enumerate the following types of misses:

$$\begin{aligned}
\textbf{A miss} &= \textbf{A-A miss} + \textbf{A-B miss} + \textbf{A-C miss} - \textbf{A-AB miss} - \textbf{A-BC miss} - \textbf{A-AC miss} + \textbf{A-ABC miss} \\
\textbf{B miss} &= \textbf{B-A miss} + \textbf{B-B miss} + \textbf{B-C miss} - \textbf{B-AB miss} - \textbf{B-BC miss} - \textbf{B-AC miss} + \textbf{B-ABC miss} \\
\textbf{C miss} &= \textbf{C-A miss} + \textbf{C-B miss} + \textbf{C-C miss} - \textbf{C-AB miss} - \textbf{C-BC miss} - \textbf{C-AC miss} + \textbf{C-ABC miss}
\end{aligned}$$

Figure 1 shows this for **A miss**.

However in the special case of matrix multiplication, some misses need not be considered; in particular there are no **A-A**, **A-AB**, **A-AC**, or **A-ABC misses** because unique elements $A_{i,k}$ are accessed in the two outermost loops only. A method to derive the types of misses that are *required* in the more general case of programs other than matrix multiplication is subject of future work. We have also proven in Section 2 that the number of simultaneous solutions is very small.

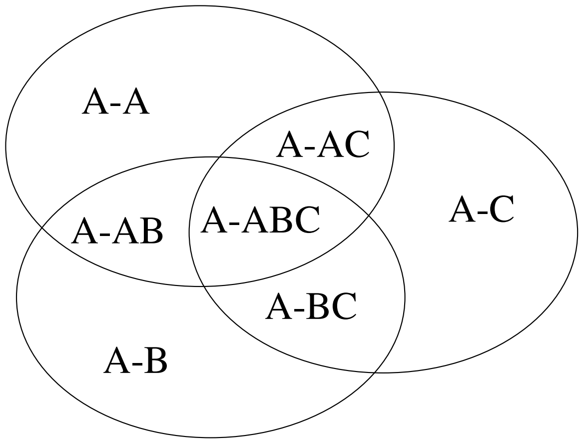

Figure 1: **A miss**: inclusion-exclusion property

Please note that we are not including every type of miss in our analysis, but are including a case that represents each of the key ideas involved in counting the number of cache misses.

Throughout this section, we will continue to assume that $2m \leq \rho$ to simplify the exposition. In the case that $2m > \rho$, the following changes must be made to the analysis in this section. Each time an iteration point (i, k, j) is counted as a miss, then only initial segments of the binary expansions of i, k and j are determined.

There are no constraints on how these initial segments are extended to give complete binary expansions of i, k and j. So each miss enumerated in this section must be multiplied by 2^D where D is the total number of undetermined binary digits in i, k and j (the number D depends on which kind of miss is being enumerated and so must be determined on a case by case basis).

4.1 Computing A miss

In this subsection we show how to efficiently compute **A miss**. An array element $A_{i,k}$ will be accessed at the n iteration points (i, k, w), where $0 \leq w \leq n-1$. Suppose that we have a cache miss when $A_{i,k}$ is accessed at the iteration point (i, k, j). As the same element of A is accessed throughout the innermost loop, there are no **A-A misses**. Since we are using the lexicographic ordering $i \succ k \succ j$, the iteration point (i, k, j) is immediately preceded by the iteration point $(i, k, j-1)$ at which the array element $A_{i,k}$ is accessed. Thus at the iteration point $(i, k, j-1)$ there must be a memory access of an element of B or C which occupies the same cache set as $A_{i,k}$.

Although possibly negligible, there could also be a small number of contributions to **A miss** along the boundary of the innermost loop. The array element $A_{i,k-1}$ is accessed during the $(i, k-1, n-1)$ iteration step and the array element $A_{i,k}$ is accessed during the following iteration step, $(i, k, 0)$. Suppose $A_{i,k-1}$ and $A_{i,k}$ occupy the same cache word, then a cache miss occurs if there is a memory access of an element of B or C that maps to the same cache set as $A_{i,k-1}$ at $(i, k-1, n-1)$. It will be the case that the array element $A_{i,k}$ existed in the cache, but was removed by an access to $B_{k-1,n-1}$ or $C_{i,n-1}$ at iteration step $(i, k-1, n-1)$.

We can now examine **A-B miss** and **A-C miss** separately.

4.1.1 Computing A-B miss

During the (i, k, j) iteration step we form the product $A_{i,k} \cdot B_{k,j}$ and add it to $C_{i,j}$. When we do so, we access these three pieces of information in the order $A_{i,k}$ followed by $B_{k,j}$ followed by $C_{i,j}$. So, in order for this cache miss to contribute to **A-B miss**, it must be the case that the array element $A_{i,k}$ was removed from cache at the previous iteration step when the array element $B_{k,j-1}$ was accessed, i.e., $A_{i,k}$ and $B_{k,j-1}$ occupy the same word in cache. This is equivalent to:

$$\lfloor \frac{\mu_1 + \Theta(i,k)}{4} \rfloor = \lfloor \frac{\mu_2 + \Theta(k, j-1)}{4} \rfloor \mod 2^{\rho-2} \tag{27}$$

where this equation is taken $\mathrm{mod} \frac{p}{4} = 2^{\rho-2}$. So **A-B miss** is equal to the number of solutions (i, k, j) to equation (27) with $0 \le i \le n-1$, $0 \le k \le n-1$ and $1 \le j \le n-1$. The number of solutions to equation (27) is computed by the **Extended AB Algorithm**.

To count **A-B misses** along the boundary of the innermost loop, we determine if $A_{i,k-1}$ and $A_{i,k}$ occupy the same cache word

$$\lfloor \frac{\mu_1 + \Theta(i, k-1)}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(i,k)}{4} \rfloor \tag{28}$$

and if so, we check if an access to the array element $B_{k-1,n-1}$ causes a cache miss

$$\lfloor \frac{\mu_1 + \Theta(i, k-1)}{4} \rfloor = \lfloor \frac{\mu_2 + \Theta(k-1, n-1)}{4} \rfloor \mod 2^{\rho-2}$$

incrementing the **A-B miss** count if both equations are satisfied.

4.1.2 Computing A-C miss

By the same reasoning as above, the number of cache misses that contribute to **A-C miss** is the number of solutions to

$$\lfloor \frac{\mu_1 + \Theta(i,k)}{4} \rfloor = \lfloor \frac{\mu_3 + \Theta(i, j-1)}{4} \rfloor \mod 2^{\rho-2} \tag{29}$$

where this equation is taken modulo $\frac{p}{4} = 2^{\rho-2}$ and i, j, k are constrained to lie in the intervals $0 \le i \le n-1, 0 \le k \le n-1$ and $1 \le j \le n-1$. The number of solutions to equation (29) is computed by the **Extended AC Algorithm**.

To count the contributions to **A-C miss** along the boundary of the innermost loop, we check if $A_{i,k-1}$ and $A_{i,k}$ occupy the same cache word exactly as in equation (28), and if so we determine if an access to the array element $C_{i,n-1}$ causes a cache miss

$$\lfloor \frac{\mu_1 + \Theta(i, k-1)}{4} \rfloor = \lfloor \frac{\mu_3 + \Theta(i, n-1)}{4} \rfloor \mod 2^{\rho-2}$$

incrementing the **A-C miss** count if both equations are satisfied.

4.1.3 Computing A-BC miss

We will count zero **A-BC misses**. The conditions in Section 2.3 can be checked in $O(\rho)$ steps to determine whether this count is accurate. As proved in Section 2.3, even if there are instances of such misses, their number is small – less than 2^{-T} of the total number of misses, where T is the number of ones in the set $\{\sigma_2, \sigma_3, \ldots, \sigma_{\rho-1}\}$. In fact, on the basis of this result, we are setting all terms requiring the simultaneous solving of equations (e.g., **B-AB miss**, **B-BC miss**, **B-AC miss**, **B-ABC miss**, **C-AB miss**, **C-BC miss**, **C-AC miss**, **C-ABC miss**) to zero.

4.2 Computing C miss

The quantity **C miss** counts the number of iteration points (i, k, j) with $k > 0$ such that the matrix element $C[i, k, j]$ is not in cache thereby causing a miss. As a first step, we will determine $L[i, k, j]$ which denotes the most recent iteration step, prior to (i, k, j) at which $C[i, k, j]$ was in cache. Note that $L[i, k, j]$ is the most recent iteration step when an element of C was accessed that occupies the same cache word as $C[i, k, j]$. If we write $L[i, k, j] = (i', k', j')$ this is equivalent to:

$$\lfloor \frac{\mu_3 + \Theta(i, j)}{4} \rfloor = \lfloor \frac{\mu_3 + \Theta(i', j')}{4} \rfloor \qquad (30)$$

4.2.1 Computing C-A miss

The solution to equation (30) depends on the form of Θ and so at this point the analysis must break into cases. There are four cases to consider depending on whether $\sigma_1\sigma_0 = 00,\ 01,\ 10$ or 11. We will write out details in two of the cases which represent the technical problems that come up in the other two cases. The details of the remaining two cases are left to the reader.

<u>$Case\ 1$</u>: $\sigma_1\sigma_0 = 00$.

In this case, the four elements of C which occupy the same cache word as $C[i, k, j] = C_{i,j}$ are *usually* $C_{i,j-u}, C_{i,j-u+1}, C_{i,j-u+2}, C_{i,j-u+3}$ where $\mu_3 + \Theta(i, j) \equiv u \mod 4$. The modifier 'usually' refers to the observation that not all of these elements of C might exist in the extreme cases where $j < u$ or $j - u + 3 \geq n$. But as long as $u > 0$ and $j > 0$, $C_{i,j-1}$ is in the same cache word as $C_{i,j}$. In this case, $C[i, k, j]$ is brought into cache at the preceding iteration step $(i, k, j-1)$ and so $L(i, k, j) = (i, k, j-1)$.

If $j = 0$ or $u = 0$ then $L(i, k, j) = (i, k-1, j-u+3)$ unless $j - u + 3 \geq n$. In that case ($u = 0$ and $j - u + 3 \geq n$), $L(i, k, j) = (i, k-1, n-1)$. To summarize: if $u \equiv \mu_3 + \Theta(i, j) \mod 4$, then

$$L(i, k, j) = \begin{cases} (i, k, j-1) & \text{if } j > 0 \text{ and } u > 0 \\ (i, k-1, j-u+3) & \text{if } j = 0 \text{ or } \{u = 0 \text{ and } j - u + 3 < n\} \\ (i, k-1, n-1) & \text{if } u = 0 \text{ and } j - u + 3 \geq n. \end{cases}$$

Now **C-A miss** is the number of pairs of iteration points $(i, k, j), (x, z, y)$ such that

$$L(i, k, j) < (x, z, y) \leq (i, k, j) \qquad (31)$$

and

$$\lfloor\frac{\mu_1+\Theta(x,z)}{4}\rfloor = \lfloor\frac{\mu_3+\Theta(i,j)}{4}\rfloor \mod 2^{\rho-2} \tag{32}$$

To clarify the connection between **C-A miss**, equation (31), and equation (32) note that equation (32) states that $A[x,z,y]$ and $C[i,k,j]$ occupy the same cache word and equation (31) states that iteration step (x,z,y) occurs sometime between the iteration step (i,k,j) and the previous iteration step when $C[i,k,j]$ was brought into cache.

We now break our analysis into two cases depending on the exact form of $L[i,k,j]$. If $L[i,k,j] = (i,k,j-1)$ then we must have $(x,z,y) = (i,k,j)$. Also, if $u = 0$ and $j+3 \geq n$ so that $L[i,k,j] = (i,k-1,n-1)$ then equation (31) becomes $(i,k-1,n-1) < (x,z,y) \leq (i,k,j)$. This cannot be satisfied with $z = k-1$ because we would then need $n-1 < y$. So we must have $z = k$ and $y = j$. This is a second instance in which (x,z,y) must be equal to (i,k,j). In this case, equation (32) states:

$$\lfloor\frac{\mu_1+\Theta(i,k)}{4}\rfloor = \lfloor\frac{\mu_3+\Theta(i,j)}{4}\rfloor \mod 2^{\rho-2}$$

Solutions to this equation are enumerated by the **Extended AC Algorithm**.

If $j = 0$ or $u = 0$ and $j-u+3 \leq n-1$ then equation (31) states that $(i,k-1,j-u+3) < (x,z,y) \leq (i,k,j)$. We deduce that $x = i$ and that z is equal to either $k-1$ or k. Also, in this case we cannot satisfy the inequality $j-u+3 < y \leq j$ so we must have $z = k-1$. Thus the contribution to **C-A miss** made in this case is number of solutions to equation (32) which is:

$$\lfloor\frac{\mu_1+\Theta(i,k-1)}{4}\rfloor = \lfloor\frac{\mu_3+\Theta(i,j)}{4}\rfloor \mod 2^{\rho-2}$$

This is equivalent to enumerating solutions to

$$\lfloor\frac{\mu_1+\Theta(i,k')}{4}\rfloor = \lfloor\frac{\mu_3+\Theta(i,j)}{4}\rfloor \mod 2^{\rho-2}$$

where $0 \leq i \leq n-1, 0 \leq k' \leq n-2, 0 \leq j \leq n-1$. Solutions to this equation are enumerated by the **Extended AC algorithm**.

This completes Case 1 in our analysis of **C-A miss**.

<u>*Case 2*</u>: $\sigma_1\sigma_0 = 10$.

The fundamental difference between the analysis in this case and the analysis in Case 1 is the relationship between cache words and the arrays A, B, C. In particular, the elements of C that occupy the same cache word as $C_{i,j}$ are

$$C_{i-v,j-u}, C_{i-v+1,j-u}, C_{i-v,j-u+1}, C_{i-v+1,j-u+1} \tag{33}$$

where

$$\mu_3+\Theta(i,j) \equiv v \mod 2$$

and

$$\frac{\mu_3+\Theta(i,j)-v}{2} \equiv u \mod 2$$

The analysis now parallels the analysis in Case 1 but with changes in some details to reflect the cache word structure given in equation (33).

If $j > 0$ and $u > 0$ the $L[i,k,j] = (i,k,j-1)$ and we proceed as in Case 1. If $u = 0$ and $j = n-1$ then $L[i,k,j] = (i,k-1,n-1) = (i,k-1,j)$. In both these cases, if a cache miss is caused by the access of $A_{x,z}$ removing $C_{i,j}$ at iteration step (x,z,y), where $L[i,k,j] < (x,z,y) \leq (i,k,j)$, then we must have $(x,z,y) = (i,k,j)$. These instances are enumerated as in Case 1 by the **Extended AC algorithm**.

If $j = 0$ or if $u = 0$ and $j < n-1$ then $L(i,k,j) = (i, k-1, j-u+1)$. In this case, (x, z, y) must equal $(i, k-1, j-u+1)$ and we enumerate these instances as in Case 1. This completes the computation of **C-A miss** in Case 2.

The computation of **C-A miss** in the remaining two cases is similar.

Note that **C-C miss** can be handled in a manner similar to **C-A miss**. There is the same consideration of the most recent iteration step at which an element of C was addessed that occupies the same cache word as $C[i,k,j]$, and the analysis breaks into the same four cases depending on $\sigma_1\sigma_0$. The key difference is that in this case, an access to $C_{x,y}$ interferes with an access to $C_{i,j}$, where x, y are as in equation (31).

4.2.2 Computing C-B miss

To compute **C-B miss**, we need to compute the number of pairs of triples (i,k,j), (x,z,y) which satisfy equation (31) such that $C[i,k,j] = C_{i,j}$ and $B[x,z,y] = B_{z,y}$ occupy the same cache block. This latter condition is equivalent to:

$$\lfloor \frac{\mu_3 + \Theta(i,j)}{4} \rfloor = \lfloor \frac{\mu_2 + \Theta(z,y)}{4} \rfloor \mod 2^{\rho-2} \tag{34}$$

As in the previous subsection, if $\lfloor \frac{\mu_3+\Theta(i,j-1)}{4} \rfloor = \lfloor \frac{\mu_3+\Theta(i,j)}{4} \rfloor$, then equation (31) implies that $(x,z,y) = (i,k,j)$. In that case, equation (34) is equivalent to:

$$\lfloor \frac{\mu_3 + \Theta(i,j)}{4} \rfloor = \lfloor \frac{\mu_2 + \Theta(k,j)}{4} \rfloor \mod 2^{\rho-2} \tag{35}$$

Let $\hat{\sigma}$ be the interleaving obtained from σ by interchanging 0's and 1's and let $\hat{\Theta}$ denote the mixing function determined by $\hat{\sigma}$. Note that for any pair of non-negative integers v, w:

$$\Theta(v,w) = \hat{\Theta}(w,v).$$

Thus, we can re-write equation (35) by

$$\lfloor \frac{\mu_3 + \hat{\Theta}(j,i)}{4} \rfloor = \lfloor \frac{\mu_2 + \hat{\Theta}(j,k)}{4} \rfloor \mod 2^{\rho-2} \tag{36}$$

The **Extended AC Algorithm** counts solutions to equation (36) which gives us a fast algorithm to count contributions to **C-B miss** that arise in the instances where $(x,z,y) = (i,k,j)$.

The remaining contributions to **C-B miss** come from solutions to equation (34) in the cases where either $j = 0$ or

$$\lfloor \frac{\mu_3 + \Theta(i,j-1)}{4} \rfloor \neq \lfloor \frac{\mu_3 + \Theta(i,j)}{4} \rfloor$$

The analyses of these two cases are somewhat different and so we do them separately.

Consider the case where

$$\lfloor \frac{\mu_3 + \Theta(i,j-1)}{4} \rfloor \neq \lfloor \frac{\mu_3 + \Theta(i,j)}{4} \rfloor$$

This can occur in one of three different ways. If $\sigma_0 = 1$ then this condition is equivalent to $\mu_3 + \Theta(i,j) \equiv 0 \mod 4$. If $\sigma_1\sigma_0 = 10$ then this condition is equivalent to $\mu_3 + \Theta(i,j) \equiv 0, 1 \mod 4$. Lastly, if $\sigma_1\sigma_0 = 00$ then this condition always holds.

In this case, we have $L(i,k,j) = (i, k-1, j+1)$ and so equation (31) becomes

$$(i, k-1, j+1) < (x,z,y) \leq (i,k,j).$$

At first glance, the enumeration of solutions to equation (34) appears to be problematic. Although we can deduce that z is either $k-1$ or k, we have very little control on j. So equation (34) contains four variables that are essentially independent. After some simplification of the problem, we will see that this is in fact an advantage and makes the enumeration of solutions particularly easy.

To count solutions to equation (34) we first loop over all possible values for the first two digits in the binary expansions of $\Theta(i,j)$ and $\Theta(z,y)$. This will involve specifying the first two digits of i, or the first digit of i and the first digit of j, or the first two digits of j depending on the values of $\sigma_1\sigma_0$. Let i', j' be the remaining, unspecified digits of i and j. Define z' and y' similarly. Also, define Θ' to be the mixing function associated with $\sigma' = \ldots\sigma_3\sigma_2$.

Having specified the first two digits of $\Theta(i,j)$ and $\Theta(z,y)$ we next perform the following steps:

1) Check whether $\lfloor\frac{\mu_3+\Theta(i,j-1)}{4}\rfloor = \lfloor\frac{\mu_3+\Theta(i,j)}{4}\rfloor$. If so, go to the next step in the loop (here *loop* refers to the outermost loop whose steps are indexed by the choices for possible first two digits of $\Theta(i,j)$ and $\Theta(z,y)$). Otherwise, continue to step 2).

2) Let μ_3' be $\lfloor\frac{\mu_3}{4}\rfloor + \epsilon_1$ where ϵ_1 is the binary carry from the first to the second binary digits in $\mu_3 + \Theta(i,j)$.

3) Let μ_2' be $\lfloor\frac{\mu_2}{4}\rfloor + \tau_1$ where τ_1 is the binary carry from the first to the second binary digits in $\mu_2 + \Theta(z,y)$.

The number of solutions to equation (34), given the specified digits in $\Theta(i,j)$ and $\Theta(z,y)$ is equal to the number of solutions to

$$\mu_3' + \Theta'(i',j') = \mu_2' + \Theta'(z',y') \mod 2^{\rho-2} \tag{37}$$

Define d by:

$$d = \begin{cases} \mu_3' - \mu_2' & if\ \mu_3' \geq \mu_2' \\ \frac{\rho}{4} + \mu_3' - \mu_2' & if\ \mu_3' < \mu_2' \end{cases}$$

Then the number of solutions to equation (37) is equal to the number of solutions to equation (38):

$$d + \Theta'(i',j') = \Theta'(z',y') \mod 2^{\rho-2} \tag{38}$$

Enumerating solutions to equation (38) is straightforward. Let d' consist of the first $2m-2$ binary digits of d. First, examine the remaining binary digits, d_ℓ for $2m-2 \leq \ell \leq \rho-3$. Unless these remaining binary digits are identical, there are no solutions to equation (38). If they are identical and equal to ϵ, then the number of solutions to equation (38) is equal to the number of solutions to:

$$d' + \Theta'(i',j') = \Theta'(z',y') \mod 2^{2m-2} \tag{39}$$

for which the terminal carry on the left hand side of equation (39) is ϵ. The key observation is that the number of solutions to equation (39) is equal to the number of pairs (i',j') which result in terminal carry ϵ – once such a pair has been specified there is a unique choice of z', y' which satisy equation (39).

In our design of the **AC Algorithm** (Section 2), we exactly determined the number of pairs i', j' which give terminal carry ϵ on the left-hand side of equation (39). This number is:

$$\begin{cases} 2^{2m-2} - d' & if\ \epsilon = 0 \\ d' & if\ \epsilon = 1. \end{cases}$$

This finishes our enumeration of solutions to equation (34) in the case that $\lfloor\frac{\mu_3+\Theta(i,j-1)}{4}\rfloor \neq \lfloor\frac{\mu_3+\Theta(i,j)}{4}\rfloor$. To evaluate the complexity of this enumeration algorithm: there is an outer loop through the sixteen possible choices for the first two binary digits of $\Theta(i,j)$ and $\Theta(z,y)$. Within that loop, we need to determine d and check whether the binary digits of d are consistent between $2m-2$ and $\rho-3$. That takes $O(\rho)$ operations.

Then we need to compute d' which can be done in $O(m)$ operations. Thus the total complexity in this case is $O(\rho)$.

To complete the calculation of **C-B miss** we need to consider the case where $j = 0$. By the same series of reductions we used in the previous case, this reduces to enumerating solutions to

$$d' + \Theta'(i', 0) = \Theta'(z', y') \mod 2^{2m-2} \tag{40}$$

As before, equation (40) has no solutions unless the binary digits $d_\ell, \ell = \rho - 3, \ldots, 2m-1, 2m-2$ are identical. If they are identical, let $\epsilon \in \{0, 1\}$ represent their common value. In that case, the number of solutions to equation (40) is equal to the number of i' for which $d' + \Theta'(i', 0)$ has terminal carry ϵ.

To efficiently compute this number, first scan the left hand sides of the equations in equation (40) from bottom to top. Consider those left hand sides which have the form $d'_\ell + 0 + k_{\ell-1}$ (where 0 comes from the $j' = 0$ component in $\Theta'(i', 0)$, i.e., $\sigma_{\ell+2} = 1$). If $d'_\ell = 1$ then erase that equation as you know that whatever carry comes in, the same carry will go out. If $d'_\ell = 0$ then stop your scan. You know that k_ℓ will have to be 0. So we can start constructing solutions from the next equation on without regard to any earlier binary digits of i'. To this end, let τ be the number of i'_a which occur in equation in equation (40) that comes before your stopping point. Let d'' be the digits of d' that remain on the left hand side of equation (40) above the ℓ^{th} digit (we use the word *remain* because we have erased some equations at earlier steps in the scan). Let i'' be the corresponding digits of i' and let μ be the number of digits of d''. Then the number of solutions to equation (40) is 2^τ times the number of i'' such that

$$\begin{cases} d'' + i'' < 2^\mu & if\ \epsilon = 0 \\ d'' + i'' \geq 2^\mu & if\ \epsilon = 1 \end{cases}$$

This number is computed as before which completes the $j = 0$ case and therefore completes our computation of **C-B miss**.

Note that the algorithm described here to handle the case $j = 0$ has complexity $O(\rho)$. Also note that for each choice of initial two digits in $\Theta(i, j)$, the solutions to equation (34) where $j = 0$ are either contained in, or else disjoint from, the solutions where $\lfloor \frac{\mu_3 + \Theta(i,j-1)}{4} \rfloor \neq \lfloor \frac{\mu_3 + \Theta(i,j)}{4} \rfloor$. It is trivial to determine whether there is inclusion by examination of the initial two digits chosen which takes care of any overcounting that results from iteration steps (i, k, j) that are enumerated in both of these cases.

4.3 Computing B miss

In this subsection we finish the analysis of misses by computing **B miss**. The quantity **B miss** counts the number of interation points (i, k, j) at which the matrix element $B_{k,j}$ is not in cache, having been there previously.

If $B_{k,j}$ is in the same cache block as $B_{k,j-1}$, (Note: This case will arise if $\sigma_0 = 1$ and $\mu_2 + \Theta(k, j) \not\equiv 1, 2, 3 \mod 4$ or if $\sigma_1\sigma_0 = 10$ and $\mu_2 + \Theta(k, j) \equiv 2, 3 \mod 4$.) any collisions that forced $B_{k,j}$ out of cache must have occured at iteration step $(i, k, j-1)$ if the collision occurs with an element of C or at step (i, k, j) if the collision occurs with an element of A. Using arguments that are similar to those in previous cases, we see that these instances are enumerated by the **Extended AB Algorithm** and the **Extended AC Algorithm**. So we only need to examine the cases where $B_{k,j}$ and $B_{k,j-1}$ occupy different cache words. This analysis depends on the form of σ and so we need to consider different cases.

4.3.1 Computing B-A miss

We want to add one to **B-A miss** if there is an iteration step (x, z, y) with

$$(i, k-1, j+1) < (x, z, y) \leq (i, k, j)$$

for which the matrix entry $A_{x,z}$ occupies the same cache word as $B_{k,j}$. Note that $x = i$ and that $z = k - 1$ or $z = k$. We are free to choose y as long as we choose $y > j + 1$ when $z = k$ and $y \leq j$ when $z = k$.
$\underline{Case\ 1}$: $B_{k,j}$ is in the same cache word as $B_{k-1,j}$ but not in the same cache word as $B_{k,j-1}$. (Note: This case will arise if $\sigma_1\sigma_0 = 10$ and $\mu_2 + \Theta(k,j) \equiv 1 \mod 4$ or if $\sigma_1\sigma_0 \equiv 00$ and $\mu_2 + \Theta(k,j) \equiv 1, 2, 3 \mod 4$).

In this case, $B_{k,j}$ is brought into cache at iteration step $(i, k, j - 1)$. The enumeration of misses in this case is similar to previous cases.

Enumerating (i, k, j) which satisfy these conditions is equivalent to counting (i, k, j) which are solutions to:

$$\lfloor \frac{\mu_2 + \Theta(k,j)}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(i,k)}{4} \rfloor \mod 2^{\rho-2} \tag{41}$$

(we can then choose any $y \leq j$) OR which are solutions to:

$$\lfloor \frac{\mu_2 + \Theta(k,j)}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(i,k-1)}{4} \rfloor \mod 2^{\rho-2} \tag{42}$$

with $j < n - 1$ (we can then choose any $y > j + 1$). In doing so, we must be careful to count those (i, k, j) which satisfy both sets of equations only once. Fortunately, because we are in Case 2, we know that $\lfloor \frac{\mu_2+\Theta(k,j)}{4} \rfloor = \lfloor \frac{\mu_2+\Theta(k-1,j)}{4} \rfloor$. So we can replace equation (42) with an identical equation which has $k - 1$ in place of k on the left hand side. When this is done, equation (42) is identical to equation (41) with the variable $k' = k - 1$ in place of k. So, to count (i, k, j) which are solutions to at least one of equation (41) or equation (42), we can just count solutions to equation (41). The number of solutions to equation (41) can be computed as in previous cases.

$\underline{Case\ 2}$: $B_{k,j}$ is in a different cache block from both $B_{k-1,j}$ and $B_{k,j-1}$. (Note: This arises exactly when $\mu_2 + \Theta(k,j) \equiv 0 \mod 4$).

In this case, the most recent previous access of $B_{k,j}$ was at iteration step

$(i - 1, k + 3, j)$ if $\sigma_1\sigma_0 = 00$
$(i - 1, k + 1, j + 1)$ if $\sigma_1\sigma_0 = 01$ or 10
$(i - 1, k, j + 3)$ if $\sigma_1\sigma_0 = 11$.

We will consider just one of these possibilities – the others are handled in similar ways. Assume that $\sigma_1\sigma_0 = 10$.

The iteration step (i, k, j) contributes to **B-A miss** if there is an iteration step (x, z, y) with

$$(i - 1, k + 1, j + 1) < (x, z, y) \leq (i, k, j)$$

satisfying

$$\lfloor \frac{\mu_3 + \Theta(k,j)}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(x,z)}{4} \rfloor \mod 2^{\rho-2}$$

This is similar to the exceptional case for **C-B miss** where $\mu_3 + \Theta(i,j) \equiv 0 \mod 4$. We enumerate solutions in a similar way.

Note that **B-B miss** and **B-C miss** are computed in ways quite similar to **B-A miss**, dividing the analysis in the same Case 1 and Case 2. For **B-B miss**, we are counting triples (i, k, j) such that the array element $B_{k,j}$ was in cache but was removed because array element $B_{z,y}$ took its place in cache. For **B-C miss**, we are counting triples (i, k, j) such that $B_{k,j}$ was displaced from cache by $C_{x,y}$.

This completes the enumeration of cache misses.

5 $\mathbb{A}$-way Associative Cache

In this section, we indicate the changes needed to generalize our enumeration of cache misses from direct mapped cache to the case of an $\mathbb{A}$-*way associative* cache (Figure 2). In this case, memory location M is mapped to the cache set $\Lambda = \lfloor \frac{M}{4} \rfloor \mod p_{\mathbb{A}}$, where $p_{\mathbb{A}} = \frac{p}{\mathbb{A}}$ is the number of cache sets. Λ contains $\mathbb{A}$ cache blocks (each consisting of four memory locations, as explained in Section 3) that are filled according to either the first-in, first-out (FIFO) protocol, the least recently used (LRU) protocol or random fill [6]. LRU gives the best performance but is usually the most difficult to describe. We will show the analysis given the LRU protocol.

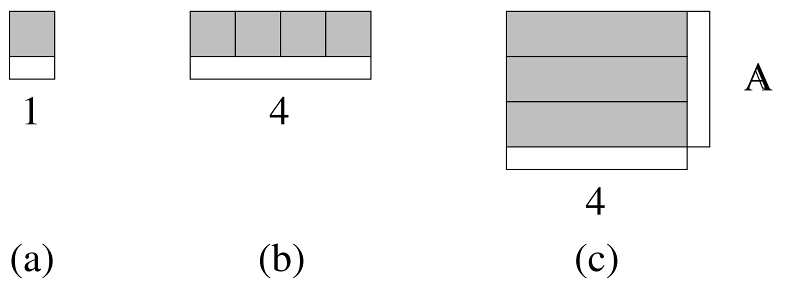

Figure 2: (a) cache word (b) cache block, size 4 (c) cache set, $\mathbb{A}$=3

Assuming the LRU protocol, a cache block is evicted on a cache miss when its last access lies furthest back in time. I.e., in our framework, we must enumerate instances where a matrix element X is accessed and brought into the cache set Λ, and where at least $\mathbb{A}$ times, since the previous access of X, different matrix elements are accessed that are not in cache and which are mapped to the same cache set Λ. We will use the term *collisions* for such instances and call these instances *collisions with* X. For more specificity, we will characterize collisions according to what kinds of array elements are involved. So, we will talk about **C-A collisions** meaning instances when an array element from A is brought into the same cache set as an element of C between consecutive accesses of that element of C. The relationship between collisions and cache misses is straightforward - when we access a matrix element X, we will have a cache miss if there have been greater than $\mathbb{A}$ collisions with X since the previous access. Thus, misses constitute a subset of collisions.

In the following, we will show the analysis for **C collisions**. According to our strategy, we will enumerate iteration steps (i, k, j) according to the number of collisions of type **C-A, C-B, C-C, C-AB, C-AC, C-BC**, and **C-ABC** that have occurred between the access of $C_{i,j}$ at iterations step $(i, k-1, j+\tau)$ and (i, k, j).

The considerations that go into enumeration of collisions will be very similar the considerations that went into the enumeration of cache misses in the direct mapped case, but the general enumeration framework will be somewhat more challenging. Instead of dividing the analysis according to which matrix is being accessed, we will divide the analysis according to the number of iteration steps since the most recent access of the matrix element under consideration.

Consider the situation where we access a matrix element X at iteration step (i, k, j). At this point, we will not yet specify which of the arrays A, B or C that X comes from.

$\underline{Case\ 6.1}$: The matrix element X was last accessed at iteration step $(i, k, j-1)$.

In this case, the access of X at iteration step (i, k, j) can only cause a cache miss for $L = 1, 2$. This case can be handled using arguments from the previous section. Note that this case includes all **A misses** and a subset of the **C misses**.

<u>$Case\ 6.2$</u>: The most recent access of X (prior to iteration step (i, k, j)) was at iteration step $(i, k-1, j+\tau)$ for some τ.

At this point, it is necessary to consider which of the arrays X comes from.

Consider the problem of determining whether there is a cache miss with an $\mathbb{A}$-way associative cache when $C_{i,j}$ is accessed at iteration step (i, k, j). For the next few paragraphs, it is important to keep in mind that i, k, j are fixed. We are going to try to find conditions on i, k, j under which there will be at least $\mathbb{A}$ collisions with $C_{i,j}$ between iteration steps $(i, k-1, j+\tau)$ and (i, k, j).

For C-A collisions, let α be the number of distinct $A_{x,z}$ which occupy the same cache set as $C_{i,j}$ and which are accessed between steps $(i, k-1, j+\tau)$ and (i, k, j). By that latter condition, we must have $x = i$ and $z \in \{k-1, k\}$. So, $\alpha = 0, 1, 2$ depending on whether neither, one of, or both of $u = k-1$ and $u = k$ give solutions to:

$$\lfloor \frac{\mu_3 + \Theta(i, j)}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(i, u)}{4} \rfloor \mod 2^\rho \tag{43}$$

For **C-B collisions**, let β be the number of distinct $B_{z,y}$ which occupy the same cache set as $C_{i,j}$ and which are accessed between steps $(i, k-1, j+\tau)$ and (i, k, j). By that latter condition, we must have $x = i$ and $z \in \{k-1, k\}$. To occupy the same cache set as $C_{i,j}$ we must have

$$\lfloor \frac{\mu_3 + \Theta(i, j)}{4} \rfloor = \lfloor \frac{\mu_2 + \Theta(z, y)}{4} \rfloor \mod 2^\rho \tag{44}$$

Finally for **C-C collisions**, let γ be the number of distinct $C_{x,y}$ which occupy the same cache set as $C_{i,j}$ and which are accessed between steps $(i, k-1, j+\tau)$ and (i, k, j). By the latter condition, we must have $x = i$. To occupy the same cache set as $C_{i,j}$ we must have

$$\lfloor \frac{\mu_3 + \Theta(i, j)}{4} \rfloor = \lfloor \frac{\mu_3 + \Theta(x, y)}{4} \rfloor \mod 2^\rho \tag{45}$$

Since $x = i$, equation (44) is equivalent to

$$\lfloor \frac{\mu_3 + \Theta(i, j)}{4} \rfloor = \lfloor \frac{\mu_3 + \Theta(i, y)}{4} \rfloor \mod 2^\rho \tag{46}$$

Before diving into details, it is worth discussing the broad outlines of the enumeration method that we follow. Our immediate goal is to enumerate **C misses** with an $\mathbb{A}$-way associative cache. More precisely, we want to count iteration steps (i, k, j) where the most recent prior access of $C_{i,j}$ was at iteration step $(i, k-1, j+\tau)$ and where there have been at least $\mathbb{A}$ distinct matrix elements X inserted into the same cache set as $C_{i,j}$ between iteration steps $(i, k-1, j+\tau)$ and (i, k, j).

The solutions to equation (43) characterize those $X = A_{i,u}$ which collide with $C_{i,j}$, the solutions to equation (44) characterize those $B_{z,y}$ which collide with $C_{i,j}$ and the solutions to equation (46) characterize those $C_{i,y}$ which collide with $C_{i,j}$, all collisions occuring between iteration steps $(i, k-1, j+\tau)$ and (i, k, j).

For any fixed (i, k, j) there can be at most two **C-A collisions** because solutions to equation (43) determine (i, k, j). The collisions will occur when $A_{i,u}$ is inserted into the same cache set as $C_{i,j}$. Since u must be either k or $k-1$, there can be at most two such collisions. There are two collisions if $(i, k-1, j)$ and (i, k, j) are simultaneously solutions to equation (43) and so we will have to enumerate such instances.

The treatment of **C-B collisions** is more complicated. The number of collisions for a fixed (i, k, j) is the number of solutions to equation (44) with $z = k$ and $y \leq j$ plus the number of solutions to equation (44) with $z = (k-1)$ and $y > j$. If $2m \leq \rho$ then z and y are completely determined by equation (44) and so the total number of **C-B collisions** for a fixed (i, k, j) will be at most two. However, if $2m > \rho$ then you must consider the number of ways you can extend $mod\ 2^\rho$ solutions i, j, z, y of equation (44). You have unrestricted choice of extensions for i, z thus creating $2^{2 \cdot E_0}$ distinct choices for i, k. For each choice of extension of j you must count to extensions of y so that $y \leq j$ if $z = k$ or $y > j$ if $z = (k-1)$. The number of such extensions of y will dictate the number of **C-B collisions** for this particular (i, k, j). A crucial consideration is whether (i, k, j) and $(i, k-1, j)$ are simultaneously solutions to equation (44). If so, any extension of y will create a **C-B collision** without any consideration of how y compares to j. So, we will need an algorithm to determine the number of iteration steps (i, k, j) for which there are simultaneous solutions to equation (44) with $z = k$ and $z = k - 1$.

Considerations of extensions also come into play when counting **C-C misses**. In this case, k is arbitrary so whatever enumeration of collisions we do for a fixed i, j will hold for all iteration steps of the form (i, k, j). Again, equation (46) determines y (in terms of i, j) mod 2^ρ. We can then extend i, j, y without restriction in digits ρ to $2m-1$. Different extensions of i, j give different iteration steps (i, k, j) (again, k is free to take on any value). However, different extensions of y give multiple **C-C collisions** at the iteration step (i, k, j).

This gives a framework for the enumeration. The method will utilize the technology we've already developed, with a couple of simple extensions, to enumerate solutions to equation (43), equation (44), and equation (46). If $2m \leq \rho$ then this analysis follows closely the analysis of cache misses in the direct mapped case done in Section 4.

So we will focus on the case where $2m > \rho$ where there are considerations not previously encountered. In this case, we must consider extensions of solutions to binary digits ρ and beyond. These extensions sometimes expand the number of iteration points (i, k, j) and sometimes expand the number of collisions per iteration point.

When this analysis is complete, we will have counted **C-A**, **C-B** and **C-C collisions** separately. We must then indicate how to count iteration points where there are simultaneous **C-A**, **C-B** and **C-C collisions**. We begin with two technical lemmas that will be key to our analysis.

Lemma 5.1 *There is an algorithm, ALGORITHM D1, that counts the number of triples (i, z, j) such that*

$$\lfloor \frac{\mu_3 + \Theta(i,j)}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(i,z)}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(i,z+1)}{4} \rfloor \mod 2^\rho \tag{47}$$

Moreover, this algorithm has the same complexity as the AC ALGORITHM.

The algorithm proceeds loops over possible first two digits of $\Theta(i, j)$ and $\Theta(i, z)$. For each such choice, the algorithm computes if there is a contribution to the total count and proceeds to the next step in the loop. The complete proof is shown in Appendix A.1.

There is a second situation, similar in nature, in which we will need to count instances where two solutions differ by just one in one of the variables.

Lemma 5.2 *There is an algorithm, ALGORITHM D2, which counts the number of triples $(i, k, j) \in B_m^3$ such that there are simultaneous solutions $v, x \in B_m$ to:*

$$\lfloor \frac{\mu_3 + \Theta(i,j)}{4} \rfloor = \lfloor \frac{\mu_2 + \Theta(k,v)}{4} \rfloor = \lfloor \frac{\mu_2 + \Theta(k-1,x)}{4} \rfloor \tag{48}$$

In addition, this algorithm will determine the number of solutions to equation (48) which satisfy $v \le j < x$. The complexity of this algorithm is $O(\rho)$.

The complete proof is somewhat lengthy. It can be found in Appendix A.2.

We are now ready to enumerate cache misses with an $\mathbb{A}$-way associative cache using the strategy outlined above. We introduce two more pieces of notation to ease discussion. First, let E_0 and E_1 be the number of σ_i with $i \ge \rho$ which are equal to 0 and 1 respectively. Let $E = E_0 + E_1$. Note that $E = max\{2m - \rho, 0\}$. Second, when referring to a variable that occurs in one of the equations $equation\ (43) - equation\ (46)$ we will use $\overline{r}$ to denote the digits in the variable r that occur in the equation taken mod 2^ρ and $(p, q, r)_{2^\rho}$ to denote that all variables in the tuple should be taken mod 2^ρ, i.e. triple $(p, q, r)_{2^\rho}$ contains $\overline{p}, \overline{q}$ and $\overline{r}$. Note that $\overline{r} = r$ if $E = 0$. Let us first enumerate **C-A collisions**.

5.1 Enumeration of C-A collisions

$\underline{Step\ 1}$: Using the methods from Sections 2–4, determine NS, the number of triples $(i, j, u)_{2^\rho}$ which satisfy equation (43). Using ALGORITHM D1 from Lemma 5.1, determine ND, the number of triples $(i, j, u)_{2^\rho}$ such that $(i, j, u)_{2^\rho}$ and $(i, j, u-1)_{2^\rho}$ simultaneously satisfy equation (43).

$\underline{Step\ 2}$: There are $(NS - ND) \cdot 2^{E_0+2E_1}$ iteration steps (i, k, j) at which there has been a single **C-A collision** since the previous access of $C_{i,j}$. There are $ND \cdot 2^{E_0+2E_1}$ iteration points (i, k, j) at which there have been two **C-A collisions** since the previous access of $C_{i,j}$.

5.2 Enumeration of C-B collisions

This is the far more interesting case because elements of both B and C ($C[i, j]$ and $B[k, j]$) are less "well behaved" than those $A[i, k]$ of array A. Thus, subsequently we show the approach in full length.

$\underline{Step\ 1}$: Using the methods from Sections 2–4, determine NS, the number of triples $T = (i, j, z)_{2^\rho}$ having the property that there is a $\overline{y}$ such that $(i, j, z, y)_{2^\rho}$ satisfies equation (44). To each such triple T we attach a multiplicity $m[T]$, this being the number of $\overline{y}$. Equation equation (44) almost completely determine $\overline{y}$ however this multiplicity may arise if there is more than one choice of initial digits for $\overline{y}$ which give the same carry in $\mu_2 + \Theta(z, y)$ from digits $0, 1$ to digit 2. Using ALGORITHM D2 from Lemma 5.2, determine ND, the number of triples $(i, j, k)_{2^\rho}$ such that there are simultaneous solutions $(i, j, k, u)_{2^\rho}$ and $(i, j, k-1, x)_{2^\rho}$ to equation (44). Also, using ALGORITHM D2, determine ND1, the number of triples $(i, j, k)_{2^\rho}$ such that there are simultaneous solutions $(i, j, k, u)_{2^\rho}$ and $(i, j, k-1, x)_{2^\rho}$ to equation (44) with $\overline{x} \le j < \overline{u}$. Again, we will attach a multiplicity to each of these solutions.

$\underline{Step\ 2}$: The next step differs significantly depending on whether $E > 0$ or $E = 0$, i.e whether $2m \leqslant \rho$ or $2m > \rho$. We divide into those two cases, of which the latter is the more interesting.

Case 1: E = 0

In this case, our enumeration is straightforward. There are $ND1$ iteration points (i, k, j) in which there are two collisions of the forms $\gamma_1 = (i, k, j, u)$ and $\gamma_2 = (i, k-1, j, x)$. Each of γ_1 and γ_2 must be counted according to its multiplicity. There are $NS - 2 \cdot ND1$ iteration points where there has been a unique collision (which must be counted with multiplicity).

Case 2: E $>$ 0:

In this case, the enumeration is more challenging. Consider a solution $T = (i, z, j)_{2^\rho}$ counted in the number NS. It is enumerated because there exists a $\overline{y}$ such that $(i, j, z, y)_{2^\rho}$ is a solution to equation (44). Let

$T_{2^\rho} = (i, z-1, j)_{2^\rho}$. Assume first that T_{2^ρ} is not also a solution to equation (44). Let (i, k, j) be any triple of numbers that extend T. Any extension of $\overline{y}$ will count a collision that occurs when $B_{k,j}$ is accessed at iteration step (i, k, y) so long as $y \leq j$. Let $\phi(j) = \lfloor \frac{j}{2^\rho} \rfloor$ and let $\phi(y) = \lfloor \frac{y}{2^\rho} \rfloor$. If $\phi(y) < \phi(j)$, then $y < j$. If $\phi(y) > \phi(j)$ then $y > j$. If $\phi(y) = \phi(j)$, then $y \leq j$ iff $\overline{y} \leq \overline{j}$. So, $\phi(j)$ is an estimate for the number of collisions that is correct to within one.

On the other hand, the extension of y is arbitrary. So, for every solution $T = (i, z, j)_{2^\rho}$ to equation (44) counted by NS which is not counted by ND, and for every choice of $\phi \in \{0, 1, \ldots, 2^{E_1} - 1\}$ there are $2^{2 \cdot E_0}$ iteration steps (i, k, j) for which the number of **C-B collisions** is ϕ times the multiplicity of the triples, and this estimate is correct to within the multiplicity.

Assume now that T_{2^ρ} is also a solution to equation (44). Such pairs (T, T_{2^ρ}) are enumerated by ALGORITHM D2: their contribution to the analysis above must be subtracted out as a first step. In this case, every one of the $2^{(2E_0+E_1)}$ extensions (i, k, j) of $T = (i, z, j)_{2^\rho}$ is an iteration step at which there have been $m[T] \cdot 2^{E_1}$ collisions between the access of $C_{i,j}$ at iteration step $(i, k-1, j+\tau)$ and current access.

The reasoning is as follows. Let $(i, z, j, u)_{2^\rho}$ and $(i, z-1, j, x)_{2^\rho}$ be the simultaneous solutions to equation (48). There are $2^{(2E_0+E_1)}$ extensions of $(i, z, j)_{2^\rho}$ to a triple (i, k, j). Let $\phi \in B_{E_1}$. If $\phi \leq \phi(j)$ then we assign ϕ to be an extension of $\overline{u}$ to a $u \leq j$ so that there is a collision when $B_{k,u}$ is accessed at iteration step (i, k, u). If $\phi > \phi(j)$ then we assign ϕ to be an extension of $s\overline{x}$ to an $x > j$ so that there is a collision when $B_{k-1,x}$ is accessed at iteration step $(i, k-1, x)$.

5.3 Enumeration of C-C collisions

Let NS be the number of solutions to equation (46) which we can compute using the methods in Sections 2-4. For every choice of solution $(i, j, y)_{2^\rho}$ to equation (46) there are $2^{E_0+E_1}$ ways to extend $\overline{i}, \overline{j}$ to $i, j \in B_m$. For each such pair of extensions there are 2^m ways to choose a k to complete the determination of the iteration step (i, k, j). Then every one of the 2^{E_1} possible extensions of $\overline{y}$ to $y \in B_m$ indexes a collision between $C_{i,y}$ and $C_{i,j}$ that occurs between the access of $C_{i,j}$ at iteration step $(i, k-1, j+\tau)$ and the access at iteration step (i, k, j). If $y < j$ then the collision occurs at the iteration step (i, k, y), whereas if $y > j$ then the collision occurs at iteration step $(i, k-1, y)$. There is one exception to this analysis. Clearly $\overline{y} = \overline{j}$ is a solution to equation (46) if and only if the choice of initial digits for $\overline{y}$ and $\overline{j}$ are identical. So, it is straightforward to compute R, the number of solutions to equation (46) in which $\overline{y} = \overline{j}$. The significance of these R solutions,is that if $\overline{y} = \overline{j}$, then $y = j$ is a possible extension of $\overline{y}$ in which case, the collision we count above is not genuine.

To summarize, there are $NS \cdot 2^{E_0+E_1+m}$ iteration points where there have been **C-C collisions**. Of these, in $R \cdot 2^{E_0+E_1+m}$ cases there have been $2^{E_1} - 1$ collisions and in $(NS - R) \cdot 2^{E_0+E_1+m}$ there have been 2^{E_1} collisions.

The following chart summarizes the analysis of **C-A**, **C-B**, and **C-C** collisions above.

Type	**# of iteration pts**	**# of Collisions**	
$C-A$	$NS_{equation\ (43)} \cdot 2^{E_0+2 \cdot E_1}$	1 or 2	
$C-B_1$	$(NS_{equation\ (44)} - 2 \cdot NS_{equation\ (48)}) \cdot 2^{2 \cdot E_0} \cdot (\#\phi < 2^{E-1})$	ϕ	$z = k$
		$2^{E_1} - \phi$	$z = k-1$
$C-B_2$	$NS_{equation\ (48)} \cdot 2^{2 \cdot E_0+E_1}$	2^{E_1}	
$C-C$	$NS \cdot 2^{E_0+E_1+m}$	2^{E_1} or $(2^{E_1} - 1)$	

It remains to enumerate iteration points that fall into more than one of those categories. To understand the need for this, assume for example that $2^{E_1} < \mathbb{A} < 2^{2 \cdot E_1}$. Then no iteration point would exhibit more than $\mathbb{A}$ collisions of a single type **C-A**, **C-B** or **C-C**. However, if there exists and iteration point that is simultaneously of type **C-B**$_2$ and **C-C**, then there would be at least $2^{2 \cdot E_1} > \mathbb{A}$ collisions between the access of $C_{i,j}$ at $(i, k-1, j+\tau)$ and the access at (i, k, j). So there would be a **C miss** at (i, k, j) with an $\mathbb{A}$-way associative cache.

The way we proceed is largely similar to what we have already shown in Section 4 for the direct mapped case. The lengthy analysis shown in Appendix A.3 yields a method to enumerate iteration points by number of **C collisions**. Using this method, we can determine $\phi(C, t)$, the number of iteration points (i, k, j) for which there have been exactly t collisions with $C_{i,j}$ between the prior access of $C_{i,j}$ and the access at (i, k, j). Assuming an $\mathbb{A}$-way associative cache with a LRU protocol the number of **C misses** is $\sum_{t \geq \mathbb{A}} \phi(C, t)$.

At this point we have indicated how to enumerate **A misses** and **C misses** in the case of an $\mathbb{A}$-way associative cache. It remains to enumerate **B misses**. Since the technical difficulties we encounter, as well as the ideas we use to overcome these difficulties, are similar to those seen in the enumeration of **C misses** we leave details to the reader.

The extension to first in first out (FIFO) replacement is straightforward. Here, the requirement that the accessed matrix elements are *different* is dropped in the definition of a collision.

6 Conclusions

This paper introduced a class of array layouts, *interleavings*, and efficient algorithms to exactly assess the number of cache misses caused by such layouts when used in the context of matrix multiplication. The layouts are described by bit-level address manipulations, and cache misses are counted by reasoning about the solutions to simple bit-level equations. Most importantly, we achieve a reduction in complexity from $\mathcal{O}(2^m + \rho)$ to $\mathcal{O}(\max(m, \rho))$ with respect to the naive algorithm by exploiting properties of carry propagation. Although there are various subcases in the analysis of cache misses, each case can be ultimately reduced to one of two combinatorial enumeration problems.

A particular strength of our techniques is that it explicitly handles cross interference between arrays, which is generally considered to be difficult to handle. Also, our model allows an elegant extension to a set-associative cache with LRU replacement strategy.

Our current work has several limitations. First, we have thus far provided an analysis only of matrix multiplication, and for $2^m \times 2^m$ matrices at that. It seems likely that the ideas can be generalized to handle other computations, but this remains to be demonstrated. Second, a number of special cases arise in dealing with the least significant bits of σ that are truncated when converting a memory address to a block address. Our restriction to a cache block size of four elements required us to handle only two bits of σ, but the problem could be more acute for larger block sizes (*e.g.*, in analyzing TLB behavior). Finally, our use of inclusion-exclusion poses the imminent danger of combinatorical explosion when the interaction of many arrays has to be calculated. However, as mentioned in Section 2.3, this case can be adequately approximated as many of these intersections are empty or sparse.

Our immediate future work will tackle the optimization problem of determining layout functions that minimize the number of cache misses for matrix multiplication. There are also related problems—such as counting compulsory misses, differentiating capacity and compulsory misses, and identifying cache contents at the end of executing a loop nest—for which efficient algorithms remain to be found.

A Proofs and Details for Section 5

A.1 Proof of Lemma 5.1

Proof: First, the algorithm will loop over the possible first two digits of $\Theta(i, j)$. As before in Section 3, we will let $\mu'_3 + \Theta'(i', j')$ denote the part of $\mu_3 + \Theta(i, j)$ in digits 2 to $(\rho - 1)$ where the carry from digits $0, 1$ are incorporated into μ'_3.

Similarly, loop over the possible first two digits of $\Theta(i, z)$ (some of which might have already been fixed because i is common to $\Theta(i, j)$ and $\Theta(i, z)$. When $\Theta(i, z + 1)$ is computed from $\Theta(i, z)$ there will be some carry $\epsilon \in B_2$ from the part of $z + 1$ that occurs in the first two digits of $\Theta(i, z + 1)$ to the part of $z + 1$ that occurs in the digits $2 - (\rho - 1)$ of $\Theta(i, z + 1)$. Note that ϵ is determined by the first two digits of $\Theta(i, z)$ that we are looping over. Lastly, let γ_1 and γ_2 be the carry from digit one to digit two in $\mu_1 + \Theta(i, z)$ and $\mu_1 + \Theta(i, z + 1)$ respectively. We use the prime notation from Section 4 to denote digits $2 - (\rho - 1)$. Combining all this we have

$$\mu'_3 + \Theta'(i', j') = \mu'_1 + \Theta'(i', z') + \gamma_1 = \mu'_1 + \Theta'(i', z' + \epsilon) + \gamma_2 \tag{49}$$

The equalities in the equation above are modulo $2^{\rho-2}$ but still it is clear that the only possible ways in which they can be realized are if

1) $\gamma_1 = \gamma_2 = \epsilon = 0$
or
2) $\gamma_1 = \epsilon = 1, \gamma_2 = 0$ and $\sigma_2 = 1$.

So the algorithm proceeds in the following way. It loops over possible first two digits of $\Theta(i, j)$ and $\Theta(i, z)$. For each such choice, the algorithm computes γ_1, γ_2 and ϵ. If neither $1)$ or $2)$ above is satisfied, then there is no contribution to the total count and the algorithm proceeds to the next step in the loop. If either $1)$ and $2)$ above is satisfied, then the algorithm computes the number of solutions to the equation:

$$\mu'_3 + \Theta'(i', j') = \mu'_1 + \Theta'(i', z') \tag{50}$$

using the AC ALGORITHM and adds that number to the total.

A.2 Proof of Lemma 5.2

Proof: We will need some terminology and notation to explain this algorithm. Let I be the number of 0's in the set $\{\sigma_0, \sigma_1\}$. The *initial digits* of either k or $k - 1$ will refer to the first I, i.e., those that appear in the first two digits of $\Theta(k, v)$ and $\Theta(k - 1, x)$. Let ν be the minimal index greater than 1 with $\sigma_\nu = 0$. So we have $\sigma_2 = \sigma_3 = \cdots = \sigma_{\nu-1} = 1$.

The first step in this algorithm is to loop over choices for the initial digits of $k - 1$ (which will also determine the initial digits of k. Let $\tau \cdot 2^I$ be the carry when 1 is added to the initial digits of $k - 1$. Note that τ is carried to the ν^{th} binary digit when $\Theta(k, v)$ is computed from $\Theta(k - 1, v)$.

The next step is to loop over possible carries ϵ_0, ϵ_1 and ϵ_2 from the zero and first binary digits to the second binary digit in $\mu_3 + \Theta(i, j)$, $\mu_2 + \Theta(k, v)$ and $\mu_2 + \Theta(k - 1, x)$ respectively. We compute the number $N_0 N_1 N_2$ where N_0 is the number of choices for the zero and first binary digits of $\mu_3 + \Theta(i, j)$ which will result in a carry of ϵ_0, where N_1 is the number of choices of initial digits of v and x respectively which will result in carries of ϵ_1 and ϵ_2 respectively (given the choices we've already made for initial digits in $k-1$ and k).

With this notation and the prime notation we can express equations equation (48) as:

$$\mu_3' + \Theta'(i', j') + \epsilon_0 = \mu_2' + \Theta'(k', v') + \epsilon_1 = \mu_2' + \Theta'((k-1)', x') + \epsilon_2. \tag{51}$$

So,

$$\Theta'((k-1)', v') + \epsilon_1 + \tau \cdot 2^{\nu-2} = \Theta'((k-1)', x') + \epsilon_2. \tag{52}$$

Rewriting equation (52) we obtain,

$$(\epsilon_1 - \epsilon_2) + \tau \cdot 2^{\nu-2} = \Theta'((k-1)', x') - \Theta'((k-1)', v'). \tag{53}$$

Let $\hat{v}$ and $\hat{x}$ denote v' and x' taken mod $2^{\nu-2}$. We note that the right-hand side of equation (53) is equal to

$$\hat{x} - \hat{v} + GLOB$$

where $GLOB$ is a multiple of $2^{\nu-1}$. Since the left-hand side of equation (53) is strictly less than $2^{\nu-1}$, we deduce that $GLOB = 0$ and so

$$\epsilon_1 - \epsilon_2 + \tau \cdot 2^{\nu-2} = \hat{x} - \hat{v}. \tag{54}$$

$\underline{Case\ 1}$: $\tau = 0$.

In this case, equation (54) becomes

$$\epsilon_1 + \hat{v} = \epsilon_2 + \hat{x}. \tag{55}$$

Also, $\epsilon_1 + \hat{v} = \epsilon_2 + \hat{x}$ is completely determined by the equality equation (51). So there is exactly one choice of $\hat{v}$ and $\hat{x}$ in this case for every i, j. So in this case there are $min\{2^{\rho-2}, 2^{2m-2}\}$ choices for i, j. For each such choice there is exactly one choice of k. For this triple (i, k, j) there are $N_0 \cdot N_1 \cdot N_2$ choices of $B_{w,z}$ that collide with $C_{i,j}$ between its access at iteration steps $(i, k-1, j+\tau)$ and (i, k, j).

It remains to determine the number of these collisions satisfying $v \le j < x$. The reader will note that

$$\epsilon_1 + v' = \epsilon_2 + x'$$

so there are no such j unless $\epsilon_1 = 1, \epsilon_2 = 0$ and in this case any j with $v \le j < x$ must satisfy

$$v' = j' < x' = v' + 1. \tag{56}$$

The first thing to check is whether the choices of initial digits for j, x, v would mean that j', x', v' which satisfy equation (56) would give j, x, v with $v \le j < x$. If not, then there are no such j. If so, we enumerate j', x', v' by enumerating solutions to:

$$\mu_3' + \Theta'(i', j') = \mu_2' + \Theta'(k', j')$$

using the methods developed in Section 2.

$\underline{Case\ 2}$: $\tau = 1$.

In this case equation (54) is equivalent to:

$$\hat{v} + \epsilon_1 = \hat{x} + \epsilon_2 + 2^{\nu}. \tag{57}$$

Recalling that $\hat{v}, \hat{x} \in B_\nu$, we see that equation (57) can have a solution only if $\hat{x} = 0, \epsilon_2 = 0, \epsilon_1 = 1$ and $\hat{v} = 2^\nu - 1$. In order for these choices of $\hat{x}$, $\hat{v}$, ϵ_1 and ϵ_2 to satisfy equation (51) we must have that $\mu_3' + j' + \epsilon_0$

agrees with μ_2' in digits $0-(\nu-3)$. This implies that digits $0-(\mu-3)$ in j' are determined by μ_3, ϵ_0 and μ_2. So there are $min\{2^{\rho-\nu}, 2^{2m-\nu}\}$ choices for such i, j. Each determines a unique k and $N_0 \cdot N_1 \cdot N_2$ pairs w, z such that $B_{w,z}$ collides with $C_{i,j}$ between the access of $C_{i,j}$ at iteration steps $(i, k-1, j+\tau)$ and (i, k, j).

Lastly, we need to determine the number of these solutions which satisfy $v \le j < x$. However, in this case, $v > x$ and so there are no such j.

This completes the proof of Lemma 5.2 and the construction of ALGORITHM D2.

A.3 Enumeration of iteration points that exhibit simultaneous collisions

We show the case of the enumeration of iteration points that exhibit simultaneous C-A and C-B collisions and thus seek to enumerate iteration points (i, k, j) such that there have been both **C-A** and **C-B collisions** between $(i, k-1, j+\tau)$ and (i, k, j). We begin by enumeration simultaneous solutions to equation (43) and equation (44) but with the added factor that either $u = z$, $u+1 = z$ or $u = z+1$. We split our analysis into those three cases.

We will consider only the case where $u = z$ here - the other cases can be handled via similar methods. In the case $u = z$, a simultaneous solution to equation (43) and equation (44) must satisfy:

$$\lfloor \frac{\mu_3 + \Theta(i,j)_{2^\rho}}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(i,z)_{2^\rho}}{4} \rfloor = \lfloor \frac{\mu_2 + \Theta(z,y)_{2^\rho}}{4} \rfloor \tag{58}$$

We enumerate solutions to equation (58) by first counting solutions $\overline{j}, \overline{z}$ to the left-most equality using the AB-Algorithm, but keeping the choice of $\overline{i}$ open for the moment. For each such solution $\overline{j}, \overline{z}$, there is one and only one choice of $\overline{i}$ and $\overline{y}$ that satisfies the second equality.

If $E = 0$, then i, j, z and y are determined at this point. Also, k is determined, as $k = z$ if $y \le j$ and $k = z+1$ if $y > j$. For this iteration point (i, k, j), the number of collisions will be $2, 3$ or 4. The normal case will be 2 but 3 or 4 may result if there are two **C-A collisions** or two **C-B collisions** (or both). We will discuss these cases below.

If $E > 0$ then we must extend $\overline{i}, \overline{j}, \overline{z}$ and $\overline{y}$ to get i, j, z and y. The consideration on extensions is identical to those above. In particular, k will be either z or $z+1$ depending on how the extension of y compares to the extension on j. In cases where $\overline{z}$ and $\overline{z+1}$ are not both solutions to equation (44) then the number of collisions will be $1+\phi(j)$ or $2+\phi(j)$ for $k = z$ depending on whether there are multiple **C-A collisions** (which is disussed below). The number of collisions will be $1+2^{E_1}-\phi(j)$ or $2+2^{E_1}-\phi(j)$ for $k = z+1$ depending on whether there are multiple **C-A collisions**. However, if $\overline{z}$ and $\overline{z+1}$ are both solutions to equation (44) then the number of collisions will be either $1+2^{E_1}$ or $2+2^{E_1}$ depending on whether there are multiple **C-A collisions**.

So we will need to enumerate simultaneous solutions to equation (47) and equation (44), equation (43) and equation (48), equation (47) and equation (48). These enumeration problems can be solved using the tools we have already developed and applied to counting solutions to equation (43) and equation (44). So, we will omit many of the details in our account of how to proceed.

To enumerate simultaneous solutions to equation (47) and equation (44), we begin by counting solutions to

$$\lfloor \frac{\mu_3 + \Theta(i,j)_{2^\rho}}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(i,z)_{2^\rho}}{4} \rfloor = \lfloor \frac{\mu_1 + \Theta(i,z+1_{2^\rho})}{4} \rfloor = \lfloor \frac{\mu_2 + \Theta(z,y)_{2^\rho}}{4} \rfloor \tag{59}$$

To count solutions to this equation, use Lemma 5.1 to enumerate solutions $\overline{j}, \overline{z}$ to the first two equalitites leaving $\overline{i}$ undecided. With $\overline{j}, \overline{z}$ fixed there are unique choices for $\overline{i}$ and $\overline{y}$ which satisfy the third equality.

Each choice of $\overline{j}, \overline{z}, \overline{i}$, and $\overline{y}$ which satisfy equation (59) must now be extended. First, note that there are $2^{E_0+2\cdot E_1}$ ways to extend each solution $\overline{j}, \overline{z}, \overline{i}$. Observe that the extension of z must be $k-1$ because of the form of equations equation (59). For each such extension, there are $2+Y$ collisions of types $C-A$ and $C-B$ at iteration point (i,k,j) where Y is the number of extensions of $\overline{y}$. Since $z=k-1$, the Y is the number of extensions of $\overline{y}$ is $2^{E_1}-\phi(j)$ where $\phi(j)$ is the extension of j.

As noted in the previous paragraph, the form of equations equation (59) imply that $z=k-1$. So we must also solve a second set of equations which differ from equation (59) only in that the last $\overline{z}$ is replaced by $\overline{z+1}$. The enumeration of solutions to this set of equations follows the lines above with the only major difference being that there are $\phi(j)$ extensions possible for $\overline{y}$. This impacts the number of collisions that have occurred at iteration step (i,k,j).

To enumerate simultaneous solutions to equation (43) and equation (48), we must count solutions to the system of equations

$$\lfloor\frac{\mu_3+\Theta(i,j)_{2^\rho}}{4}\rfloor=\lfloor\frac{\mu_1+\Theta(i,z)_{2^\rho}}{4}\rfloor=\lfloor\frac{\mu_1+\Theta(z,y)_{2^\rho}}{4}\rfloor=\lfloor\frac{\mu_2+\Theta(z+1,x)_{2^\rho}}{4}\rfloor \tag{60}$$

To enumeration of solutions to equation (60), we use the AB Algorithm to count solutions to the second equality. The reasoning that went into the proof of Lemma 5.2 give (in each of two cases), the relationship between $\overline{y}$ and $\overline{x}$. The one complication that arises is in the case (from Lemma 5.2) where $\tau=1$. In this case, the first ν digits of y' are determined and so the first ν digits of $\overline{z}'$ will also be determined. This gives a partial determination of $\overline{z}$ which must be factored into the AB Algorithm as was done in Section 3.

Each solution of equation (60) must be extended. The number of extensions is straightforward to count in this case, since there are 2^{E_1} extensions of the pair $\overline{y}, \overline{x}$.

To complete this analysis, we must enumerate simultaneous solutions to equation (47) and equation (48). We begin by enumerating solutions to

$$\lfloor\frac{\mu_3+\Theta(i,j)_{2^\rho}}{4}\rfloor=\lfloor\frac{\mu_1+\Theta(i,z)_{2^\rho}}{4}\rfloor=\lfloor\frac{\mu_1+\Theta(i,z+1)_{2^\rho}}{4}\rfloor \tag{61}$$

$$=\lfloor\frac{\mu_1+\Theta(z,y)_{2^\rho}}{4}\rfloor=\lfloor\frac{\mu_2+\Theta(z+1,x)_{2^\rho}}{4}\rfloor.$$

To characterize solutions to equation (61), begin with the first two equalities. Following the reasoning in the proof of Lemma 5.1, we can eliminate some choices for initial digits of $\overline{j}, \overline{z}, \overline{i}, \overline{x}$, and $\overline{y}$. For those that are not eliminated, solutions of equation (61) are equivalent to solutions of equation (60) and so we can use the methods developed above to enumerate those solutions. As in other cases, once solutions to equation (61) are enumerated, they must be extended to give simultaneous solutions to equation (47) and equation (48). These extensions will then give the number of iteration points. At each there will be $2+2^{E_1}$ collisions between the access of $C_{i,j}$ at that iteration step and the most recent previous access.

This completes the enumeration of iteration steps where there have been both and **C-A** and **C-B collisions**. The next step is to enumerate iteration points where there have been both **C-A** and **C-C collisions**, iteration points where there have been both **C-B** and **C-C collisions**, and iteration points where there have been all three of **C-A**, **C-B** and **C-C collisions**. To shorten this exposition, we will only indicate where the modifications of the previous analyses come in.

To enumerate cases where there have been both **C-A** and **C-C collision**s is straightforward. Enumerate solutions $\overline{j}, \overline{u}$ of to equation (43) including the number for which both $\overline{j}, \overline{u}$ and $\overline{j}, \overline{u-1}$ are solutions. For each of these, there is at most one $\overline{y}$ such that $\overline{j}, \overline{y}$ satisfy equation (46). There will exist an $\overline{y}$ unless the carries from the initial two digits of the two sides of equation (46) are different. In this case you may have to eliminate one possible solutions to equation (43). This determination can be made by an $O(\rho)$ examination of μ_1 and μ_3. Once $\overline{j}, \overline{u}$ and $\overline{y}$ have been determined, we can choose i arbitrarily and we can arbitrarily extend $\overline{j}$, $\overline{u}$ and $\overline{y}$ to j, u and y. Different extensions of i, j and u lead to different iteration points. However, for fixed i, j and u, different extensions of $\overline{y}$ correspond to multiple collisions.

To enumerate cases where there have been both **C-B** and **C-C collisions** is likewise straightforward. First enumerate solutions to equation (46). For every solution $\overline{i}, \overline{j}, \overline{v}$, there is a uniquely determined solution to $\overline{z}, \overline{y}$ up to multiplicity $m[T]$ that might arise from differing initial digits in $\Theta(z, y)$. As before, we must determine whether different choices of initial digits might lead to both $\overline{z}$ and $\overline{z+1}$ being solutions. This is straightforward. Each solution $\overline{i}, \overline{j}, \overline{v}, \overline{z}, \overline{y}$ just enumerated must be extended. The issues related to extensions are identical to the issues that arose in the enumeration of **C-B collisions**. We leave details to the reader.

Lastly, we need to enumerate cases where there have been **C-A, C-B** and **C-C collisions**. We first enumerate iteration points where there have been **C-A** and **C-C collisions** as above. For every such solution, we can uniquely solve for $\overline{z}, \overline{y}$, uniquely up to choice of initial digits. As usual, consideration must be given to whether there $\overline{u}$ and $\overline{u-1}$ are both solutions to equation (43) and to whether $\overline{z}$ and $\overline{z+1}$ are both solutions to equation (44). There are no novel issues that arise around extensions and so again we leave details to the reader.

References

[1] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi. Nonlinear array layouts for hierarchical memory systems. In *Proceedings of the 1999 ACM International Conference on Supercomputing*, pages 444–453, Rhodes, Greece, June 1999.

[2] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 222–231, Saint-Malo, France, June 1999.

[3] P. Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–54, 1991.

[4] J. D. Frens and D. S. Wise. Auto-blocking matrix-multiplication or tracking BLAS3 performance with source code. In *Proceedings of the Sixth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 206–216, Las Vegas, NV, June 1997.

[5] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: A compiler framework for analyzing and tuning memory behavior. *ACM Trans. Prog. Lang. Syst.*, 21(4):703–746, July 1999.

[6] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2nd edition, 1996.

[7] M. D. Hill and A. J. Smith. Evaluating associativity in CPU caches. *IEEE Trans. Comput.*, C-38(12):1612–1630, Dec. 1989.

[8] M. S. Lam, E. E. Rothberg, and M. E. Wolf. The cache performance and optimizations of blocked algorithms. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 63–74, Apr. 1991.

[9] W. Pugh. The Omega test: A fast and practical integer programming algorithm for dependence analysis. *Commun. ACM*, pages 102–114, Aug. 1992.