Beyond Time Complexity: Data Movement Complexity Analysis for Matrix Multiplication (TECH REPORT)

Wesley Smith
Aidan Goldfarb
Chen Ding
wsmith6@cs.rochester.edu
agoldfa7@u.rochester.edu
cding@cs.rochester.edu
University of Rochester
Rochester, NY, USA

Abstract

Data movement is becoming the dominant contributor to the time and energy costs of computation across a wide range of application domains. However, time complexity is inadequate to analyze data movement. This work expands upon Data Movement Distance, a recently proposed framework for memory-aware algorithm analysis, by 1) demonstrating that its assumptions conform with microarchitectural trends, 2) applying it to four variants of matrix multiplication, and 3) showing it to be capable of asymptotically differentiating algorithms with the same time complexity but different memory behavior, as well as locality optimized vs. non-optimized versions of the same algorithm. In doing so, we attempt to bridge theory and practice by combining the operation count analysis used by asymptotic time complexity with per-operation data movement cost resulting from hierarchical memory structure. Additionally, this paper derives the first fully precise, fully analytical form of recursive matrix multiplication's miss ratio curve on LRU caching systems. Our results indicate that the Data Movement Distance framework is a powerful tool going forward for engineers and algorithm designers to understand the algorithmic implications of hierarchical memory.

CCS Concepts: • **Theory of computation** → **Design and analysis of algorithms**.

Keywords: matrix multiplication, hierarchical memory, algorithm analysis, data movement

ICS '22, June 28–30, 2022, Virtual Event, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9281-5/22/06…$15.00
https://doi.org/10.1145/3524059.3532395

ACM Reference Format:
Wesley Smith, Aidan Goldfarb, and Chen Ding. 2022. Beyond Time Complexity: Data Movement Complexity Analysis for Matrix Multiplication (TECH REPORT). In *2022 International Conference on Supercomputing (ICS '22), June 28–30, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3524059.3532395

1 INTRODUCTION

In exascale computing, the cost of data movement exceeds that of computation [6]: as such, data movement is a key factor in not only system performance but also in the computer science community's growing responsibility to address computing's role in the climate crisis.

Optimizing a program or a system for locality is difficult, as modern memory systems are large and complex. For portability, we should not program data movement directly, but we should be aware of its cost. However, there does not yet exist a single quantity that can characterize the effect of locality optimization at the program or algorithm level.

Standard techniques for understanding algorithm-hierarchy interactions, like miss ratio analysis, yield insight for algorithm designers with a target set of machine parameters in mind, but do not allow for general understanding of an algorithm's intrinsic data movement for an arbitrary target machine.

The actual effect of cache usage depends on all components of a program and also its running environment. Assumptions about a machine may be wrong, imprecise, or soon obsolete. A program may run on a remote computer in a public or commercial computing center with limited information about its memory system. Auto-tuning can select the best parameters for a given system, but it is difficult to tune if a system is shared.

In a recent position paper, Ding and Smith [7] defined an abstract measure of memory cost called Data Movement Distance (DMD). Memory complexity is measured by DMD in the same way time complexity is by operation count. They showed results for two types of data traversals with only constant factor differences in DMD.

This paper presents DMD analysis for different approaches to matrix multiplication. It differs from past work in several ways. First, unlike practical analysis, i.e. those based on miss ratios, DMD analysis is asymptotic and machine agnostic. Second, unlike I/O complexity, DMD analysis includes the effect of a cache hierarchy. In addition, the derivation is radically different from past solutions. For example, efficient algorithms often make use of temporaries that are dynamically allocated. They share cache, but the cache sharing is not analyzed by past complexity analysis. It is measured by practical analysis in concrete terms, i.e. cache misses, not asymptotic terms.

DMD measures data movement. Running time depends also on latency tolerance techniques, especially prefetching, which is outside the scope of this paper. Latency tolerance techniques, however, do not reduce the amount of data movement intrinsic to the algorithm. In addition, this paper targets sequential computation and assumes a memory hierarchy that is concentric, layered and managed using the least-recently-used (LRU) cache replacement policy.

2 MAIN CONTRIBUTIONS

The focus of this work is exploring the algorithmic implications of hierarchical memory systems by fleshing out the memory-aware algorithm analysis framework Data Movement Distance (DMD) introduced in [7]. The main contributions are as follows:

- Derivation and empirical validation of the first fully precise analytical form of recursive matrix multiplication's miss ratio on LRU caching systems,
- Application of the DMD framework for memory-aware algorithm analysis to four variants of matrix multiplication,
- Exploration of the effects of locality optimizations on the previous results,

In addition, we expand the motivation for and justification of the DMD framework by demonstrating the following:

- microarchitectural trends in cache memory conform with the framework's assumptions,
- DMD is capable of asymptotically differentiating algorithms with the same time complexity as a result of their memory behavior, as well as asymptotically differentiating between locality optimizations

Taken together, we believe that the results of our analyses indicate that the DMD framework is a powerful tool going forward for engineers and algorithm designers to understand the algorithmic implications of hierarchical memory.

3 BACKGROUND AND MOTIVATION

3.1 Locality Concepts

The locality concept most central to this work is *reuse distance* (RD) [27]. Reuse distance, or LRU stack distance [14], characterizes an individual memory access by counting the number of distinct memory locations accessed by the program between the most recent previous use of that memory location and the current use.

For example, let letters denote distinct memory locations in the following access trace:

$$abbca$$

In this example, the reuse distance of the second access to b is 1, as only b occurs in the window from position 2 to position 3, while the reuse distance of the second access to a is 3, as a, b, c all occur in the window from position 1 to position 5.

Reuse distance and miss ratio for fully-associative LRU cache are interconvertible, with their relation as follows:

$$MR(c) = P(rd > c)$$

Accesses with reuse distance greater than c are misses in LRU caches of size c or less. So, reuse distance distributions and miss ratio curves are the same information.

3.2 Data Movement Distance

The most ubiquitous technique for algorithm cost analysis is asymptotic time complexity, which measures operation count as a function of input size. However, on machines with hierarchical memory, execution cost will scale with input size *faster* than time complexity would indicate, because as data size increases, more program data must be stored in large, slow hierarchy components: the cost of data movement scales with input size as well. Data Movement Distance (DMD) is a novel framework for memory-aware algorithm analysis proposed by first Snyder and Ding [20] and then Ding and Smith [7] in which operation count is combined with per-access data movement cost by considering the algorithm's reuse distance distribution.

Because data movement cost varies across machines, the DMD framework includes an abstracted version of a memory hierarchy, termed the *geometric stack*, on which the behavior of algorithms is considered. The geometric stack can be understood to be an infinite-level memory hierarchy in which each level stores a single datum. The cost of accessing the datum at level n is $\sqrt{n}$. DMD for a program is then defined as follows:

Definition 1 (Data Movement Distance). *For a program p with data accesses a_i, let the reuse distance of a_i be d_i. The DMD for p under caching algorithm A is*

$$DMD(p) = \sum_i \sqrt{d_i}$$

In words, a program's DMD is the sum of the square roots of its memory accesses' reuse distances. As such, we arrive at our first theorem:

Theorem 1 (DMD bounds)**.** *Let the time complexity of program p be $O(f(n))$ and let its space complexity be $O(g(n))$. Then*

$$f(n) \leq DMD(p) \leq f(n) \cdot \sqrt{g(n)}$$

where DMD(p) is asymptotic (big-O) DMD.

Proof. It suffices to note that this program will have $f(n)$ memory accesses, the minimum value a reuse distance can take is 1, and the maximum value a reuse distance can take is $g(n)$ (data size). Then, Definition 1 yields the above. □

Memory complexity is measured by DMD in the same way time complexity is by operation count. As Theorem 1 shows, DMD is at least time complexity, because every operation accesses some data. In the following analysis of matrix multiplication, DMD is asymptotically greater than time complexity, and we specifies constant co-efficients. Indeed, DMD is useful only when it is greater than time complexity; otherwise constant-size memory is sufficient, and the memory problem is trivial, i.e. *compute bound*.

An intuitive explanation for the $\sqrt{n}$ cost function is that it is the distance the data must travel if we represent the infinite-level hierarchy as a series of concentric 2-D shapes (such as circles) and let area represent capacity. In the following section, we will discuss the microarchitectural trend that this cost function reflects. In Ding and Smith's formulation for DMD, cache replacement policy is a parameter, however in the derivations in this paper we will use LRU replacement.

Throughout this paper we will use the asymptotic equivalence notation $\sim$ (), which is identical to big-O notation except it retains primary factor coefficients. DMD measures the memory cost and is *complexity without Big-O.*

3.3 Why $\sqrt{n}$?

At the core of the DMD framework is the notion that we need a relationship between stack position, or reuse distance, and cost. Ding and Smith [7] use $\sqrt{n}$ with the argument that it reflects physical memory layout. We expand on that here by demonstrating that $\sqrt{n}$ also reflects the cost of memory access on modern architectures. Figure 1 demonstrates access latency as a function of distance from the processor for cache sizes and latencies corresponding to the AMD Zen2 and Intel Sunny Cove architectures (numbers from [5]) up to $\approx$ 2MB data size. As stack position increases, data is forced into slower caches and its access latency increases. While "distance from the processor" is not a perfect match with LRU stack distance on optimized hardware and Figure 1 contains no empirical measurements, it is clear that the general trend in these architectures' latencies is well captured by the $\sqrt{x}$ family of functions.

This relationship has been observed in other contexts as well: Yavitz et al. [26] show that access latency scales with the square root of cache size, and Cassidy and Andreou [4] demonstrate that energy cost behaves similarly. These

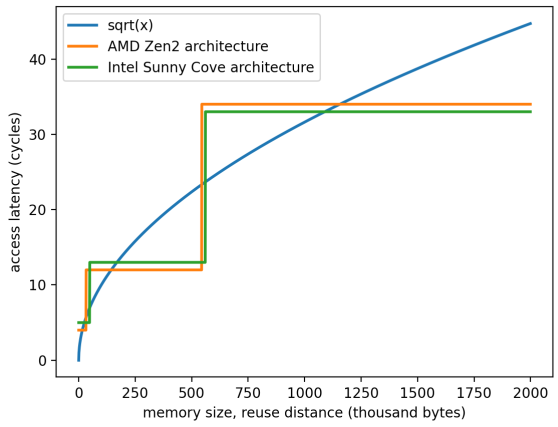

Figure 1. Access latencies of modern microarchitectures as a function of stack position

results, and thus the square root concept, have been used in the design of cutting edge memory systems and techniques, such as Tsai et al.'s Jenga [23].

3.4 Why Measure Hierarchical Locality?

DMD is to measure hierarchical locality. A program has *hierarchical locality* if it makes use of a *cache hierarchy*, where there is has more than a single level of cache, and the cache size and organization may vary from machine to machine. Such cache hierarchies are the norm on today's machines.

As a contrast, consider single-cache locality which means programming to utilize a cache of a specific size. Single-cache locality is unreliable for two reasons. The first is portability. The actual cache usage depends on the choice of programming languages, compilers, and target machines. The second problem is environmental, e.g. interference from run-time systems and from peer programs that share the same cache. Single-cache locality is not a robust program property because of these two sources of uncertainty.

Hierarchical locality is portable and elastic. It is independent of implementation, and it runs well in a shared environment. Cache oblivious algorithms, e.g. recursive matrix multiplication, were developed for hierarchical locality. Next, we use DMD to analyze this effect.

4 RECURSIVE MATRIX MULTIPLICATION MISS RATIO ANALYSIS

In this section we derive what is to our knowledge the first fully precise analytical form of recursive matrix multiplication's (RMM) miss ratio curve on LRU memory. Previously, RMM's asymptotic cache behavior has been derived [18], but we derive the precise, numeric miss ratio for all cache sizes and matrix sizes and validate our model's accuracy against instrumented executions.

Recursive matrix multiplication is a straightforward algorithm whose memory behavior complexity is hidden behind simple pseudocode. It looks as follows:

```
Function rmm(A,B):
    n = A.rows
    let C be a new nxn matrix
    if n == 1:
        C11 = A11 * B11
    else:
        C11 = rmm(A11, B11) + rmm(A12, B21)
        C12 = rmm(A11, B12) + rmm(A12, B22)
        C21 = rmm(A21, B11) + rmm(A22, B12)
        C22 = rmm(A21, B12) + rmm(A22, B22)
    return C
```

The base case computes a 1x1 (i.e. scalar) multiplication, while the recursive case takes a divide-and-conquer approach by splitting each matrix into 4 quadrants and performing 8 recursive calls and 4 additions of the results. This structure is simple, but the way the computation is ordered creates a large amount of complexity and nuance in the resulting memory access pattern.

We can visualize the recursive structure of this algorithm as a graph as follows: Figure 2 contains the decomposition

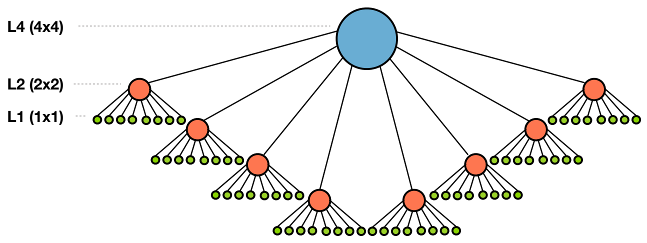

Figure 2. The tree structure of recursive matrix multiplication

of a 4x4 multiplication. The blue node, marked L4, represents a 4x4 multiplication, the orange nodes, marked L2, represent 2x2 multiplications, and the green nodes, marked L1, represent 1x1 multiplications. Execution is a depth-first, left-to-right traversal of this tree.

4.1 Temporaries

As a first step in understanding the memory behavior of RMM's temporaries, we must first understand the way the algorithm is sequenced: the 8 recursive calls are partitioned into 4 groups of 2 calls, and after each group of 2 calls, a matrix addition on two $\frac{N}{2}x\frac{N}{2}$ matrices is performed. To visualize, Figure **??** contains the tree breakdown from before, but with these addition groups explicitly annotated: We will now derive the total number of temporaries T_N introduced in anLN call (multiplication of NxN matrices). First note the number of L1 temporaries, which is of course the number of

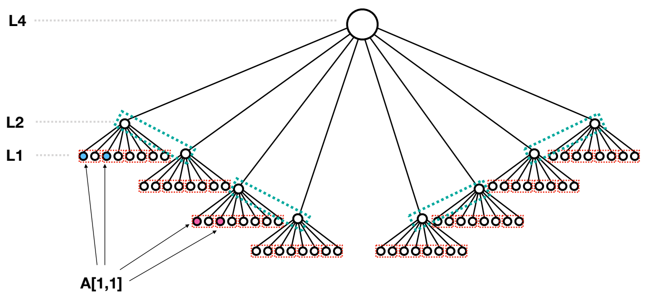

Figure 3. Tree decomposition with A[1,1] accesses highlighted

leaves in the tree:

$$T_{N,L1} = 8^{log_2(N)} \tag{1}$$

Similarly, each parent of a leaf in the tree introduces a 2x2 matrix of temporaries, giving us the following:

$$T_{N,L2} = 8^{log_2(N)-1} \cdot 4 \tag{2}$$

Generalizing:

$$T_N = \sum_{i=0}^{log_2(N)} 8^{log_2(N)-i} \cdot (2^i)^2 \tag{3}$$

Expanding and simplifying:

Definition 2 (Temporary Count)**.** *Let T_N represent the number of temporaries introduced in an LN call to recursive matrix multiplication, i.e. multiplying NxN matrices. Then*

$$T_N = N^2 \cdot (2N-1). \tag{4}$$

Because program data size consists of temporaries and matrices A and B, we can then define D_N, or total data size in an LN call, in terms of T_N:

$$D_N = T_N + 2N^2 \tag{5}$$

For future reference, here are D_N and T_N evaluated for the first few values of N (powers of 2):

N	1	2	4	8
T_N	1	12	112	960
D_N	3	20	144	1088

Definition 3. *In an NxN recursive matrix multiplication, the number of Ln nodes is as follows:*

$$\#Ln = 8^{log_2(N)-log_2(n)} = \frac{N^3}{n^3} \tag{6}$$

Now we turn our attention to the reuse distances of the temporaries themselves. We note that, unlike elements of matrices A and B, there are not reuses between arbitrary levels of the tree: each temporary is defined once and used once as part of an addition. Each temporary incurs only one reuse distance, and it is the distance between its definition and the end of the current addition group (see Figure **??**). As a result, we can exploit the tree's symmetry as follows:

Lemma 1 (Temporary Symmetry). *Let $F_T(i,j,N,a)$ denote the reuse distance of the (i,j)-th element of the a-th temporary matrix introduced at LN. Then*

$$F_T(i,j,N,a) = F_T(i,j,N,(a \% 2)) \tag{7}$$

The order of a follows the depth-first, left-to-right traversal order of the execution tree. Lemma 1 has a straightforward high-level interpretation: at LN, there are only at most two NxN matrices' worth of temporaries with different reuse distances. These two matrices are those resulting from the first and second nodes in an addition group.

We will now diverge slightly from our purely analytical approach. With the aid of an instrumented version of the pseudocode in Figure **??**, we have collected reuse distance data for the introduced temporaries. To minimize the effort of this derivation, we will manually examine the behavior of temporaries for small values of N, extract patterns, and then return to our fully analytical approach to derive the remaining information needed.

We know there are only two matrices to examine per level, the one that appears in the first node of an addition group and the one that appears in the second node. We denote these matrices $DT_{1,N}$ and $DT_{2,N}$. The values in these matrices represent the reuse distance for the single reuse of the temporary that appears in that position of one of the two temporary matrices created at LN.

$DT_{1,N}$:

$$DT_{1,1} = \begin{bmatrix} 4 \end{bmatrix} \tag{8}$$

$$DT_{1,2} = \begin{bmatrix} 38 & 35 \\ 32 & 27 \end{bmatrix} \tag{9}$$

$$DT_{1,4} = \begin{bmatrix} 270 & 269 & 238 & 237 \\ 270 & 269 & 240 & 239 \\ 214 & 213 & 174 & 173 \\ 214 & 213 & 176 & 175 \end{bmatrix} \tag{10}$$

$$DT_{1,8} = \begin{bmatrix} 1982 & 1981 & 1980 & 1979 & 1718 & 1717 & 1716 & 1715 \\ 1982 & 1981 & 1980 & 1979 & 1722 & 1721 & 1720 & 1719 \\ 1982 & 1981 & 1980 & 1979 & 1726 & 1725 & 1724 & 1723 \\ 1982 & 1981 & 1980 & 1979 & 1730 & 1729 & 1728 & 1727 \\ 1502 & 1501 & 1500 & 1499 & 1206 & 1205 & 1204 & 1203 \\ 1502 & 1501 & 1500 & 1499 & 1210 & 1209 & 1208 & 1207 \\ 1502 & 1501 & 1500 & 1499 & 1214 & 1213 & 1212 & 1211 \\ 1502 & 1501 & 1500 & 1499 & 1218 & 1217 & 1216 & 1215 \end{bmatrix} \tag{11}$$

These matrices lead to the following observations:

- The reuse distance patterns evident in $DT_{1,N}$ strongly align with partitioning $DT_{1,N}$ into four quadrants,
- Quadrants (1,1) and (2,1) and quadrants (2,1) and (2,2) differ by a constant,
- Quadrants (1,1) and (2,1) have identical rows,
- Quadrants (2,1) and (2,2) span contiguous ranges of the integers.

The differences between quadrants (1,1), (2,1) and quadrants (2,1), (2,2) can be shown to be $8^{log_2(N)} - 2^{2 \cdot log_2(N)-1}$ and $8^{log_2(N)}$ respectively. This gives rise to the following formulation for $DT_{1,N}$:

Lemma 2 (First Temporary Matrix). *Let $DT_{1,N}$ denote the matrix of reuse distances of temporaries introduced at level N in the first node of an addition group. Then*

See Appendix A

where $\delta(N) = 8^{log_2(N)}$, $\phi(N) = 8^{log_2(N)} - 2^{2 \cdot log_2(N)-1}$, and d_1, d_2 are as described in the following paragraph.

For this matrix, all that remains is to derive d_1 and d_2 from the algorithm itself. We will omit the details, as the analysis is lengthy but straightforward: we count what portions of what matrices are accessed between the temporaries with reuse distances $d1$ and $d2$ to compute them as functions of N. The results, valid for powers of 2 valued N (for $N > 2$, as $d1$ is 4 for $N = 1$):

$$d_1 = \lfloor 2D_N - (2 \cdot (T_{\frac{N}{2}} - 2((\frac{N}{2})^2 - 1)) \rfloor \tag{12}$$

$$d_2 = \lfloor 2D_N - (4T_{\frac{N}{2}} + 2(\frac{N}{2})^2 - 2(\frac{N}{2} - 1) - (2(\frac{N}{2})^2 - \frac{N}{2})) \rfloor \tag{13}$$

We will now apply the same analysis to the other temporary matrix.

$DT_{2,N}$:

$$DT_{2,1} = \begin{bmatrix} 2 \end{bmatrix} \tag{14}$$

$$DT_{2,2} = \begin{bmatrix} 19 & 17 \\ 15 & 11 \end{bmatrix} \tag{15}$$

$$DT_{2,4} = \begin{bmatrix} 127 & 127 & 97 & 97 \\ 131 & 131 & 103 & 103 \\ 79 & 79 & 41 & 41 \\ 83 & 83 & 47 & 47 \end{bmatrix} \tag{16}$$

$$DT_{2,8} = \begin{bmatrix} 895 & 895 & 895 & 895 & 635 & 635 & 635 & 635 \\ 903 & 903 & 903 & 903 & 647 & 647 & 647 & 647 \\ 911 & 911 & 911 & 911 & 659 & 659 & 659 & 659 \\ 919 & 919 & 919 & 919 & 671 & 671 & 671 & 671 \\ 447 & 447 & 447 & 447 & 155 & 155 & 155 & 155 \\ 455 & 455 & 455 & 455 & 167 & 167 & 167 & 167 \\ 463 & 463 & 463 & 463 & 179 & 179 & 179 & 179 \\ 471 & 471 & 471 & 471 & 191 & 191 & 191 & 191 \end{bmatrix} \tag{17}$$

These matrices lead to the following observations:

- The reuse distance patterns evident in $DT_{2,N}$ strongly align with partitioning $DT_{2,N}$ into four quadrants,
- Quadrants (1,1) and (2,1) and quadrants (2,1) and (2,2) differ by a constant,
- All four quadrants have identical columns.
- The values in adjacent rows are separated by a constant across the whole matrix

The differences between quadrants (1,1), (2,1) and quadrants (2,1), (2,2) can be shown to be $8^{log_2(N)} - 2^{2 \cdot log_2(N)}$ and $8^{log_2(N)} - 2^{2 \cdot log_2(N)-1}$ respectively. In addition, the constant separating adjacent rows can be shown to be n for quadrants (1,1), (2,1) and $6 \cdot 2^{log_2(n)-2}$ for quadrants (1,2),(2,2). This gives rise to the following formulation for $DT_{2,N}$:

Lemma 3 (Second Temporary Matrix). *Let $DT_{2,N}$ denote the matrix of reuse distances of temporaries introduced at level N in the second node of an addition group. Then*

See Appendix A

where $\gamma(N) = 8^{log_2(N)} - 2^{2 \cdot log_2(N)}$, $\omega(N) = 8^{log_2(N)} - 2^{2 \cdot log_2(N)-1}$, and d_3, d_4 are as described in the following paragraph.

As before, all that remains is to derive d_3 and d_4 from the algorithm itself. We again omit the details. The results, valid for powers of 2 valued N (again for $N > 2$, as $d3 = 2$ for $N = 1$):

$$d_3 = \lfloor D_N - (2 \cdot T_{\frac{N}{2}} - (2(\frac{N}{2})^2 - 1)) \rfloor \tag{18}$$

$$d_4 = \lfloor D_N - (2(\frac{N}{2})^2) - (4T_{\frac{N}{2}} - 2(\frac{N}{2})^2 + 2) - ((\frac{N}{2})^2 - N) + (\frac{N}{2} + 1) \rfloor \tag{19}$$

4.2 Matrices A and B

To analyze the reuse behavior of the data items in matrices A and B, we must first modify our concept of an LN reuse from our temporary analysis, as accesses to such items only occur in leaves of the tree and not at higher levels..

Definition 4 (Input data reuse levels). *When referring to a reuse of data in matrix A or B, LN indicates the **largest** N for which there exists a complete LN call between the data item's use and reuse.*

To help visualize, revisit Figure 3. In Figure 3, there are two L1 reuses, between the two blue nodes and between the two pink nodes, and 1 L2 reuse, between the second blue node and the first pink node (as there exists a full L2 call between those two).

The tree structure combined with this definition yields the following lemma:

Lemma 4. *Let $RC(N, M)$ denote the number of LM reuses of any item in matrix A or B in an LN call (an NxN multiplication). Then*

$$RC(N, M) = \frac{N}{2M} \tag{20}$$

With an understanding of the frequencies of the LN reuses for matrices A and B, we can now turn our attention to computing the reuse distance values for these LN reuses.

Definition 5 (Reuse distance decomposition). *Let $F_A(i, j, N)$ represent the reuse distance of element $A[i, j]$ at an LN reuse. Then*

$$F_A(i, j, N) = f_T(i, j, N) + f_{A,B}(i, ((j-1) \ \% \ N) + 1, N), \tag{21}$$

where $f_T(i, j, N)$ is the number of unique temporaries within the $A[i, j]$ LN reuse pair, and $f_{A,B}(i, j, N)$ is the number of unique elements of matrices A, B within the same reuse pair. Additionally,

$$F_B(i, j, N) = g_T(i, ((j-1) \ \% \ N) + 1, N) + g_{A,B}(i, j, N), \tag{22}$$

for matrix B.

Definition 5 provides an angle of attack for how to derive $F_A(i, j, N)$ and $F_B(i, j, N)$. In the following sections we will derive, for both matrices A and B, the summed functions in Definition 5. As matrices A and B have substantially different reuse patterns, we must derive these functions separately for each.

4.3 Matrix A

We will take the same angle of approach here as we did for our temporary analysis: we will examine experimentally obtained data to derive most of the elements of $A_{T,N}$ and $A_{A/B,N}$ in terms of their relationships to other elements, and then manually derive those elements in terms of N. Here, $A_{T,N}$ is the matrix whose $[i, j]$th element is $f_T(i, j, N)$, and $A_{A/B,N}$ is the matrix whose $[i, j]$th element is $f_{A,B}(i, j, N)$.

Because elements of $A_{T,N}$ depend on whether the access is in the first or second node in an addition group, $A_{T,N}$ is an $Nx2N$ matrix, while $A_{A/B,N}$ is NxN.

$A_{T,N}$:

$$A_{T,1} = \begin{bmatrix} 3 & 4 \end{bmatrix} \tag{23}$$

$$A_{T,2} = \begin{bmatrix} 26 & 27 & 30 & 31 \\ 28 & 29 & 32 & 33 \end{bmatrix} \tag{24}$$

$$A_{T,4} = \begin{bmatrix} 214 & 215 & 218 & 219 & 230 & 231 & 234 & 235 \\ 216 & 217 & 220 & 221 & 232 & 233 & 236 & 237 \\ 222 & 223 & 226 & 227 & 238 & 239 & 242 & 243 \\ 224 & 225 & 228 & 229 & 240 & 241 & 244 & 245 \end{bmatrix} \tag{25}$$

$$A_{T,8} = \textbf{\textit{See Appendix A}} \tag{26}$$

Note that each of these matrices span the integer range $[a_{T,N}, a_{T,N} + 2N^2 - 1]$, where $a_{T,N} = A_{T,N}[1, 1]$. We need to then derive $a_{T,N}$ as well as the mapping from a tuple (i, j) to an integer in $[a_{T,N}, a_{T,N} + 2N^2 - 1]$.

As before, we elide the full derivation of $a_{T,N}$ from the tree/computation structure, but the result is as follows:

$$a_{T,N} = T_N + 2N^2 + 8T_{\frac{N}{2}} - \sum_{k=0}^{log_2(N)-1} 2 \cdot T_{2^k} \tag{27}$$

We present the mapping $(i,j) \longrightarrow k \in [a_{T,N}, a_{T,N} + 2N^2 - 1]$ algorithmically in Algorithm 1. Transforming this algorithm

Algorithm 1 compute_$a_{T,N}$_constant(i,j,N)

```
Require: $i, j \in [1, N]$
 1: $t \leftarrow 0$
 2: $n_{temp} = N$
 3: if $j > N$ then
 4:     $t \leftarrow N^2$
 5: end if
 6: while $n_{temp} > 1$ do
 7:     if $j > n_{temp}$ then
 8:         $t \leftarrow t + N^2$
 9:     end if
10:     if $i > n_{temp}$ then
11:         $t \leftarrow t + 2 \cdot (\frac{N}{2})^2$
12:     end if
13:     $j \leftarrow ((j-1) \% n_{temp}) + 1$
14:     $i \leftarrow ((i-1) \% n_{temp}) + 1$
15:     $n_{temp} \leftarrow \frac{n_{temp}}{2}$
16: end while
17: return $t$
```

into an equation yields the following:

$$\begin{aligned} t = &\sum_{k=0}^{log_2(N)} (2^k)^2 \cdot I(((j-1\ ) \ \% \ 2^{k+1}) + 1 > 2^k) \\ &+2 \cdot \sum_{k=0}^{log_2(N)} (2^k)^2 \cdot I(((i-1\ ) \ \% \ 2^{k+1}) + 1 > 2^k) \end{aligned} \tag{28}$$

We can now combine the two:

Lemma 5 (Matrix A temporaries). *Let $f_T(i,j,N)$ be the number of unique temporaries within an $A[i,j]$ LN reuse pair. Then*

$$\begin{aligned} f_T(i,j,N) = T_N + 2N^2 + 8T_{\frac{N}{2}} - \sum_{i=0}^{log_2(N)-1} 2T_{2^k} \\ + \sum_{k=0}^{log_2(N)+1} (2^k)^2 \cdot I(((j-1 \ \% \ 2N) \ \% \ 2^{k+1}) + 1 > 2^k) \\ + \sum_{k=0}^{log_2(N)} (2^k)^2 \cdot I(((i-1 \ \% \ N) \ \% \ 2^{k+1}) + 1 > 2^k) \end{aligned} \tag{29}$$

$A_{A/B,N}$:

$$A_{A/B,1} = \begin{bmatrix} 4 \end{bmatrix} \tag{30}$$

$$A_{A/B,2} = \begin{bmatrix} 16 & 17 \\ 18 & 17 \end{bmatrix} \tag{31}$$

$$A_{A/B,4} = \begin{bmatrix} 64 & 65 & 68 & 69 \\ 68 & 68 & 72 & 72 \\ 72 & 72 & 68 & 68 \\ 70 & 69 & 66 & 65 \end{bmatrix} \tag{32}$$

$$A_{A/B,8} = \begin{bmatrix} 256 & 257 & 260 & 261 & 272 & 273 & 276 & 277 \\ 260 & 260 & 264 & 264 & 276 & 276 & 280 & 280 \\ 272 & 272 & 272 & 272 & 288 & 288 & 288 & 288 \\ 272 & 272 & 272 & 272 & 288 & 288 & 288 & 288 \\ 288 & 288 & 288 & 288 & 272 & 272 & 272 & 272 \\ 288 & 288 & 288 & 288 & 272 & 272 & 272 & 272 \\ 280 & 280 & 276 & 276 & 264 & 264 & 260 & 260 \\ 278 & 277 & 274 & 273 & 262 & 261 & 258 & 257 \end{bmatrix} \tag{33}$$

An examination of the algorithm's tree decomposition yields that all of these reuse distances must be in the range $[4N^2, 4N^2 + 2(\frac{N}{2})^2]$.

With this in mind, let $A'_{A/B,N} = A_{A/B,N} - 4N^2$:

$$A'_{A/B,1} = \begin{bmatrix} 0 \end{bmatrix} \tag{34}$$

$$A'_{A/B,2} = \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} \tag{35}$$

$$A'_{A/B,4} = \begin{bmatrix} 0 & 1 & 4 & 5 \\ 4 & 4 & 8 & 8 \\ 8 & 8 & 4 & 4 \\ 6 & 5 & 2 & 1 \end{bmatrix} \tag{36}$$

$$A'_{A/B,8} = \begin{bmatrix} 0 & 1 & 4 & 5 & 16 & 17 & 20 & 21 \\ 4 & 4 & 8 & 8 & 20 & 20 & 24 & 24 \\ 16 & 16 & 16 & 16 & 32 & 32 & 32 & 32 \\ 16 & 16 & 16 & 16 & 32 & 32 & 32 & 32 \\ 32 & 32 & 32 & 32 & 16 & 16 & 16 & 16 \\ 32 & 32 & 32 & 32 & 16 & 16 & 16 & 16 \\ 24 & 24 & 20 & 20 & 8 & 8 & 4 & 4 \\ 22 & 21 & 18 & 17 & 6 & 5 & 2 & 1 \end{bmatrix} \tag{37}$$

The recurrence relation here is as follows. Let $A'_{A/B,N} = \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix}$. Then

$$A'_{A/B,2N} = \begin{bmatrix} \delta_1 & \delta_1 + N^2 \\ N^2 & 2 \cdot N^2 \\ 2 \cdot N^2 & N^2 \\ \delta_2 + N^2 & \delta_2 \end{bmatrix} \tag{38}$$

where each entry in $A'_{A/B,2N}$ represents an $\frac{N}{2}xN$ matrix. Transforming this into a numeric recursive equation as a function of (i,j):

Lemma 6. *Let $f_{A,B}(i,j,N)$ be the number of unique items in matrices A and B within an $A[i,j]$ LN reuse pair.*

$$f_{A,B}(i,j,N) = 4N^2 + f'_{A,B}(i,j,N) \tag{39}$$

where

$$\begin{aligned} f'_{A,B}(i,j,N) = 4\cdot N^2 &+ (\frac{N}{2})^2 \cdot I(\frac{N}{4} < i \leq \frac{N}{2}, j \leq \frac{N}{2}) \\ &+ 2(\frac{N}{2})^2 \cdot I(\frac{N}{4} < i \leq \frac{N}{2}, \frac{N}{2} < j \leq N) \\ &+ (\frac{N}{2})^2 \cdot I(\frac{N}{2} < i \leq \frac{3N}{4}, \frac{N}{2} < j \leq N) \\ &+ 2(\frac{N}{2})^2 \cdot I(\frac{N}{2} < i \leq \frac{3N}{4}, j \leq \frac{N}{2}) \\ &+ I(i > \frac{3N}{4}, j \leq \frac{N}{2}) \cdot ((\frac{N}{2})^2 + f_{A,B}(i - \frac{N}{2}, j, \frac{N}{2})) \\ &+ I(i > \frac{3N}{4}, j > \frac{N}{2}) \cdot (f_{A,B}(i - \frac{N}{2}, j - \frac{N}{2}, \frac{N}{2})) \\ &+ I(i \leq \frac{N}{4}, j > \frac{N}{2}) \cdot ((\frac{N}{2})^2 + f_{A,B}(i, j - \frac{N}{2}, \frac{N}{2})) \\ &+ I(i \leq \frac{N}{4}, j \leq \frac{N}{2}) \cdot (f_{A,B}(i, j, \frac{N}{2})) \end{aligned} \tag{40}$$

and

$$f'_{A,B}(i,j,2) = \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} \tag{41}$$

The eight calls to the indicator function in Lemma 6 correspond to the eight submatrices in the recursive definition of $A'_{A,B}$. The base case of the recursion is $A'_{A/B,2} = \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix}$.

4.4 Matrix B

The reuse structure of matrix B is similar to that of matrix A, although the way computation is ordered with respect to matrix partitioning and traversal means that B's reuse distances will typically be larger than A's. To help visualize this, Figure 4 contains highlighted accesses to element $B[1,1]$: contrast this with the highlighted accesses to element $A[1,1]$ in Figure 3.

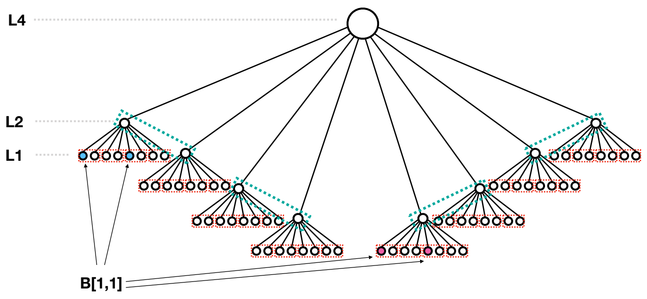

Figure 4. Tree decomposition with $B[1,1]$ accesses highlighted

Our approach for matrix B will be the same as for A: split $F_B(i,j,N)$ into $g_T(i,j,N)$ and $g_{A,B}(i,j,N)$ per Definition 5 and derive each through a mix of experimental analysis and derivation from the algorithm itself. Here, $B_{T,N}$ is the matrix whose $[i,j]$th element is $g_T(i,j,N)$, and $B_{A/B,N}$ is the matrix whose $[i,j]$th element is $g_{A,B}(i,j,N)$.

Because elements of $B_{T,N}$ depend on whether the access is in the first or second node in an addition group, $B_{T,N}$ is an $2NxN$ matrix, while $B_{A/B,N}$ is $2Nx2N$.

$B_{T,N}$:

$$B_{T,1} = \begin{bmatrix} 6 \\ 7 \end{bmatrix} \tag{42}$$

$$B_{T,2} = \begin{bmatrix} 52 & 53 \\ 53 & 54 \\ 56 & 57 \\ 57 & 58 \end{bmatrix} \tag{43}$$

$$B_{T,4} = \begin{bmatrix} 428 & 429 & 432 & 433 \\ 429 & 430 & 433 & 434 \\ 432 & 433 & 436 & 437 \\ 433 & 434 & 437 & 438 \\ 444 & 445 & 448 & 449 \\ 445 & 446 & 449 & 450 \\ 448 & 449 & 452 & 453 \\ 449 & 450 & 453 & 454 \end{bmatrix} \tag{44}$$

$$B_{T,8} = \textbf{\textit{See Appendix A}} \tag{45}$$

We want to derive $b_{T,N} = B_{T,N}[1,1]$ as well as $t_N(i,j) = B_{T,N}[i,j] - b_{T,N}$.

As before, we elide the full derivation of $b_{T,N}$ from the tree/computation structure, but the result is as follows:

$$b_{T,N} = 4 \cdot T_N + 2N^2 - \sum_{k=0}^{log_2(N)-1} 4 \cdot T_{2^k} \tag{46}$$

Note then that, for a set value of i or j (within a column or row), incrementing the other variable (traversing that row or column) results in values for $t(i,j)$ equivalent to interpreting binary-representation integers as having column weights of 4^n not 2^n (e.g. $b, b+1, b+4, b+4+1, b+16, b+16+1\ldots$).

The analytical form for $t(i,j)$ is then as follows:

$$\begin{aligned} t_N(i,j) = &\sum_{k=0}^{\lceil log_2(N) \rceil} 4^k \cdot I(((i-1) \;\%\; 2^{k+1}) + 1 > 2^k) \\ &+ \sum_{k=0}^{\lceil log_2(N) \rceil} 4^k \cdot I(((j-1) \;\%\; 2^{k+1}) + 1 > 2^k) \end{aligned} \tag{47}$$

We can now combine the two:

Lemma 7 (Matrix B temporaries). *Let $g_T(i,j,N)$ be the number of unique temporaries within an $B[i,j]$ LN reuse pair. Then*

$$g_T(i,j,N) = 4 \cdot T_N + 2N^2 - \sum_{k=0}^{log_2(N)-1} 4 \cdot T_{2^k} + \sum_{k=0}^{\lceil log_2(N) \rceil} 4^k \cdot I(((i-1) \ \% \ 2^{k+1}) + 1 > 2^k) + \sum_{k=0}^{\lceil log_2(N) \rceil} 4^k \cdot I(((j-1) \ \% \ 2^{k+1}) + 1 > 2^k) \quad (48)$$

$B_{A/B,N}$:

$$B_{A/B,1} = \begin{bmatrix} 7 & 7 \\ 8 & 6 \end{bmatrix} \quad (49)$$

$$B_{A/B,2} = \begin{bmatrix} 25 & 26 & 30 & 29 \\ 26 & 26 & 30 & 28 \\ 29 & 30 & 26 & 25 \\ 30 & 30 & 26 & 24 \end{bmatrix} \quad (50)$$

$$B_{A/B,4} = \begin{bmatrix} 97 & 98 & 104 & 104 & 120 & 120 & 118 & 117 \\ 98 & 98 & 104 & 104 & 120 & 120 & 118 & 116 \\ 101 & 102 & 104 & 104 & 120 & 120 & 114 & 113 \\ 102 & 102 & 104 & 104 & 120 & 120 & 114 & 112 \\ 113 & 114 & 120 & 120 & 104 & 104 & 102 & 101 \\ 114 & 114 & 120 & 120 & 104 & 104 & 102 & 100 \\ 117 & 118 & 120 & 120 & 104 & 104 & 98 & 97 \\ 118 & 118 & 120 & 120 & 104 & 104 & 98 & 96 \end{bmatrix} \quad (51)$$

An examination of the algorithm's tree decomposition yields that the minimum reuse distance an access to B can incur at LN is $6N^2$.

With this in mind, let $B'_{A/B,N} = B_{A/B,N} - 6N^2$:

$$B'_{A/B,1} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} \quad (52)$$

$$B'_{A/B,2} = \begin{bmatrix} 1 & 2 & 6 & 5 \\ 2 & 2 & 6 & 4 \\ 5 & 6 & 2 & 1 \\ 6 & 6 & 2 & 0 \end{bmatrix} \quad (53)$$

$$B'_{A/B,4} = \begin{bmatrix} 1 & 2 & 8 & 8 & 24 & 24 & 22 & 21 \\ 2 & 2 & 8 & 8 & 24 & 24 & 22 & 20 \\ 5 & 6 & 8 & 8 & 24 & 24 & 18 & 17 \\ 6 & 6 & 8 & 8 & 24 & 24 & 18 & 16 \\ 17 & 18 & 24 & 24 & 8 & 8 & 6 & 5 \\ 18 & 18 & 24 & 24 & 8 & 8 & 6 & 5 \\ 21 & 22 & 24 & 24 & 8 & 8 & 2 & 1 \\ 22 & 22 & 24 & 24 & 8 & 8 & 2 & 0 \end{bmatrix} \quad (54)$$

The recurrence relation here is as follows. Let $B'_{A/B,N} = \begin{bmatrix} \delta_1 & \delta_2 \end{bmatrix}$. Then

$$B'_{A/B,2N} = \begin{bmatrix} \delta_1 & 2N^2 & 6N^2 & \delta_2 + (2N)^2 \\ \delta_1 + (2N)^2 & 6N^2 & 2N^2 & \delta_2 \end{bmatrix} \quad (55)$$

where each entry in $B'_{A/B,2N}$ represents an $2Nx\frac{N}{2}$ matrix. Transforming this into a numeric recursive equation as a function of (i,j):

Lemma 8. *Let $g_{A,B}(i,j,N)$ be the number of unique items in matrices A and B within an $B[i,j]$ LN reuse pair. Then*

$$g_{A,B}(i,j,N) = 6N^2 + g'_{A,B}(i,j,N) \quad (56)$$

where

$$\begin{aligned} g'_{A,B}(i,j,N) = {} & \frac{N^2}{2} \cdot I(i \le N, \frac{N}{2} < j \le N) \\ & + (\frac{3N^2}{2}) \cdot I(i \le N, N < j \le \frac{3N}{2}) \\ & + (\frac{3N^2}{2}) \cdot I(N < i \le 2N, \frac{N}{2} < j \le N) \\ & + (\frac{N^2}{2}) \cdot I(N < i \le 2N, N < j \le \frac{3N}{2}) \\ & + I(i \le N, j \le \frac{N}{2}) \cdot g'_{A,B}(i,j,\frac{N}{2}) \\ & + I(N < i \le 2N, j \le \frac{N}{2}) \cdot (N^2 + g'_{A,B}(i-N,j,\frac{N}{2})) \\ & + I(i \le N, \frac{3N}{2} < j \le 2N) \cdot (N^2 + g'_{A,B}(i,j-N,\frac{N}{2})) \\ & + I(N < i \le 2N, \frac{3N}{2} < j \le 2N) \cdot g'_{A,B}(i-N,j-N,\frac{N}{2}) \end{aligned} \quad (57)$$

and

$$g'_{A,B}(i,j,N) = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} \quad (58)$$

The eight calls to the indicator function in Lemma 8 correspond to the eight submatrices in the recursive definition of $B'_{A,B}$. The base case of the recursion is $B'_{A/B,2} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}$.

4.5 Distribution

We now have enough information to specify the distribution resulting from the previous sections' analyses. Combining Definitions 3,5 and Lemmas 2, 3, 4 we arrive at our first theorem:

Theorem 2 (Reuse Distance Multiset). *The multiset of all reuse distances in an execution of NxN recursive matrix multiplication can be expressed as follows:*

$$\begin{aligned}RD_N = &\bigcup_{l\in\{1,2,4...N\}} \bigcup_{(i,j,k)\in\{1..L\}\times\{1..L\}\times\{1,2\}} \underbrace{\{DT_{k,l}(i,j)...DT_{k,l}(i,j)\}}_{\frac{\#Ll}{2}} \\ &\cup \bigcup_{l\in\{1,2,4...N\}} \bigcup_{(i,j)\in\{1..L\}\times\{1..2L\}} \underbrace{\{F_A(i,j,l)...F_A(i,j,l)\}}_{\frac{N}{2l}\cdot\frac{N^2}{2l^2}} \\ &\cup \bigcup_{l\in\{1,2,4...N\}} \bigcup_{(i,j)\in\{1..2L\}\times\{1..2L\}} \underbrace{\{F_B(i,j,l)...F_B(i,j,l)\}}_{\frac{N}{2l}\cdot\frac{N^2}{4l^2}} \end{aligned} \tag{59}$$

where $DT_{k,l}(i,j)$, $F_A(i,j,N)$, $F_B(i,j,N)$ and $\#L_l$ are defined in earlier lemmas and definitions.

4.6 Algorithmic form

Algorithm 2 contains a specification for how to compute a reuse distribution for RMM given the previous handful of mathematical results and an input size:

Algorithm 2 Reuse Distance Computation

```
Require: $RD : N \rightarrow N${Dictionary. key:RD, value:count.}
compute_RMM_RDD($N$):
for $l \in \{1,2,4...N\}$ do
    for $(i,j,k) \in \{1..L\} \times \{1..L\} \times \{1,2\}$ do
        /* Lemmas 1, 2, 3, Definition 3*/
        $RD[DT_{k,l}(i,j)] \leftarrow RD[DT_{k,l}(i,j)] + \frac{\#Ll}{2}$
    end for
    for $(i,j) \in \{1..L\} \times \{1..2L\}$ do
        /* Lemmas 4, 7, 6, Definition 5 */
        $RD[F(i,j,l)] \leftarrow RD[F(i,j,l)] + \frac{N}{2l}\cdot\frac{N^2}{2l^2}$
    end for
    for $(i,j) \in \{1..2L\} \times \{1..2L\}$ do
        /* Lemmas 4, ??, 8, Definition 5 */
        $RD[G(i,j,l)] \leftarrow RD[G(i,j,l)] + \frac{N}{2l}\cdot\frac{N^2}{4l^2}$
    end for
end for
return $RD${Dictionary stores distribution}
```

Algorithm 2 has runtime $O(n^2 \cdot log(n))$ for matrix dimension n, while collecting this data by running the program and performing trace analysis has runtime $O(n^3 \cdot log(n))$ [16]. Any profiling involving running the program must be $\Omega(n^3)$, demonstrating that our approach has guaranteed asymptotic improvement.

4.7 Verification

We verified Theorem 2 by comparing its resultant RD distribution to an RD distribution formed by instrumenting RMM and performing trace analysis. The two distributions are verified to be identical up to size 256×256.

5 DATA MOVEMENT DISTANCE ANALYSES

In the following section we will derive bounds on the data movement distance (see Definition 1) incurred by several forms of matrix multiplication: naive, tiled, and recursive and Strassen's both with and without temporary reuse. In doing so, we demonstrate the following two valuable properties of DMD:

1. DMD is capable of asymptotically differentiating algorithms with the same time complexity as a result of their memory behavior
2. DMD is capable of asymptotically differentiating between versions of the same algorithm with and without locality optimizations

We construct precise DMD values for naive MM, asymptotically tight upper and lower bounds (differing only in coefficient) for tiled and recursive MM, and upper bounds for Strassen's algorithm and recursive MM with memory management.

5.1 Naive Matrix Multiplication

To precisely derive naive MM's DMD, we first must examine its reuse distance distribution. The left matrix L has the following distribution:

$$P(rd = c) = \begin{cases} 1 & c = 2n \\ 0 & otherwise \end{cases}$$

The right matrix R has a more complex distribution. First, we will look at reuse distance for element $R[i,j]$ as a function of i,j:

$$rd(R[i,j]) = \begin{cases} n^2+n+i & j=1 \\ n^2+2n-i & j=n \\ n^2+2n & otherwise \end{cases}$$

This variance between n^2+n and n^2+2n is because elements in the first and last columns of R see all elements of R, all elements in one row or L, and part of another row of L in between uses. In contrast, elements outside the first and last row see all elements of R and all elements in two rows of L. The resulting reuse distance distribution then looks as follows:

The right matrix R:

$$P(rd = c) = \begin{cases} \frac{n^2-2n+1}{n^2} & c = n^2+2n \\ \frac{1}{n^2} & c = n^2+n \\ \frac{2}{n^2} & c = n^2+n+1 \\ \frac{2}{n^2} & c = n^2+n+2 \\ ... & ... \\ \frac{2}{n^2} & c = n^2+2n-1 \end{cases}$$

The result:

$$DMD_{MM} = (n^3 \cdot \sqrt{2n}) + (n^3 - 2n^2 + n) \cdot \sqrt{n^2 + 2n}$$
$$+(\sum_{i=1}^{n-1} 2n \cdot \sqrt{n^2 + n + i}) + (n \cdot \sqrt{n^2 + n})$$

Simplifying asymptotically:

$$DMD_{MM} \sim n^4$$

5.2 Tiled Matrix Multiplication

Consider matrix multiplication with the computation reordered by partitioning the input matrices into DxD tiles as follows, from [2]:

```
for(jj = 0; jj < N; jj = jj + D)
  for(kk = 0; kk< N; kk = kk + D)
    for(i = 0; i< N; i = i + 1)
      for(j = jj; j < jj + D; j = j + 1)
        for(k = kk; k < kk + D; k = k + 1)
          C[i][j] = C[i][j] + A[i][k]*B[k][j]
```

In this section we use L and R to refer to the left and right matrices being multiplied to avoid overloading B, which represents tile dimension.

Note that this execution corresponds to a sequence of $\frac{N^3}{B^3}$ BxB matrix multiplications, with each BxB tile of R used in $\frac{N}{B}$ of them consecutively. As such, we can modify the reuse distance distribution for regular recursive MM by changing the matrix size to B and including the effect of first/last access reuse. Doing so, we can construct the following bounds on the reuse distances of each access to R:

- Lower bound: B^2
- Upper bound: $2B^2 + B$

In matrix L, all but one reuse per BxB multiplication of each individual datum has reuse distance $3B$, and the other (between BxB multiplications) has reuse distance $N^2 + 2N \cdot B$:

$$P(rd = c) = \begin{cases} \frac{B-1}{B} & c = 3B \\ \frac{1}{B} & c = 2 \cdot N \cdot B + N^2 \end{cases}$$

For tiled matrix multiplication we must consider accesses to matrix C as well, which have the following distribution (similar to matrix L):

$$P(rd = c) = \begin{cases} \frac{B-1}{B} & c = 3 \\ \frac{1}{B} & c = 2 \cdot N \cdot B^2 + 2N \cdot B \end{cases}$$

Combining the previous and simplifying, we can then compute DMD upper and lower bounds.

Upper:

$$DMD_{TMM} > (N^3 - \frac{N^3}{B}) \cdot \sqrt{B^2} + \frac{N^3}{B}\sqrt{N^2} \quad (60)$$
$$+\frac{B-1}{B} \cdot N^3 \cdot \sqrt{3B} + \frac{B-1}{B} \cdot N^3 \cdot \sqrt{3} + \frac{N^3}{B} \cdot \sqrt{2NB} \quad (61)$$

$$\sim (DMD_{TMM}) > \frac{N^4}{B} + N^3 \cdot B \quad (62)$$

Lower:

$$DMD_{TMM} < (N^3 - \frac{N^3}{B}) \cdot \sqrt{2B^2 + B} + \frac{N^3}{B}\sqrt{3N^2} \quad (63)$$
$$+\frac{B-1}{B} \cdot N^3 \cdot \sqrt{3B} + \frac{B-1}{B} \cdot N^3 \cdot \sqrt{3} + \frac{N^3}{B} \cdot \sqrt{3N^2} \quad (64)$$

$$\sim (DMD_{TMM}) < 2\sqrt{3}\frac{N^4}{B} + \sqrt{2}N^3 \cdot B \quad (65)$$

Theorem 3 (Tiled Matrix Multiplication DMD). *Upper and lower bounds on the data movement distance incurred by tiled matrix multiplication operating on NxN matrices with BxB tiles are as follows:*

$$\frac{N^4}{B} + N^3 \cdot B <\sim (DMD_{TMM}) < 2\sqrt{3}\frac{N^4}{B} + \sqrt{2}N^3 \cdot B \quad (66)$$

Note that for $D = 1$ and $D = N$, the bounds are (asymptotically) the same as the DMD of naive matrix multiplication, as tile sizes of 1 and N result in the same computation order as naive MM.

5.3 Recursive Matrix Multiplication

F :

Firstly, note that as $f_T(i, j, N) = \Omega(N^3)$ and $f_{A,B}(i, j, N) = O(N^2)$, $F(i, j, N) \sim f_T(i, j, N)$ (see Definition 5). Similarly, $G(i, j, N) \sim g_T(i, j, N)$. Noting this:

$$f_T(i, j, N) \sim T_N + 8T_{\frac{N}{2}} - \sum_{k=0}^{log_2(N)-1} 2 \cdot T_{2^k}$$

Taking a lower bound on the summation to derive an upper bound on DMD:

$$F(i, j, N) \sim 3N^3$$

Upper bounding the summation for a lower bound:

$$F(i, j, N) \sim 2N^3$$

G has a similar analysis:

$$g_T(i, j, N) \sim 4 \cdot T_N - \sum_{k=0}^{log_2(N)-1} 2 \cdot T_{2^k}$$

An upper bound here, bounding the summation again:

$$G(i, j, N) \sim 7 \cdot N^3$$

The corresponding lower bound:

$$G(i, j, N) \sim 6 \cdot N^3$$

5.4 Accesses to temporaries

We now turn our attention to accesses to temporaries. We will consider the four quadrants of $DT_{1,N}, DT_{2,N}$ separately. First, we need the asymptotic behavior of $d_1, d_2, d_3, d_4, \phi, \delta, \omega, \gamma$:

$$d_1 \sim \frac{7N^3}{2}, d_2 \sim 3N^3, d_3 \sim \frac{3N^3}{2}, d_4 \sim N^3$$
$$\phi(N), \gamma(N), \omega(N), \delta(N) \sim N^3$$

The distribution of asymptotic reuse distances for DT_1 and DT_2 are then as follows, by partitioning each into four quadrants and combining the asymptotic costs of the above:

$$DT_{1,N}: \begin{cases} 1/4 & \frac{7N^3}{2} \\ 1/4 & \frac{5N^3}{2} \\ 1/4 & 3N^3 \\ 1/4 & 2N^3 \end{cases} \qquad DT_{2,N}: \begin{cases} 1/4 & \frac{3N^3}{2} \\ 1/4 & \frac{N^3}{2} \\ 1/4 & N^3 \\ 1/4 & 3N^2 \end{cases}$$

5.5 DMD calculation

With asymptotic reuse distance for each type of memory access, we can now use the frequency information in Theorem 2 to calculate DMD.

5.5.1 Temporaries. DT_1 :

$$DMD_{DT1} = \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{2(2^i)^3}}{4 \cdot 2^i} + \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{3(2^i)^3}}{4 \cdot 2^i} + \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{\frac{7(2^i)^3}{2}}}{4 \cdot 2^i} + \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{\frac{5(2^i)^3}{2}}}{4 \cdot 2^i}$$

$$DMD_{DT1} \sim \frac{2 + \sqrt{5} + \sqrt{6} + \sqrt{7} + 2\sqrt{2} + 2\sqrt{3} + \sqrt{14} + \sqrt{10}}{4} N^{3.5}$$

DT_2 :

$$DMD_{DT2} = \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{(2^i)^3}}{4 \cdot 2^i} + \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{\frac{(2^i)^3}{2}}}{4 \cdot 2^i} + \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{\frac{3(2^i)^3}{2}}}{4 \cdot 2^i} + \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{3(2^i)^2}}{4 \cdot 2^i}$$

$$DMD_{DT2} \sim \frac{3 + 2\sqrt{2} + \sqrt{3} + \sqrt{6}}{4} N^{3.5}$$

5.5.2 Accesses to A and B.

First, we will consider the upper bound on F:

$$DMD_F^{up} = \sum_{i=0}^{log_2(N)} \sum_{j=1}^{2^i} \sum_{k=1}^{2^{i+1}} \frac{N \cdot \sqrt{3(2^i)^3}}{2 \cdot 2^i} = \sum_{i=0}^{log_2(N)} N \cdot 2^i \sqrt{3(2^i)^3}$$

$$DMD_F^{up} \sim \frac{4\sqrt{6}}{4\sqrt{2} - 1} N^{3.5}$$

Next, the lower bound:

$$DMD_F^{low} = \sum_{i=0}^{log_2(N)} \sum_{j=1}^{2^i} \sum_{k=1}^{2^{i+1}} \frac{N \cdot \sqrt{2(2^i)^3}}{2 \cdot 2^i} = \sum_{i=0}^{log_2(N)} N \cdot 2^i \sqrt{2(2^i)^3}$$

$$DMD_F^{low} \sim \frac{8}{4\sqrt{2} - 1} N^{3.5}$$

Similarly for B:

$$DMD_G^{up} = \sum_{i=0}^{log_2(N)} \sum_{j=1}^{2^{i+1}} \sum_{k=1}^{2^i} \frac{N \cdot \sqrt{7(2^i)^3}}{2 \cdot 2^i} = \sum_{i=0}^{log_2(N)} N \cdot 2^i \sqrt{7(2^i)^3}$$

$$DMD_G^{up} \sim \frac{4\sqrt{14}}{4\sqrt{2} - 1} N^{3.5}$$

Next, the lower bound:

$$DMD_G^{low} = \sum_{i=0}^{log_2(N)} \sum_{j=1}^{2^{i+1}} \sum_{k=1}^{2^i} \frac{N \cdot \sqrt{6(2^i)^3}}{2 \cdot 2^i} = \sum_{i=0}^{log_2(N)} N \cdot 2^i \sqrt{6(2^i)^3}$$

$$DMD_G^{low} \sim \frac{8\sqrt{3}}{4\sqrt{2} - 1} N^{3.5}$$

5.5.3 Total DMD.

Theorem 4 (Recursive Matrix Multiplication Data Movement Distance)**.** *Combining the previous, upper and lower bounds on the data movement distance incurred by a standard recursive matrix multiplication algorithm on NxN matrices is as follows:*

$$12.82N^{3.5} <\sim (D_{RMM}(N)) < 13.46N^{3.5}$$

5.5.4 Memory Management.

The RMM pseudocode analyzed earlier allocates memory on each call but contains no calls to **free()**, resulting in poor locality. Practical implementations of RMM will use memory management strategies for handling temporaries, so we will now adapt the previous DMD analysis to take this into account.

One way to introduce temporary freeing would be to call **free()** twice after each addition group, once the addition result has been stored in a C submatrix. Doing so creates the following upper bound on the total number of temporaries needed:

$$\#T(N) = N^2 + \sum_{i=0}^{log_2(N)-1} 2 * (2^i)^2 = \frac{2}{3}(N^2 - 1)$$
$$\#T(N) < 2N^2$$

With this, a bound on the total data size of the execution (including input data) is $N^2 + N^2 + 2N^2 = 4N^2$. We previously derived reuse distances as functions of tree position: we can now reuse our analysis of the DMD of RMM without temporary reuse, but instead of using said functions, we will use the minimum of those functions and $4N^2$ (as reuse distance is always upper bounded by data size). For temporary matrix DT_1:

$$DMD_{DT1}^{up} = \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{min(2(2^i)^3, 4N^2)}}{4 \cdot 2^i} + \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{min(3(2^i)^3, 4N^2)}}{4 \cdot 2^i}$$
$$+ \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{min(\frac{7(2^i)^3}{2}, 4N^2)}}{4 \cdot 2^i} + \sum_{i=0}^{log_2(N)} \frac{N^3 \cdot \sqrt{min(\frac{5(2^i)^3}{2}, 4N^2)}}{4 \cdot 2^i}$$

Solving for the points where the parameters to **min()** are equal, we partition each summation into two by splitting values of induction variable i:

$$DMD_{DT1}^{up} = \sum_{i=0}^{\lfloor \frac{log_2(2N^2)}{3} \rfloor} \frac{N^3 \cdot \sqrt{2(2^i)^3}}{4 \cdot 2^i} + \sum_{i=\lfloor \frac{log_2(2N^2)}{3} \rfloor+1}^{log_2(N)} \frac{N^4}{2(2^i)}$$
$$+ \sum_{i=0}^{\lfloor \frac{log_2(\frac{4}{3}N^2)}{3} \rfloor} \frac{N^3 \cdot \sqrt{3(2^i)^3}}{4 \cdot 2^i} + \sum_{\lfloor \frac{log_2(\frac{4}{3}N^2)}{3} \rfloor+1}^{log_2(N)} \frac{N^4}{2(2^i)}$$
$$+ \sum_{i=0}^{\lfloor \frac{log_2(\frac{8}{7}N^2)}{3} \rfloor} \frac{N^3 \cdot \sqrt{\frac{7}{2}(2^i)^3}}{4 \cdot 2^i} + \sum_{\lfloor \frac{log_2(\frac{8}{7}N^2)}{3} \rfloor+1}^{log_2(N)} \frac{N^4}{2(2^i)}$$
$$+ \sum_{i=0}^{\lfloor \frac{log_2(\frac{8}{5}N^2)}{3} \rfloor} \frac{N^3 \cdot \sqrt{\frac{5}{2}(2^i)^3}}{4 \cdot 2^i} + \sum_{\lfloor \frac{log_2(\frac{8}{5}N^2)}{3} \rfloor+1}^{log_2(N)} \frac{N^4}{2(2^i)}$$

Evaluating:

$$DGC_{DT1} \sim (\frac{1}{2 \cdot 2^{\frac{1}{3}}} + \frac{1}{2 \cdot \frac{4}{3}^{\frac{1}{3}}} + \frac{1}{2 \cdot \frac{8}{7}^{\frac{1}{3}}} + \frac{1}{2 \cdot \frac{8}{5}^{\frac{1}{3}}}$$
$$+ \frac{(2+\sqrt{2})}{4}(2^{\frac{1}{6}} \cdot \sqrt{2} + \frac{4}{3}^{\frac{1}{6}} \cdot \sqrt{3} + \frac{8}{7}^{\frac{1}{6}} \cdot \sqrt{\frac{7}{2}} + \frac{8}{5}^{\frac{1}{6}} \cdot \sqrt{\frac{5}{2}}))N^{\frac{10}{3}}$$

The same analysis for DT_2 results in the following:

$$DMD_{DT2}^{up} \sim (\frac{1}{2 \cdot 4^{\frac{1}{3}}} + \frac{1}{2 \cdot 8^{\frac{1}{3}}} + \frac{1}{2 \cdot \frac{8}{3}^{\frac{1}{3}}}$$
$$+ \frac{(2+\sqrt{2})}{4}(2^{\frac{1}{3}} + 8^{\frac{1}{6}} \cdot \sqrt{\frac{1}{2}} + \frac{8}{3}^{\frac{1}{6}} \cdot \sqrt{\frac{3}{2}}))N^{\frac{10}{3}}$$

We analyze F and G in the same way, but find that they are asymptotically insignificant in comparison to DT_1 and DT_2. So, we arrive at our DMD bound for RMM with memory management:

Theorem 5 (RMM Data Movement Distance With Temporary Reuse: Upper Bound)**.** *Combining the previous, an upper bound on the data movement distance incurred by a recursive matrix multiplication algorithm on NxN matrices employing temporary reuse is as follows:*

$$\sim (DGC_{RMM}^{up}) < 11.85N^{\frac{10}{3}}$$

5.6 Strassen's Algorithm

Strassen's algorithm for matrix multiplication, which reduces time complexity from $O(N^3)$ to around $O(N^{2.8})$ at the cost of worse locality and additional $O(N^2)$ cost, is our final target for analysis. Figure 5 [25] gives pseudocode for Strassen's, which is similar to standard RMM. The core idea is that each call to Strassen's decomposes into 7 recursive calls instead of 8 but contains additional matrix arithmetic. As before, we will analyze Strassen's both with and without locality optimizations for temporary reuse. Note in the pseudocode

```
$M_1 = (A_{11} + A_{22})(B_{11} + B_{22});$
$M_2 = (A_{21} + A_{22})B_{11};$
$M_3 = A_{11}(B_{12} - B_{22});$
$M_4 = A_{22}(B_{21} - B_{11});$
$M_5 = (A_{11} + A_{12})B_{22};$
$M_6 = (A_{21} - A_{11})(B_{11} + B_{12});$
$M_7 = (A_{12} - A_{22})(B_{21} + B_{22}),$
```

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

Figure 5. Strassen's algorithm pseudocode

that, at each level of recursion, 17 temporary matrices are introduced: $M_1 ... M_7$ store the results of recursive computation, and there are ten matrix additions or subtractions that are then passed into recursive calls. As each of these matrices is $\frac{N}{2} \times \frac{N}{2}$, the total number of temporaries needed for this call is $\frac{17N^2}{4}$. Summing over all nodes in the tree decomposition:

Lemma 9 (Strassen's Algorithm Temporary Usage)**.** *Let TS_N be the total number of temporaries required by an $N \times N$ execution of Strassen's algorithm where all temporaries are unique. Then*

$$TS_N = \sum_{i=1}^{log_2(N)} \frac{17}{4} \cdot (2^i)^2 \cdot 7^{log_2(N)-i} = \frac{17}{3}(N^{log_2(7)} - N^2)$$

We can now asymptotically upper bound the reuse distances of each access from each quadrant of A, B as well as the temporary matrices M in terms of TS_N by counting the calls to an $\frac{N}{2} \times \frac{N}{2}$ matrix multiplication between them. For example, in Figure 5 we see that there exist three calls between the first and second uses of $A_{1,1}$, upper bounding each reuse distance from an access in $A_{1,1}$ by $3 \cdot TS_{\frac{N}{2}}$. Without enumerating all of these distances, we see there are a total of 31 such intervals, with a total distance of $96 \cdot TS_{\frac{2}{2}}$. As before, we can sum over all nodes to compute DMD:

$$DMD_{Strassen} \le \sum_{i=1}^{log_2(N)} \sqrt{78 \cdot TS_{2^{i-1}}} \cdot \frac{(2^i)^2}{4} \cdot 7^{log_2(N)-i}$$

Evaluating and asymptotically simplifying:

$$\sim (DMD_{Strassen}) \le \frac{4\sqrt{34}(N^2\sqrt{N^{log_2(7)}} - N^{log_2(7)})}{4\sqrt{7} - 7}$$

Theorem 6 (Strassen's Algorithm DMD: Upper Bound)**.** *An upper bound on the data movement distance incurred by*

Algorithm	**MM**		**RMM**		**Strassen**	
	Naive	Tiled	Naive	Temporary Reuse	Naive	Temporary Reuse
Time Comp.	$O(N^3)$	$O(N^3)$	$O(N^3)$	$O(N^3)$	$O(N^{2.8})$	$O(N^{2.8})$
DMD	N^4	$\sqrt{2}N^3D + \frac{2\sqrt{3}N^4}{D}$	$13.46N^{3.5}$	$11.85N^{3.33}$	$6.51N^{3.4}$	$15.36N^{3.23}$

Table 1. Summary of DMD complexity compared to time complexity

Strassen's algorithm operating on NxN matrices without temporary reuse is as follows:

$$\sim (DMD^{up}_{Strassen}) < 6.51N^{(2+\frac{log_2(7)}{2})}$$

$$\sim (DMD^{up}_{Strassen}) <\approx 6.51N^{3.4}$$

5.6.1 Memory Management. We will adapt the previous analysis in the same way that we did when exploring the effect of adding temporary reuse to RMM. First, we note that Huss-Lederman et al. [11] discuss a memory management technique for Strassen's that requires only N^2 temporaries. This brings total data size for an execution to $3N^2$. We will again use this data size as an upper bound on the value of an individual reuse distance.

The largest of the intervals in the pseudocode is $7 \cdot TS_{\frac{N}{2}}$, so we replace each of them with this so there is only one intersection point to consider. That shifts the sum from $96 \cdot TS_{\frac{N}{2}}$ to $31 \cdot 7 \cdot TS_{\frac{N}{2}} = 217 \cdot TS_{\frac{N}{2}}$. We must multiply our $3N^2$ upper bound by 31 as well, yielding $93N^2$.

$$DMD^{up}_{Strassen'} = \sum_{i=1}^{log_2(N)} \sqrt{min(217 \cdot TS_{2^{i-1}}, 93N^2)} \cdot \frac{(2^i)^2}{4} \cdot 7^{log_2(N)-i}$$

Again we solve for an approximate equivalence point for the two functions that preserves the upper bound:

$$i \approx log_2(0.797N^{\frac{2}{log_2(7))}})$$

Partitioning the previous summation:

$$DMD^{up}_{Strassen'} = \sum_{i=1}^{\lfloor log_2(0.797N^{\frac{2}{log_2(7))}})\rfloor} \sqrt{217 \cdot TS_{2^{i-1}}} \cdot \frac{(2^i)^2}{4} \cdot 7^{log_2(N)-i} + \sum_{i=\lceil log_2(0.797N^{\frac{2}{log_2(7))}})\rceil}^{log_2(N)} \sqrt{93N^2} \cdot \frac{(2^i)^2}{4} \cdot 7^{log_2(N)-i}$$

Simplifying (while preserving the upper bound), removing asymptotically insignificant terms, and evaluating:

Theorem 7 (Strassen's Algorithm With Temporary Reuse DMD: Upper Bound)**.** *An upper bound on the data movement distance incurred by Strassen's algorithm operating on NxN matrices with temporary reuse is as follows:*

$$\sim (DMD^{up}_{Strassen'}) < 15.36N^{3.23}$$

5.7 Summary

Table 1 contains asymptotic simplifications of the results of the previous DMD analyses, with a mix of precise results and upper bounds. We make the following observations:

- DMD is able to distinguish both between different algorithms with the same time complexity and between locality optimized vs. non-optimized versions of the same algorithm,
- The DMD reduction ($\approx N^{\frac{1}{6}}$) from adding temporary reuse is the same for RMM and Strassen even though they have different time and space complexities without temporary reuse,
- The gap between Strassen and RMM DMD is smaller than the gap between their time complexities, demonstrating that DMD has captured some of the factors that make Strassen not practical.

6 RELATED WORK

Hong and Kung [9] pioneered the study of I/O complexity, measuring memory complexity by the amount of data transfer and deriving this complexity symbolically as a function of the memory size and the problem size. The same complexity measures were used in the study of cache oblivious algorithms [8] and communication-avoiding algorithms. Olivry et al. [15] introduce a compiler technique to statically derive I/O complexity bounds. Olivry et al. use asymptotic complexity (~), much as we do, to consider constant-factor performance differences, while the rest do not. I/O complexity suffers from an issue inherited from miss ratio curves: it is not ordinal across cache sizes. Ordinality means that data can be usefully ordered; integers are ordinal, while functions are typically not. Miss ratio as a function of cache size can be numerically compared by an algorithm designer when a target cache size is known, but it cannot be used to evaluate the general effectiveness of an optimization across cache sizes. DMD, on the other hand, is an ordinal metric that is agnostic to cache size.

Memory hierarchies in practice may vary in many ways, which make a unified cost model difficult. Valiant [24] defined a bridging model, Multi-BSP, for a multi-core memory hierarchy with a set of parameters including the number of levels and the memory size and three other factors at each level. A simpler model was the uniform memory hierarchy (UMH) by Alpern et al. [1] who used a single scaling factor for the capacity and the access cost across all levels. Both are models of memory, where caching is not considered beyond the point that the memory may be so implemented.

Matrix multiplication is well researched. Much effort has been put into the analysis and optimization of the Strassen algorithm and its cache utilization [11, 17, 19, 22] as well as its parallel behavior [3, 21]. Lincoln et al. [13] explore performance in a dynamically sized caching environment. Recently, researchers have argued that, with proper implementation considerations and under the correct conditions, the Strassen algorithm can outperform more conventionally practical variants of MM [10].

Kung and Leiserson [12] showed that matrix multiplication for $N \times N$ matrices on systolic arrays takes N steps with N^2 processors computing in parallel. At each step, data moves only between adjacent processors. Taking the wire length between neighboring processors as unit distance and ignoring the data movement outside the systolic array, the DMD is N^3. Systolic arrays achieve the lowest possible asymptotic data movement, matching time complexity. However, the algorithm requires N^2 processors. We leave the DMD of parallel algorithms as the subject of future study.

7 CONCLUSION

We have explored the interactions between six variants of matrix multiplication and hierarchical memory through the lens of Data Movement Distance. We have demonstrated that DMD's assumptions conform with microarchitectural trends and that it is capable of exposing algorithmic properties that traditional analyses cannot. Time complexity, while an important theoretic metric by which to analyze algorithms, is at odds with a computing environment in which memory systems are increasingly large and complex. We argue that data movement complexity analysis through DMD has great potential to help engineers and algorithm designers to understand the algorithmic implications of hierarchical memory.

The results presented in this paper leave open several interesting avenues for future work. One is applying our analysis to more algorithms. Interesting targets are widely used algorithms with nontrivial memory behavior, such as convolution and Fourier transforms. Another is exploring the performance to data movement complexity relationship with empirical results. Lastly, extending this work to consider the effect of parallelism on interactions between algorithms and memory hierarchies would improve its applicability.

Acknowledgments

Donovan Snyder participated in the early stages of this project. We also thank ICS reviewers, Xiaobai Sun at Duke, Nathan Beckmann at CMU, and our colleagues at University of Rochester especially Benjamin Reber, Fangzhou Liu, John Criswell, and Michael Scott for the discussion and feedback on the DMD concept and its presentation. This work was supported in part by the National Science Foundation (Contract No. CCF-2217395, CCF-2114319, CNS-1909099). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding organizations.

References

[1] B. Alpern, L. Carter, E. Feig, and T. Selker. 1994. The uniform memory hierarchy model of computation. *Algorithmica* 12, 2/3 (1994), 72–109.

[2] Bin Bao and Chen Ding. 2013. Defensive loop tiling for shared cache. In *Proceedings of the International Symposium on Code Generation and Optimization*. 1–11.

[3] Guy E. Blelloch, Rezaul A. Chowdhury, Phillip B. Gibbons, Vijaya Ramachandran, Shimin Chen, and Michael Kozuch. 2008. Provably Good Multicore Cache Performance for Divide-and-Conquer Algorithms *(SODA '08)*. Society for Industrial and Applied Mathematics, USA, 501–510.

[4] Andrew S. Cassidy and Andreas G. Andreou. 2012. Beyond Amdahl's Law: An Objective Function That Links Multiprocessor Performance Gains to Delay and Energy. *IEEE Trans. Comput.* 61, 8 (2012), 1110–1126. https://doi.org/10.1109/TC.2011.169

[5] Ian Cutress. 2019. The Ice Lake Benchmark Preview: Inside Intel's 10nm. https://www.anandtech.com/show/14664/testing-intel-ice-lake-10nm/2

[6] Bill Dally. [n.d.]. From Here to Exascale: Challenges and Potential Solutions.

[7] Chen Ding and Wesley Smith. 2021. Memory Access Complexity: A Position Paper. In *Proceedings of the International Symposium on Memory Systems (MEMSYS)*.

[8] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. 1999. Cache-Oblivious Algorithms. In *Proceedings of the Symposium on Foundations of Computer Science*. 285–298.

[9] Jia-Wei Hong and H. T. Kung. 1981. I/O complexity: The red-blue pebble game. In *Proceedings of the ACM Conference on Theory of Computing*. Milwaukee, WI, 326–333.

[10] Jianyu Huang, Tyler M. Smith, Greg M. Henry, and Robert A. van de Geijn. 2016. Strassen's Algorithm Reloaded. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, Utah) *(SC '16)*. IEEE Press, Article 59, 12 pages.

[11] Steven Huss-Lederman, Elaine Jacobson, Jeremy Johnson, Anna Tsao, and Thomas Turnbull. 1997. Implementation of Strassen's Algorithm for Matrix Multiplication. (10 1997). https://doi.org/10.1145/369028.369096

[12] H. T. Kung and Charles E. Leiserson. 1979. *Systolic Arrays for (VLSI)*. Technical Report CMU-CS-79-103. Cargegie-Mellon University.

[13] Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Helen Xu. 2018. Cache-Adaptive Exploration: Experimental Results and Scan-Hiding for Adaptivity. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures* (Vienna, Austria) *(SPAA '18)*. Association for Computing Machinery, New York, NY, USA, 213–222. https://doi.org/10.1145/3210377.3210382

[14] R. L. Mattson, J. Gecsei, D. Slutz, and I. L. Traiger. 1970. Evaluation techniques for storage hierarchies. *IBM System Journal* 9, 2 (1970), 78–117.

[15] Auguste Olivry, Julien Langou, Louis-Noël Pouchet, P. Sadayappan, and Fabrice Rastello. 2020. Automated Derivation of Parametric Data Movement Lower Bounds for Affine Programs. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 808–822. https://doi.org/10.1145/3385412.3385989

[16] F. Olken. 1981. *Efficient methods for calculating the success function of fixed space replacement policies.* Technical Report LBL-12370. Lawrence Berkeley Laboratory.

[17] V.Paul Pauca, Pauca Xiaobai, Sun Chatterjee, Xiaobai Sun, and Alvin Lebeck. 1998. Architecture-efficient Strassen's Matrix Multiplication: A Case Study of Divide-and-Conquer Algorithms. (07 1998).

[18] Harald Prokop. 1999. Cache-Oblivious Algorithms.

[19] Vikash Kumar Singh, Hemant Makwana, and Richa Gupta. 2015. Comparative Study of Cache Utilization for Matrix Multiplication Algorithms.

[20] Donovan Snyder and Chen Ding. 2021. Measuring Cache Complexity Using Data Movement Distance (DMD). In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* IEEE, 417–419.

[21] Yuan Tang. 2020. *Balanced Partitioning of Several Cache-Oblivious Algorithms.* Association for Computing Machinery, New York, NY, USA, 575–577. https://doi.org/10.1145/3350755.3400214

[22] Mithuna Thottethodi, Siddhartha Chatterjee, and Alvin Lebeck. 1998. Tuning Strassen's Matrix Multiplication for Memory Efficiency. 36–36. https://doi.org/10.1109/SC.1998.10045

[23] Po-An Tsai, Nathan Beckmann, and Daniel Sanchez. 2017. Jenga: Software-Defined Cache Hierarchies *(ISCA '17).* Association for Computing Machinery, New York, NY, USA, 652–665. https://doi.org/10.1145/3079856.3080214

[24] Leslie G. Valiant. 2008. A Bridging Model for Multi-core Computing. In *Algorithms - ESA 2008, 16th Annual European Symposium.* 13–28.

[25] Wikipedia contributors. 2021. Strassen algorithm — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Strassen_algorithm&oldid=1049348598 [Online; accessed 24-January-2022].

[26] Leonid Yavits, Amir Morad, and Ran Ginosar. 2014. Cache Hierarchy Optimization. *IEEE Computer Architecture Letters* 13, 2 (2014), 69–72. https://doi.org/10.1109/L-CA.2013.18

[27] Liang Yuan, Chen Ding, Wesley Smith, Peter J. Denning, and Yunquan Zhang. 2019. A Relational Theory of Locality. *ACM Transactions on Architecture and Code Optimization* 16, 3 (2019), 33:1–33:26.

A Full size matrices

Lemma 2:

Let $DT_{1,N}$ denote the matrix of reuse distances of temporaries introduced at level N in the first node of an addition group. Then

$$DT_{1,N} = \begin{bmatrix} d_1 & d_1 - 1 & ... & d_1 - (\frac{n}{2} - 1) & d_2 - (\frac{n}{2})^2 + \frac{n}{2} & ... & d_2 - (\frac{n}{2})^2 + 2 & d_2 - (\frac{n}{2})^2 + 1 \\ d_1 & d_1 - 1 & ... & d_1 - (\frac{n}{2} - 1) & d_2 - (\frac{n}{2})^2 + n & ... & d_2 - (\frac{n}{2}^2) + \frac{n}{2} + 2 & d_2 - (\frac{n}{2})^2 + \frac{n}{2} + 1 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ d_1 & d_1 - 1 & ... & d_1 - (\frac{n}{2} - 1) & d_2 & ... & d_2 - \frac{n}{2} & d_2 - \frac{n}{2} - 1 \\ d_1 - \phi(n) & d_1 - \phi(n) - 1 & ... & d_1 - \phi(n) - (\frac{n}{2} - 1) & d_2 - \delta(n) - (\frac{n}{2})^2 + \frac{n}{2} & ... & d_2 - \delta(n) - (\frac{n}{2})^2 + 2 & d_2 - \delta(n) - (\frac{n}{2}^2) + 1 \\ d_1 - \phi(n) & d_1 - \phi(n) - 1 & ... & d_1 - \phi(n) - (\frac{n}{2} - 1) & d_2 - \delta(n) - (\frac{n}{2})^2 + n & ... & d_2 - \delta(n) - (\frac{n}{2})^2 + \frac{n}{2} + 2 & d_2 - \delta(n) - (\frac{n}{2})^2 + \frac{n}{2} + 1 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ d_1 - \phi(n) & d_1 - \phi(n) - 1 & ... & d_1 - \phi(n) - (\frac{n}{2} - 1) & d_2 - \delta(n) & ... & d_2 - \delta(n) - \frac{n}{2} & d_2 - \delta(n) - \frac{n}{2} - 1 \end{bmatrix} \tag{67}$$

Lemma 3:

Let $DT_{2,N}$ denote the matrix of reuse distances of temporaries introduced at level N in the second node of an addition group. Then

$$DT_{2,N} = \begin{bmatrix} d_3 & ... & d_3 & d_4 & ... & d_4 \\ d_3 + n & ... & d_3 + n & d_4 + 6 \cdot 2^{log_2(n)-2} & ... & d_4 + 6 \cdot 2^{log_2(n)-2} \\ ... & ... & ... & ... & ... & ... \\ d_3 + n(\frac{n}{2} - 1) & ... & d_3 + n(\frac{n}{2} - 1) & d_4 + (\frac{n}{2} - 1)(6 \cdot 2^{log_2(n)-2}) & ... & d_4 + (\frac{n}{2} - 1)(6 \cdot 2^{log_2(n)-2}) \\ d_3 - \gamma(N) & ... & d_3 - \gamma(N) & d_4 - \omega(N) & ... & d_4 - \omega(N) \\ d_3 - \gamma(N) + n & ... & d_3 - \gamma(N) + n & d_4 - \omega(N) + 6 \cdot 2^{log_2(n)-2} & ... & d_4 - \omega(N) + 6 \cdot 2^{log_2(n)-2} \\ ... & ... & ... & ... & ... & ... \\ d_3 - \gamma(N) + n(\frac{n}{2} - 1) & ... & d_3 - \gamma(N) + n(\frac{n}{2} - 1) & d_4 - \omega(N) + (\frac{n}{2} - 1)(6 \cdot 2^{log_2(n)-2}) & ... & d_4 - \omega(N) + (\frac{n}{2} - 1)(6 \cdot 2^{log_2(n)-2}) \end{bmatrix} \tag{68}$$

$$A_{T,8} = \begin{bmatrix} 1734 & 1735 & 1738 & 1739 & 1750 & 1751 & 1754 & 1755 & 1798 & 1799 & 1802 & 1803 & 1814 & 1815 & 1818 & 1819 \\ 1736 & 1737 & 1740 & 1741 & 1752 & 1753 & 1756 & 1757 & 1800 & 1801 & 1804 & 1805 & 1816 & 1817 & 1820 & 1821 \\ 1742 & 1743 & 1746 & 1747 & 1758 & 1759 & 1762 & 1763 & 1806 & 1807 & 1810 & 1811 & 1822 & 1823 & 1826 & 1827 \\ 1744 & 1745 & 1748 & 1749 & 1760 & 1761 & 1764 & 1765 & 1808 & 1809 & 1812 & 1813 & 1824 & 1825 & 1828 & 1829 \\ 1766 & 1767 & 1770 & 1771 & 1782 & 1783 & 1786 & 1787 & 1830 & 1831 & 1834 & 1835 & 1846 & 1847 & 1850 & 1851 \\ 1768 & 1769 & 1772 & 1773 & 1784 & 1785 & 1788 & 1789 & 1832 & 1833 & 1836 & 1837 & 1848 & 1849 & 1852 & 1853 \\ 1774 & 1775 & 1778 & 1779 & 1790 & 1791 & 1794 & 1795 & 1838 & 1839 & 1842 & 1843 & 1854 & 1855 & 1858 & 1859 \\ 1776 & 1777 & 1780 & 1781 & 1792 & 1793 & 1796 & 1797 & 1840 & 1841 & 1844 & 1845 & 1856 & 1857 & 1860 & 1861 \end{bmatrix} \tag{69}$$

$$B_{T,8} = \begin{bmatrix} 3468 & 3469 & 3472 & 3473 & 3484 & 3485 & 3488 & 3489 \\ 3469 & 3470 & 3473 & 3474 & 3485 & 3486 & 3489 & 3490 \\ 3472 & 3473 & 3476 & 3477 & 3488 & 3489 & 3492 & 3493 \\ 3473 & 3474 & 3477 & 3478 & 3489 & 3490 & 3493 & 3494 \\ 3484 & 3485 & 3488 & 3489 & 3500 & 3501 & 3504 & 3505 \\ 3485 & 3486 & 3489 & 3490 & 3501 & 3502 & 3505 & 3506 \\ 3488 & 3489 & 3492 & 3493 & 3504 & 3505 & 3508 & 3509 \\ 3489 & 3490 & 3493 & 3494 & 3505 & 3506 & 3509 & 3510 \\ 3532 & 3533 & 3536 & 3537 & 3548 & 3549 & 3552 & 3553 \\ 3533 & 3534 & 3537 & 3538 & 3549 & 3550 & 3553 & 3554 \\ 3536 & 3537 & 3540 & 3541 & 3552 & 3553 & 3556 & 3557 \\ 3537 & 3538 & 3541 & 3542 & 3553 & 3554 & 3557 & 3558 \\ 3548 & 3549 & 3552 & 3553 & 3564 & 3565 & 3568 & 3569 \\ 3549 & 3550 & 3553 & 3554 & 3565 & 3566 & 3569 & 3570 \\ 3552 & 3553 & 3556 & 3557 & 3568 & 3569 & 3572 & 3573 \\ 3553 & 3554 & 3557 & 3558 & 3569 & 3570 & 3573 & 3574 \end{bmatrix} \tag{70}$$