Faster Matrix Multiplication via Sparse Decomposition

Gal Beniamini
The Hebrew University of Jerusalem
gal.beniamini@mail.huji.ac.il

Oded Schwartz
The Hebrew University of Jerusalem
odedsc@cs.huji.ac.il

ABSTRACT

Fast matrix multiplication algorithms are of practical use only if the leading coefficient of their arithmetic complexity is sufficiently small. Many algorithms with low asymptotic cost have large leading coefficients, and are thus impractical. Karstadt and Schwartz have recently demonstrated a technique that reduces the leading coefficient by introducing fast $O(n^2 \log n)$ basis transformations, applied to the input and output matrices.

We generalize their technique, by allowing larger bases for the transformations while maintaining low overhead. Thus we accelerate several matrix multiplication algorithms, beyond what is known to be possible using the previous technique. Of particular interest are a few new sub-cubic algorithms with leading coefficient 2, matching that of classical matrix multiplication. For example, we obtain an algorithm with arithmetic complexity of $2n^{\log_3 23} + o(n^{\log_3 23})$ compared to $2n^3 - n^2$ of the classical algorithm. Such new algorithms can outperform previous ones (classical included) even on relatively small matrices. We obtain lower bounds matching the coefficient of several of our algorithms, proving them to be optimal.

CCS CONCEPTS

• **Mathematics of computing → Computations on matrices**;
• **Computing methodologies → Linear algebra algorithms**;

KEYWORDS

Fast Matrix Multiplication, Bilinear Algorithms

ACM Reference Format:
Gal Beniamini and Oded Schwartz. 2019. Faster Matrix Multiplication via Sparse Decomposition. In *31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '19), June 22–24, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3323165.3323188

1 INTRODUCTION

Matrix Multiplication is a fundamental computation kernel, used in many parallel and sequential algorithms. Improving its performance has attracted the attention of many researchers. Strassen's algorithm [32] was the first sub-cubic matrix multiplication algorithm. Since then, research regarding fast multiplication algorithms has bifurcated into two main streams.

The first focuses on deriving asymptotic improvements by reducing the exponent of the arithmetic complexity (cf. [4, 10, 11, 21, 28, 29, 31, 33, 37]). Often, these improvements come at the cost of large "hidden constants", rendering them impractical. Moreover, the aforementioned algorithms are typically only applicable to matrices of very large dimensions, further restricting their practicality.

The second line of research focuses on obtaining asymptotically fast algorithms while maintaining lower hidden costs; allowing multiplication of reasonably-sized matrices (cf. [3, 16–20, 24, 25, 30, 34]). These methods are thus more likely to have practical applications. Within this line of research, several algorithms have been discovered via computer-aided techniques (cf. [1, 3, 30]).

In 1978, Pan [24, 25] reduced the problem of matrix multiplication to the Triple-Product Trace, allowing the derivation of several sub-cubic algorithms with relatively small base cases, such as $\langle 70, 70, 70; 143640\rangle$[1], $\langle 40, 40, 40; 36133\rangle$, and $\langle 18, 18, 18; 3546\rangle$, allowing multiplication in $\Theta(n^{\omega_0})$, where $\omega_0 = \log_{70} 143640 \approx 2.79$, $\omega_0 = \log_{40} 36133 \approx 2.84$, and $\omega_0 = \log_{18} 3546 \approx 2.82$, respectively. In 2013, Smirnov [30] used computer-aided search to find base cases. Notable among Smirnov's algorithms are $\langle 6, 3, 3; 40\rangle$-algorithm, and $\langle 4, 3, 3; 29\rangle$-algorithm, allowing multiplication in $\Theta(n^{\omega_0})$, where $\omega_0 = \log_{54}(40^3) \approx 2.774$ and $\omega_0 = \log_{36}(29^3) \approx 2.818$, respectively. Similarly, Ballard and Benson [3] used computer-aided techniques to derive additional multiplication algorithms, such as $\langle 5, 2, 2; 18\rangle$ and $\langle 3, 2, 2; 11\rangle$, allowing multiplication in $\Theta(n^{\omega_0})$, where $\omega_0 = \log_{20} 18^3 \approx 2.89$ and $\omega_0 = \log_{12} 11^3 \approx 2.89$, respectively.

1.1 Previous Research

The hidden constants of the arithmetic complexity of recursive-bilinear algorithms, including matrix multiplication, is determined by the number of linear operations performed in the base case. Strassen's $\langle 2, 2, 2; 7\rangle$-algorithm has a base-case with 18 additions, resulting in a leading coefficient of 7. This was later reduced to 15 additions by Winograd [38], decreasing the leading coefficient from 7 to 6. Probert [27] and Bshouty [7] showed that 15 additions are necessary for any $\langle 2, 2, 2; 7\rangle$-algorithm, leading to the conclusion that the leading coefficient of Strassen-Winograd is optimal for the 2×2 base case.

Karstadt and Schwartz [18] recently observed that these lower-bounds implicitly assume the input and output are given in the standard basis. Discarding this assumption allows further reduction in the number of arithmetic operations from 15 to 12, decreasing the leading coefficient to 5. The same approach, applied to other algorithms, resulted in a significant reduction of the corresponding leading coefficients (see Table 1). Moreover, Karstadt and Schwartz extended the lower bounds due to Probert [27] and Bshouty [7] by allowing algorithms that include basis transformations, thus

[1] The notation $\langle n, m, k; t\rangle$ refers to an algorithm with a base case that multiplies matrices of dimension $n \times m$, $m \times k$ using t scalar multiplications. See Notation 2.6

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SPAA '19, June 22–24, 2019, Phoenix, AZ, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6184-2/19/06…$15.00
https://doi.org/10.1145/3323165.3323188

Table 1: Examples of Decomposed Algorithms

Algorithm	Leading Monomial	Linear Operations			Leading Coefficient				Improvement	
		Original	[18]	[Here]	Original	[18]	[Here]	Optimal	[18]	[Here]
$\langle 6,3,3;40\rangle$ [30]	$n^{\log_{54} 40^3} \approx n^{2.774}$	1246	202	6	55.63	9.39	7	?	83.1%	87.4%
$\langle 2,2,2;7\rangle$ [32]	$n^{\log_2 7} \approx n^{2.807}$	18	12	4	7	5	5	✓	28.6%	28.6%
$\langle 4,3,3;29\rangle$ [30]	$n^{\log_{36} 29^3} \approx n^{2.818}$	137	109	1	8.54	6.96	2	✓	18.5%	76.6%
$\langle 3,3,3;23\rangle$ [3, 17, 20]	$n^{\log_3 23} \approx n^{2.854}$	97	61	1	7.93	5.36	2	✓	32.4%	74.8%
$\langle 5,2,2;18\rangle$ [3]	$n^{\log_{20} 18^3} \approx n^{2.894}$	53	32	1	6.98	4.46	2	✓	36.1%	71.3%
$\langle 3,2,2;11\rangle$ [3]	$n^{\log_{12} 11^3} \approx n^{2.894}$	22	18	1	5.06	4.6	2	✓	9.1%	60.5%

The leading monomial of rectangular $\langle n, m, k; t\rangle$-algorithms refers to their composition [15] into square $\langle nmk, nmk, nmk; t^3\rangle$-algorithms. The improvement column is the ratio between the leading coefficients of the arithmetic complexity of Alternative Basis [18] or Decomposed algorithms [Here], and the original algorithms. The $\langle 4,3,3;29\rangle$, $\langle 3,3,3;23\rangle$, $\langle 5,2,2;18\rangle$, and $\langle 3,2,2;11\rangle$-algorithms are optimally decomposed with a leading coefficient matching that of the classical algorithm, for up to three levels of decomposition (Definition 3.5). All optimally decomposed $\langle n, m, k; t\rangle$-algorithms with the leading coefficient 2 maintain the same coefficient when converted to square $\langle nmk, nmk, nmk; t^3\rangle$-algorithms.

proving that their $\langle 2,2,2;7\rangle$-algorithm obtains an optimal leading coefficient in the alternative basis regime.

Key to the approach of Karstadt and Schwartz are fast basis transformations, which can be computed in $O(n^2 \log n)$, asymptotically faster than the matrix multiplication itself. These transformations can be viewed as an extension of the "intermediate representation" approach, which previously appeared in Bodrato's [5] method for matrix squaring.

Cenk and Hassan [8] developed a technique for computing multiplication algorithms, such as Strassen's, which utilizes memoization, allowing a reduction of the leading coefficient. Their approach obtains a $\langle 2,2,2;7\rangle$-algorithm with a leading coefficient of 5, as in Karstadt and Schwartz [18], albeit with larger exponents in the low-order monomials.

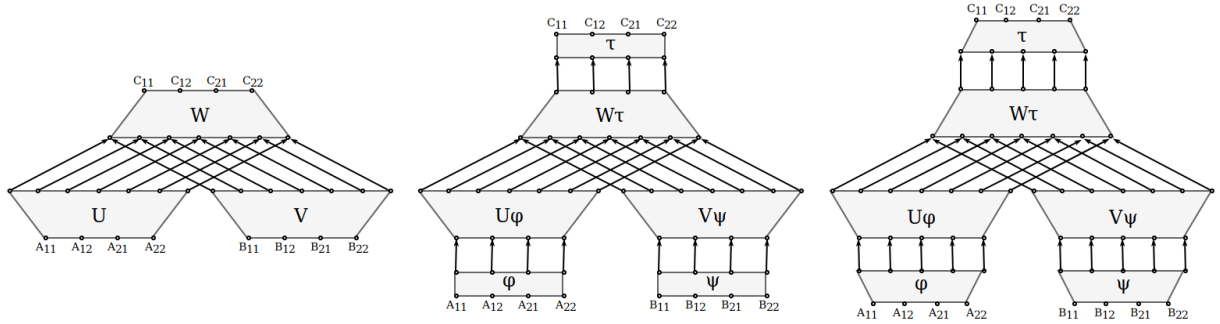
1.2 Our Contribution

We extend Karstadt and Schwartz's method for Alternative Basis Multiplication. While the basis transformations of [18] are homomorphisms over the same linear space (i.e., changes of basis), we consider transformations into a linear space of any intermediate dimension (see Figure 1). Such transformations incur costs of low-order monomials, as opposed to the $O(n^2 \log n)$ cost of basis transformations, but allow further reduction of the leading (and other) coefficients.

We rely on the mixed-product property of the Kronecker Product (Fact 3.2) to rearrange the computation graph, allowing us to aggregate all the decompositions into a single stage of the algorithm. As the aforementioned transformations correspond to low-order monomials, we intentionally "offload" part of the computation onto them. To this end, we utilize decompositions in which the matrices of maps contributing to the leading monomial are sparse, whereas the matrices of transformations contributing to low-order monomials may be relatively dense.

We apply our decomposition scheme to several fast matrix multiplication algorithms, resulting in significant reduction of their arithmetic complexity compared to previous techniques (Table 1). We present several decomposed sub-cubic algorithms with leading coefficient 2, matching that of the classical multiplication algorithm. Such algorithms outperform previous ones (classical included) even

Figure 1: Illustrations of Recursive-Bilinear, Alternative Basis and Decomposed Matrix Multiplication



Recursive Bilinear | Alternative Basis [18] | Decomposed [here]

Table 2: Example of Arithmetic Complexity: $\langle 3,3,3\rangle$-Algorithms

Algorithm	Arithmetic Complexity
$\langle 3,3,3;27\rangle$ Classical	$2n^3 - n^2$
$\langle 3,3,3;23\rangle$ Original Algorithm [3, 17, 20]	$7.93n^{\log_3 23} - 6.93n^2$
$\langle 3,3,3;23\rangle$ Alternative Basis [18]	$5.36n^{\log_3 23} + 3.22n^2\log_3 n - 4.36n^2$
$\langle 3,3,3;23\rangle$ Decomposed [Here]	$2n^{\log_3 23} + 6.75n^{\log_3 21} - 7.75n^2$
$\langle 3,3,3;23\rangle$ Fully Decomposed [Here]	$2n^{\log_3 23} + 3n^{\log_3 20} + 2n^{\log_3 14} + 2n^{\log_3 12} + 2n^{\log_3 11} + 33n^{\log_3 10} - 43n^2$

In terms of arithmetic complexity, the fully decomposed algorithm outperforms all $\langle 3,3,3\rangle$-algorithms for $n \geq 243$ (see Figure 3)

on small matrices. In particular, we obtain decompositions with said properties for $\langle 4,3,3;29\rangle$-algorithm [30], $\langle 3,3,3;23\rangle$-algorithm [3, 17, 20], $\langle 5,2,2;18\rangle$-algorithm [3], and $\langle 3,2,2;11\rangle$-algorithm [3]. Furthermore, optimally decomposed algorithms maintain the leading coefficient of 2 when converted into square $\langle nmk, nmk, nmk; t^3\rangle$-algorithms. See Table 1 for a brief overview of these results.

Lastly, we obtain lower bounds for several of the leading coefficients. We extend the lower bound of [18] for alternative basis $\langle 2,2,2;7\rangle$-algorithms, showing that even in our new framework, the leading coefficient of any $\langle 2,2,2;7\rangle$-algorithm is at least 5, matching the best known coefficient. Furthermore, we show that the leading coefficient of any $\langle n,m,k;t\rangle$-algorithm in our framework is at least 2, matching several of our obtained algorithms.

1.3 Paper Organization

In Section 2 we recall some preliminaries regarding recursive-bilinear algorithms. In Section 3 we describe our Decomposed Recursive-Bilinear Algorithm. We prove its correctness, and provide an analysis of its complexity. In Section 4 we show how our approach can be applied to reduce the coefficients of low-order monomials as well. In Section 5 we prove lower bounds on the leading coefficient, and present algorithms obtaining this bound. We also prove a lower bound on the leading coefficient of $\langle 2,2,2;7\rangle$-algorithms. In Section 6 we present our sparsification method. Section 7 contains a discussion of this work. Lastly, in Appendix A we provide an example of a decomposed algorithm.

2 PRELIMINARIES

2.1 Notations

Notation 2.1. *Let $t \in \mathbb{N}$. The notation $[t]$ represents the set:*

$$[t] = \{1, 2, \ldots, t\}$$

Notation 2.2. *Let R be a ring and let $l, n, m \in \mathbb{N}$. Denote $N = n^l, M = m^l$. Let $A \in R^{N\times M}$ be a matrix. Denote $A_{i,j}$ the (i,j)-th block of size $\frac{N}{n} \times \frac{M}{m}$. The block-row order vectorization of A corresponding to blocks of size $\frac{N}{n} \times \frac{M}{m}$, is recursively defined as follows:*

$$\vec{A} = (\vec{A}_{1,1} \ \ldots \ \vec{A}_{1,m} \ \ldots \ \vec{A}_{n,1} \ \ldots \ \vec{A}_{n,m})^T$$

Notation 2.3. *Denote the number of non-zero entries in a matrix by* $\mathrm{nnz}(A) = |\{x \in A : x \neq 0\}|$

Notation 2.4. *Denote the number of non-singleton entries in a matrix by* $\mathrm{nns}(A) = |\{x \in A : x \notin \{0, +1, -1\}\}|$.

Notation 2.5. *Let R be a ring and let $l, n, m \in \mathbb{N}$. Denote $N = n^l, M = m^l$. Let $a \in R^{NM}$ be a vector, and let:*

$$\forall i \in [nm] : a^{(i)} = (a_{\frac{NM}{nm}\cdot(i-1)+1}, \ldots, a_{\frac{NM}{nm}\cdot i})$$

The block segmentation of a is denoted:

$$\hat{a} = (a^{(1)}, \ldots, a^{(nm)})^T \in (R^{\frac{NM}{nm}})^{nm}$$

2.2 Recursive Bilinear Algorithms

Recursive bilinear algorithms use a divide-and-conquer strategy. They utilize a fixed-size base case, allowing fast computation of small inputs. For recursive-bilinear algorithms representing matrix multiplication, we denote them by their base case using the following notation.

Notation 2.6. *A recursive-bilinear matrix multiplication algorithm with a base case that multiplies matrices of dimension $n \times m$ and $m \times k$ using t scalar multiplications, is denoted by $\langle n,m,k;t\rangle$.*

Any such algorithm can be naturally extended into a recursive-bilinear algorithm which multiplies matrices of dimensions $n^l \times m^l, m^l \times k^l$, where $l \in \mathbb{N}$. The input matrices are first segmented into blocks of sizes $\frac{n^l}{n} \times \frac{m^l}{m}$, $\frac{m^l}{m} \times \frac{k^l}{k}$, respectively. Subsequently, linear combinations of blocks are performed directly, while scalar multiplication of blocks is computed via recursive invocations of the base algorithm. Once the blocks are decomposed into single scalars, multiplication is performed directly.

The asymptotic complexity of an $\langle n,n,n;t\rangle$-algorithm is $O(n^{\omega_0})$, where $\omega_0 = log_n(t)$. In the rectangular case, the exponent of an $\langle n,m,k;t\rangle$-algorithm is $\omega_0 = log_{nmk}(t^3)$.

Any bilinear algorithm, matrix multiplication included, can be described using three matrices, in the following form:

Fact 2.7. *Bilinear Representation: Let R be a ring, and let $n, m, k \in \mathbb{N}$. Let $f(x,y) : (R^{n\cdot m} \times R^{m\cdot k}) \to R^{n\cdot k}$ be a bilinear algorithm that performs t multiplications. There exist three matrices: $U \in R^{t\times n\cdot m}, V \in R^{t\times m\cdot k}, W \in R^{t\times n\cdot k}$, such that:*

$$\forall x \in R^{n\times m}, y \in R^{m\times k} : f(x,y) = W^T((U\cdot\vec{x}) \odot (V\cdot\vec{y}))$$

Where $\odot$ is the Hadamard (element-wise) product.

Definition 2.8. Let R be a ring, and let $U \in R^{t\times nm}$, $V \in R^{t\times mk}$, $W \in R^{t\times nk}$ be three matrices. A recursive-bilinear algorithm with the encoding matrices U, V and the decoding matrix W, is defined as follows:

Algorithm 1 Recursive-Bilinear Algorithm $ALG_{\langle U,V,W\rangle}$

Input: $a \in R^{(nm)^l}, b \in R^{(mk)^l}$
Output: $c = ALG_{\langle U,V,W\rangle}(a,b)$
1: **procedure** $ALG_{\langle U,V,W\rangle}(a,b)$
2: $\tilde{a} = U \cdot \hat{a}$ ▷ Transform inputs (Notation 2.5)
3: $\tilde{b} = V \cdot \hat{b}$
4: **if** $l = 1$ **then** ▷ Base Case
5: $\tilde{c} = W^T \cdot (\tilde{a} \odot \tilde{b})$ ▷ Scalar multiplication
6: **else**
7: **for** $r = 1$ to t **do**
8: $\tilde{c}[r] = ALG_{\langle U,V,W\rangle}(\tilde{a}[r], \tilde{b}[r])$ ▷ Recursion
9: **return** $W^T \cdot \tilde{c}$

Notation 2.9. *A recursive-bilinear algorithm defined by the matrices U,V,W is denoted by $ALG_{\langle U,V,W\rangle}$.*

The following necessary and sufficient condition characterizes the encoding and decoding matrices of matrix multiplication algorithms:

Fact 2.10. *Triple Product Condition [6]: Let R be a ring and let $m,n,k,t \in \mathbb{N}$. Let $U \in R^{t\times nm}, V \in R^{t\times mk}, W \in R^{t\times nk}$. For every $r \in [t]$, denote $U_{r,(i,j)}$ the element in the r'th row of U corresponding to the input element $A_{i,j}$. Similarly, $V_{r,(i,j)}$ corresponds to the input element $B_{i,j}$, and $W_{r,(i,j)}$ to the output element $(AB)_{i,j}$. U,V are the encoding matrices and W is the decoding matrix of an $\langle n,m,k;t\rangle$-algorithm if and only if:*

$$\forall i_1,i_2 \in [n], \forall k_1,k_2 \in [m], \forall j_1,j_2 \in [k]$$

$$\sum_{r=1}^{t} U_{r,(i_1,k_1)} V_{r,(k_2,j_1)} W_{r,(i_2,j_2)} = \delta_{i_1,i_2}\delta_{k_1,k_2}\delta_{j_1,j_2}$$

$$\text{Where } \delta_{i,j} = 1 \Leftrightarrow i = j$$

Claim 2.11. *Additive Complexity: [18] Encoding the inputs and decoding the outputs of an $\langle n,m,k;t\rangle$-algorithm using the corresponding encoding/decoding matrices U,V,W incurs an arithmetic cost. Let q_u, q_v, q_w be the number of arithmetic operations performed by the encoding and decoding matrices, correspondingly. Then:*

$$q_u = \text{nnz}(U) + \text{nns}(U) - \text{rows}(U)$$
$$q_v = \text{nnz}(V) + \text{nns}(V) - \text{rows}(V)$$
$$q_w = \text{nnz}(W) + \text{nns}(W) - \text{cols}(W)$$

We provide the proof of [18], for completeness.

Proof. Each row of U,V corresponds to a linear combination of A,B's elements. Each column of W corresponds to combinations of the multiplicands. The first non-zero entry in each row selects the first element to include in the combination (at no arithmetic cost). Each additional non-zero element indicates another element in the combination, requiring an additional arithmetic operation. If the entry is not a singleton, it requires an additional multiplication by a scalar, thus requiring two operations in total. □

3 DECOMPOSED RECURSIVE-BILINEAR ALGORITHM

In this section we present the Decomposed Recursive-Bilinear Algorithm. To this end, we first define the notion of fast recursive transformations, which will be key to our algorithm. Next, we present the algorithm itself, followed by a proof of its correctness, and an analysis of its arithmetic complexity.

3.1 Fast Recursive Transformation

By *Claim* 2.11, the additive complexities q_u, q_v, q_w are determined by the amount of non-zeros and non-singletons in the matrices U,V,W. Thus, sparsifying these matrices accelerates their corresponding algorithms. To this end, we now define a set of efficiently computable recursive transformations which will later be leveraged to increase the sparsity of the encoding/decoding matrices.

Definition 3.1. (Generalization of [18]). Let R be a ring. Let $\varphi_1 : R^{s_1} \to R^{s_2}$ be a linear transformation. Let $l \in \mathbb{N}$ and denote $S_1 = (s_1)^l$, $S_2 = (s_2)^l$. Let $v \in R^{S_1}$, and denote $\forall i \in [s_1] : v^{(i)} = (v_{\frac{S_1}{s_1}\cdot(i-1)+1}, \ldots, v_{\frac{S_1}{s_1}\cdot i})$. The linear map $\varphi_l : R^{S_1} \to R^{S_2}$ is recursively-defined as follows:

$$\varphi_l(v) = \varphi_1 \begin{pmatrix} \varphi_{l-1}(v^{(1)}) \\ \varphi_{l-1}(v^{(2)}) \\ \vdots \\ \varphi_{l-1}(v^{(s_1)}) \end{pmatrix}$$

Where φ_1 is applied to a vector of s_1 elements in $R^{\frac{S_1}{s_1}}$.

Applying the recursively-defined φ_l to the block-row order vectorization (as in Definition 2.2) of a matrix $A \in R^{N\times M}$ yields:

$$\varphi_l(\vec{A}) = \varphi_1 \left(\overrightarrow{\begin{pmatrix} \varphi_{l-1}(\vec{A}_{1,1}) & \ldots & \varphi_{l-1}(\vec{A}_{1,m}) \\ \vdots & \ddots & \vdots \\ \varphi_{l-1}(\vec{A}_{n,1}) & \ldots & \varphi_{l-1}(\vec{A}_{n,m}) \end{pmatrix}} \right)$$

3.2 Analysis of Recursive-Bilinear Algorithms

Fact 3.2. *Mixed-Product Property [36]: Denote by $\otimes$ the Kronecker product. Let $A \in R^{m_1\times n_1}$ and $C, D \in R^{m_2\times n_2}$ be two matrices. The following equality holds:*

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$$

Claim 3.3. *Let R be a ring. Let $\varphi_1 : R^{s_1} \to R^{s_2}$ be a linear transformation. Let $l \in \mathbb{N}$ and denote $S_1 = (s_1)^l, S_2 = (s_2)^l$. Let $\vec{v} \in R^{S_1}$. Denote by $\otimes$ the Kronecker product. Then:*

$$\varphi_l(\vec{v}) = \underbrace{(\varphi_1 \otimes \cdots \otimes \varphi_1)}_{l\ times} \cdot \vec{v} := (\otimes_l \varphi_1) \cdot \vec{v}$$

Proof. The proof is by induction on l. The base case ($l = 1$) is immediate, since:

$$\varphi_1(\vec{v}) = (\otimes_1 \varphi_1)\vec{v}$$

We next assume the claim holds for $(l-1) \in \mathbb{N}$ and show it holds for l. Observe that:

$$\begin{aligned}\varphi_l(\vec{v}) &= \varphi_1\left(\varphi_{l-1}(v^{(1)}), \varphi_{l-1}(v^{(2)}), \ldots, \varphi_{l-1}(v^{(s_1)})\right)^T \\ &= \varphi_1\left((\otimes_{l-1}\varphi_1)v^{(1)}, (\otimes_{l-1}\varphi_1)v^{(2)}, \ldots, (\otimes_{l-1}\varphi_1)v^{(s_1)}\right)^T \\ &= (\varphi_1 \otimes \varphi_{1-1}) \cdot \vec{v} := (\otimes_l \varphi_1) \cdot \vec{v}\end{aligned}$$

Where the first equality is by the definition of φ_l, the second is by the induction hypothesis, and the last equality is by the definition of the Kronecker product. □

Lemma 3.4. *Let R be a ring. Let $U \in R^{t\times nm}$, $V \in R^{t\times mk}$, $W \in R^{t\times nk}$ be three matrices and let $ALG_{\langle U,V,W\rangle}$ be a recursive-bilinear algorithm defined by U,V,W. Let $l \in \mathbb{N}$ and denote $N = n^l$, $M = m^l$, $K = k^l$. Let $a \in R^{NM}$ and $b \in R^{MK}$ be two vectors. Then:*

$$ALG_{\langle U,V,W\rangle}(a,b) = W_l^T \cdot ((U_l \cdot a) \odot (V_l \cdot b))$$

Proof. The proof is by induction on l. The base case ($l = 1$) is immediate, since by definition of a recursive-bilinear algorithm, $\forall x \in R^{nm}, \forall y \in R^{mk}$:

$$ALG_{\langle U,V,W\rangle}(x,y) = W^T \cdot ((U \cdot x) \odot (V \cdot y)) = W_1^T \cdot ((U_1 \cdot x) \odot (V_1 \cdot y))$$

We next assume the claim holds for $(l-1) \in \mathbb{N}$ and show it holds for l. Denote $\hat{a}$, $\hat{b}$ the block segmentations (Notation 2.5) of a and b, respectively. Let:

$$\forall i \in [t] : c_i = ALG_{\langle U,V,W\rangle}\left((U \cdot \hat{a})_i, (V \cdot \hat{b})_i\right)$$

By the induction hypothesis:

$$c_i = W_{l-1}^T \cdot ((U_{l-1} \cdot (U \cdot \hat{a})_i) \odot (V_{l-1} \cdot (V \cdot \hat{b})_i))$$

However:

$$\begin{pmatrix} U_{l-1} \cdot (U \cdot \hat{a})_1 \\ \vdots \\ U_{l-1} \cdot (U \cdot \hat{a})_t \end{pmatrix} = \begin{pmatrix} u_{1,1} \cdot U_{l-1} & \ldots & u_{1,nm} \cdot U_{l-1} \\ \vdots & \ddots & \vdots \\ u_{t,1} \cdot U_{l-1} & \ldots & u_{t,nm} \cdot U_{l-1} \end{pmatrix}_i \cdot \hat{a} = U_l \cdot a$$

Where the last equality follows by Claim 3.3. The same applies to V. Therefore by the definition of $ALG_{\langle U,V,W\rangle}$:

$$\begin{aligned} ALG_{\langle U,V,W\rangle}(a,b) &= W^T \cdot (c_1 \ldots c_t)^T \\ &= W^T \cdot \begin{pmatrix} W_{l-1}^T\left(U_{l-1} \cdot (U \cdot \hat{a})_1) \odot (V_{l-1} \cdot (V \cdot \hat{b})_1\right) \\ \vdots \\ W_{l-1}^T\left(U_{l-1} \cdot (U \cdot \hat{a})_t) \odot (V_{l-1} \cdot (V \cdot \hat{b})_t\right) \end{pmatrix} \\ &= W_l^T((U_l \cdot a) \odot (V_l \cdot b)) \end{aligned}$$

Where the last equality follows by Claim 3.3. □

3.3 Decomposed Bilinear Algorithm

Let U,V and W be the encoding/decoding matrices of a recursive-bilinear algorithm. Let $U = U_\varphi \cdot \varphi$, $V = V_\psi \cdot \psi$ and $W = W_\tau \cdot \tau$ be decompositions of the aforementioned matrices, and let $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$ be a recursive-bilinear algorithm defined by the encoding and decoding matrices U_φ, V_ψ and W_τ (Algorithm 1). Let $l \in \mathbb{N}$ and denote $N = n^l, M = m^l$. The Decomposed Recursive-Bilinear Algorithm is defined as follows:

Algorithm 2 Decomposed Recursive-Bilinear Algorithm

Input: $a \in R^N, b \in R^M$
Output: $c = ALG_{\langle U,V,W\rangle}(a,b)$

```
function DRB(a,b)
    ã = φ_l(a)                          ▷ Transform the first input
    b̃ = ψ_l(b)                          ▷ Transform the second input
    c̃ = ALG_⟨U_φ,V_ψ,W_τ⟩(ã, b̃)         ▷ Recursive-bilinear phase
    c = τ_l^T(c̃)                        ▷ Transform the output
return c
```

3.4 Correctness

In this section we prove that output of *DRB* (Algorithm 2) is identical to the output of a recursive-bilinear algorithm with the encoding and decoding matrices U,V and W (Algorithm 1).

Definition 3.5. Let $U \in R^{t\times nm}$, $V \in R^{t\times mk}$, $W \in R^{t\times nk}$ be three matrices. Let:

$$\begin{array}{lll} U_\varphi \in R^{t\times(t-r_u)} & \varphi \in R^{(t-r_u)\times nm} & r_u \in [t-nm] \\ V_\psi \in R^{t\times(t-r_v)} & \psi \in R^{(t-r_v)\times mk} & r_v \in [t-mk] \\ W_\tau \in R^{t\times(t-r_w)} & \tau \in R^{(t-r_w)\times nk} & r_w \in [t-nk] \end{array}$$

Where $U = U_\varphi \cdot \varphi$, $V = V_\psi \cdot \psi$, and $W = W_\tau \cdot \tau$. We refer to $U_\varphi, V_\psi, W_\tau, \varphi, \psi, \tau$ as a *decomposition of U, V, and W with levels r_u,r_v, and r_w, correspondingly.*

Lemma 3.6. *Let R be a ring. Let $l \in \mathbb{N}$ and denote $N = n^l$, $M = m^l$, $K = k^l$. Let $a \in R^{NM}$, $b \in R^{MK}$ be two vectors. Let $U \in R^{t\times nm}$, $V \in R^{t\times mk}$, and $W \in R^{t\times nk}$ be three matrices, and let $U_\varphi, V_\psi, W_\tau, \varphi, \psi, \tau$ be a decomposition of U, V, W with levels r_u,r_v,r_w, as in Definition 3.5. Let $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$ be a recursive-bilinear algorithm defined by $U_\varphi, V_\psi, W_\tau$, and denote $\tilde{a} = \varphi_l(a)$, $\tilde{b} = \psi_l(b)$. The following equality holds:*

$$ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}(\tilde{a},\tilde{b}) = {W_\tau}_l^T \cdot ((U_l \cdot a) \odot (V_l \cdot b))$$

Proof. $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$ is a recursive-bilinear algorithm which uses the encoding/decoding matrices $U_\varphi, V_\psi, W_\tau$, therefore by Lemma 3.4, $\forall x \in R^{NM}, \forall y \in R^{MK}$:

$$ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}(x,y) = {W_\tau}_l^T((U_{\varphi\,l} \cdot x) \odot (V_{\psi\,l} \cdot y))$$

Observe the following equality:

$$\begin{aligned} U_{\varphi\,l} \cdot \varphi_l &= (U_\varphi \otimes U_{\varphi\,l-1}) \cdot (\varphi \otimes \varphi_{l-1})) && \text{Claim 3.3} \\ &= (U_\varphi \varphi) \otimes (U_{\varphi\,l-1}\varphi_{l-1}) && \text{Fact 3.2} \\ &= U \otimes (U_{\varphi\,l-1}\varphi_{l-1}) && \text{Definition of U} \\ &= U \otimes \cdots \otimes U = U_l \end{aligned}$$

Similarly, $V_{\psi\,l} \cdot \psi_l = V_l$ and $W_{\tau\,l} \cdot \tau_l = W_l$. Therefore, applying $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$ to $\tilde{a}, \tilde{b}$ yields:

$$\begin{aligned} ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}(\tilde{a},\tilde{b}) &= {W_\tau}_l^T((U_{\varphi\,l} \cdot \tilde{a}) \odot (V_{\psi\,l} \cdot \tilde{b})) \\ &= {W_\tau}_l^T((U_{\varphi\,l} \cdot \varphi_l \cdot a) \odot (V_{\psi\,l} \cdot \psi_l \cdot b)) \\ &= {W_\tau}_l^T(((U_{\varphi\,l} \cdot \varphi_l) \cdot a) \odot ((V_{\psi\,l} \cdot \psi_l) \cdot b)) \\ &= W_\tau{}_l^T \cdot ((U_l \cdot a) \odot (V_l \cdot b)) \end{aligned}$$

□

Theorem 3.7. *Let R be a ring. Let $l \in \mathbb{N}$ and denote $N = n^l, M = m^l, K = k^l$. Let $U \in R^{t \times nm}$, $V \in R^{t \times mk}$, $W \in R^{t \times nk}$ be three matrices, and let $U_\varphi, V_\psi, W_\tau, \varphi, \psi, \tau$ be a decomposition of U, V, W with levels r_u, r_v, r_w, as in Definition 3.5. Let DRB be defined as in Algorithm 2, and let $ALG_{\langle U,V,W\rangle}$, $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$ be recursive-bilinear algorithms. The output of DRB satisfies:*

$$\forall a \in R^{NM}, \forall b \in R^{MK} : DRB(a,b) = ALG_{\langle U,V,W\rangle}(a,b)$$

Proof. Denote $\tilde{a} = \varphi_l(a)$, $\tilde{b} = \psi_l(b)$. By *Lemma* 3.6:

$$ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}(\tilde{a}, \tilde{b}) = {W_\tau}_l^T((U_l \cdot a) \odot (V_l \cdot b))$$

Therefore by the definition of *DRB*:

$$\begin{aligned} DRB(a,b) &= \tau_l^T \cdot ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}(\tilde{a}, \tilde{b}) \\ &= \tau_l^T \cdot {W_\tau}_l^T \cdot ((U_l \cdot a) \odot (V_l \cdot b)) \\ &= (W_{\tau l} \cdot \tau_l)^T \cdot ((U_l \cdot a) \odot (V_l \cdot b)) \\ &= W_l^T \cdot ((U_l \cdot a) \odot (V_l \cdot b)) \end{aligned}$$

Where the second equality follows from Lemma 3.6 and the fourth equality follows from the identity $W_{\tau l} \cdot \tau_l = W_l$ in Lemma 3.6. □

Corollary 3.8. *Let U,V and W be the encoding and decoding matrices of an $\langle n, m, k; t\rangle$-algorithm. Then $\forall A \in R^{n^l \times m^l}, \forall B \in R^{m^l \times k^l}$:*

$$DRB(\vec{A}, \vec{B}) = ALG_{\langle U,V,W\rangle}(\vec{A}, \vec{B}) = \overrightarrow{A \cdot B}$$

3.5 Arithmetic Complexity

We next analyze the arithmetic complexity of Algorithm 2. To this end, we first compute the arithmetic complexity of an $\langle n, m, k; t\rangle$-algorithm.

Claim 3.9. *Let R be a ring and let ALG be a recursive-bilinear $\langle n, m, k; t\rangle$-algorithm. Let $l \in \mathbb{N}$ and denote $N = n^l, M = m^l, K = k^l$. Let $A \in R^{N \times M}, B \in R^{M \times K}$ be two matrices. Let q_u,q_v,q_w be the additive complexities (as in Claim 2.11) of the encoding/decoding matrices. The arithmetic complexity of $ALG(A,B)$ is:*

$$\begin{aligned} F_{ALG}(N,M,K) = &\left(\frac{q_u}{t-nm} + \frac{q_v}{t-mk} + \frac{q_w}{t-nk} + 1\right) \cdot t^l \\ &- \frac{q_u}{t-nm} \cdot NM - \frac{q_v}{t-mk} \cdot MK - \frac{q_w}{t-nk} \cdot NK \end{aligned}$$

Proof. *ALG* is a recursive algorithm. In each step, *ALG* invokes t recursive calls on blocks of size $\frac{N}{n} \times \frac{M}{m}$ and $\frac{M}{m} \times \frac{K}{k}$. During the encoding phase, *ALG* performs q_u arithmetic operations on blocks of size $\frac{NM}{nm}$ and q_v operations on blocks of size $\frac{MK}{mk}$. During the decoding phase, q_w arithmetic operations are performed on blocks of size $\frac{NK}{nk}$. Therefore:

$$\begin{aligned} F_{ALG}(N,M,K) = &t \cdot F_{ALG}\left(\frac{N}{n}, \frac{M}{m}, \frac{K}{k}\right) + \\ &q_u \cdot \left(\frac{NM}{nm}\right) + q_v \cdot \left(\frac{MK}{mk}\right) + q_w \cdot \left(\frac{NK}{nk}\right) \end{aligned}$$

Moreover, $F_{ALG}(1,1,1) = 1$ since multiplying two scalar values requires a single arithmetic operation. Thus:

$$\begin{aligned} &F_{ALG}(N,M,K) \\ &= t \cdot F_{ALG}\left(\frac{N}{n}, \frac{M}{m}, \frac{K}{k}\right) + q_u \cdot \left(\frac{NM}{nm}\right) + q_v \cdot \left(\frac{MK}{mk}\right) + q_w \cdot \left(\frac{NK}{nk}\right) \\ &= \sum_{r=0}^{l-1} \left(\left(\frac{q_u NM}{(nm)^{r+1}} + \frac{q_v MK}{(mk)^{r+1}} + \frac{q_w NK}{(nk)^{r+1}}\right) t^r\right) + t^l \cdot F_{ALG}(1,1,1) \\ &= \frac{q_u NM}{nm} \sum_{r=0}^{l-1} \left(\frac{t}{nm}\right) + \frac{q_v MK}{mk} \sum_{r=0}^{l-1} \left(\frac{t}{mk}\right) + \frac{q_w NK}{nk} \sum_{r=0}^{l-1} \left(\frac{t}{nk}\right) + t^l \\ &= q_u \cdot \frac{t^l - NM}{t-nm} + q_v \cdot \frac{t^l - MK}{t-mk} + q_w \cdot \frac{t^l - NK}{t-nk} + t^l \\ &= \left(\frac{q_u}{t-nm} + \frac{q_v}{t-mk} + \frac{q_w}{t-nk} + 1\right) \cdot t^l \\ &\quad - \frac{q_u}{t-nm} \cdot NM - \frac{q_v}{t-mk} \cdot MK - \frac{q_w}{t-nk} \cdot NK \end{aligned}$$

□

Corollary 3.10. *Denote $q = q_u + q_v + q_w$. The arithmetic complexity of an $\langle n, n, n; t\rangle$-algorithm is:*

$$\left(\frac{q}{t-n^2} + 1\right) N^{log_n(t)} - \frac{q}{t-n^2} \cdot N^2$$

Proof. Observe that $l = log_n(N)$, thus $t^l = N^{log_n(t)}$. Substituting this equality in Claim 3.9 and letting $N = M = K$, yields the expression above. □

Claim 3.11. *Let R be a ring and let $\varphi_1 : R^{s_1} \to R^{s_2}$ be a linear transformation, where $s_1 \neq s_2$. Denote by q_φ the additive complexity (as in Claim 2.11) of φ_1. Let $l \in \mathbb{N}$ and denote $S_1 = (s_1)^l$, $S_2 = (s_2)^l$. Let $v \in R^{S_1}$ be a vector. The arithmetic complexity of computing $\varphi_l(v)$ is:*

$$F_\varphi(S_1) = \frac{q_\varphi}{s_1 - s_2} \cdot (S_1 - S_2)$$

Proof. $\varphi_l(v)$ is computed recursively. In each recursive call, φ_1 is invoked on each of the s_1 blocks of v, whose sizes are $\frac{S_1}{s_1}$. In each call, φ_1 performs q_φ arithmetic operations on the resulting blocks, whose sizes are $\frac{S_2}{s_2}$. Therefore:

$$F_\varphi(S_1) = s_1 \cdot F_\varphi\left(\frac{S_1}{s_1}\right) + q_\varphi \cdot \frac{S_2}{s_2}$$

Moreover, $F_\varphi(1) = 0$ since handling a single scalar value requires no operations. Thus:

$$\begin{aligned} F_\varphi(S_1) &= \sum_{r=0}^{l-1} (s_1)^r \cdot q_\varphi \frac{S_2}{(s_2)^{r+1}} \\ &= q_\varphi \cdot \frac{S_2}{s_2} \cdot \sum_{r=0}^{l-1} \left(\frac{s_1}{s_2}\right)^r \\ &= \frac{q_\varphi \cdot S_2}{s_2} \cdot \left(\frac{S_1}{S_2} - 1\right) \cdot \frac{s_2}{s_1 - s_2} = \frac{q_\varphi}{s_1 - s_2} \cdot (S_1 - S_2) \end{aligned}$$

The expression above holds for $s_1 \neq s_2$. If $s_1 = s_2$ the complexity is:

$$F_\varphi(S_1) = \sum_{r=0}^{l-1} (s_1)^r \cdot q_\varphi \frac{S_1}{(s_1)^{r+1}} = \frac{q_\varphi}{s_1} \cdot S_1 \cdot \log_{s_1} S_1$$ □

We now compute the arithmetic complexity incurred by the "core" of the algorithm; the recursive-bilinear $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$.

Claim 3.12. *Let R be a ring and let U,V and W be the matrices of an $\langle n,m,k;t\rangle$-algorithm, where $U \in R^{t\times nm}$, $V \in R^{t\times mk}$, $W \in R^{t\times nk}$, and let $U_\varphi, V_\psi, W_\tau, \varphi, \psi, \tau$ be a decomposition of U, V, W with levels r_u,r_v,r_w, as in Definition 3.5. Let q_{u_φ}, q_{v_ψ}, q_{w_τ} be the additive complexities of $U_\varphi, V_\psi, W_\tau$, correspondingly. Let $l \in \mathbb{N}$ and denote $(m_u, m_v, m_w) = (t-r_u, t-r_v, t-r_w)$ and $(M_u, M_v, M_w) = ({m_u}^l, {m_v}^l, {m_w}^l)$. Similarly, denote $(N,M,K) = (n^l, m^l, k^l)$. Let $A \in R^{N\times M}$, $B \in R^{M\times K}$ and denote $\tilde{A} = \varphi_l(A) \in R^{M_u}$, $\tilde{B} = \psi_l(B) \in R^{M_v}$. Let $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$ be a recursive-bilinear algorithm defined by the matrices U_φ, V_ψ and W_τ.*

The arithmetic complexity of $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}(\tilde{A}, \tilde{B})$ is:

$$F_{ALG}(M_u, M_v) = \left(\frac{q_{u_\varphi}}{r_u} + \frac{q_{v_\psi}}{r_v} + \frac{q_{w_\tau}}{r_w} + 1\right)\cdot t^l - \frac{q_{u_\varphi}}{r_u}\cdot M_u - \frac{q_{v_\psi}}{r_v}\cdot M_v - \frac{q_{w_\tau}}{r_w}\cdot M_w$$

Proof. $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$ is a recursive algorithm. In each step, $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$ performs t recursive calls (multiplications) of blocks of size $\frac{M_u}{m_u}$, $\frac{M_v}{m_v}$, producing blocks of size $\frac{M_w}{m_w}$. Encoding U requires q_{u_φ} arithmetic operations on blocks of size $\frac{M_u}{m_u}$. Similarly, encoding V requires q_{v_ψ} arithmetic operations on blocks of size $\frac{M_v}{m_v}$. Decoding the multiplicands requires q_{w_τ} arithmetic operations on blocks of size $\frac{M_w}{m_w}$. Therefore:

$$\begin{aligned}F_{ALG}(M_u, M_v) &= t\cdot F_{ALG}\left(\frac{M_u}{m_u}, \frac{M_v}{m_v}\right)\\ &\quad + q_{u_\varphi}\cdot\frac{M_u}{m_u} + q_{v_\psi}\cdot\frac{M_v}{m_v} + q_{w_\tau}\cdot\frac{M_w}{m_w}\end{aligned}$$

Furthermore, observe that $F_{ALG}(1,1) = 1$ since multiplying scalar values requires a single arithmetic operation. Therefore:

$$\begin{aligned}F_{ALG}(M_u, M_v) &= \left[\sum_{r=0}^{l-1}\left(q_{u_\varphi}\frac{M_u}{m_u^{r+1}} + q_{v_\psi}\frac{M_v}{m_v^{r+1}} + q_{w_\tau}\frac{M_w}{m_w^{r+1}}\right)\cdot t^r\right]\\ &\quad + \frac{q_{u_\varphi}M_u}{r_u}\left(\frac{t^l}{M_u} - 1\right) + \frac{q_{v_\psi}M_v}{r_v}\left(\frac{t^l}{M_v} - 1\right)\\ &\quad + \frac{q_{w_\tau}M_w}{r_w}\left(\frac{t^l}{M_w} - 1\right) + t^l\\ &= \left(\frac{q_{u_\varphi}}{r_u} + \frac{q_{v_\psi}}{r_v} + \frac{q_{w_\tau}}{r_w} + 1\right)\cdot t^l - \frac{q_{u_\varphi}}{r_u}\cdot M_u\\ &\quad - \frac{q_{v_\psi}}{r_v}\cdot M_v - \frac{q_{w_\tau}}{r_w}\cdot M_w\end{aligned}$$

□

Theorem 3.13. *Let R be a ring. Let $l \in \mathbb{N}$ and denote $N = n^l, M = m^l, K = k^l$. Let $A \in R^{N\times M}, B \in R^{M,K}$ be two matrices. Let DRB be as defined in Algorithm 2, and let $U_\varphi, V_\psi, W_\tau, \varphi, \psi, \tau$ be a decomposition of U, V, W with levels r_u,r_v,r_w, as in Definition 3.5. Let $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}$ be a recursive-bilinear algorithm defined by the matrices U_φ,V_ψ,W_τ. The arithmetic complexity of DRB(A,B) is:*

$$\begin{aligned}F_{DRB}(N,M,K) &= F_\varphi(NM) + F_\psi(MK) + F_{ALG}(M_u, M_v) + F_\tau(M_w)\\ &= \left(\frac{q_{u_\varphi}}{r_u} + \frac{q_{v_\psi}}{r_v} + \frac{q_{w_\tau}}{r_w} + 1\right)\cdot t^l\\ &\quad + \left(\frac{q_\varphi}{t-r_u-nm} - \frac{q_{u_\varphi}}{r_u}\right)\cdot(t-r_u)^l\\ &\quad + \left(\frac{q_\psi}{r-r_v-mk} - \frac{q_{v_\psi}}{r_v}\right)\cdot(t-r_v)^l\\ &\quad + \left(\frac{q_\tau}{t-r_w-nk} - \frac{q_{w_\tau}}{r_w}\right)\cdot(t-r_w)^l\\ &\quad - \frac{q_\varphi\cdot NM}{t-r_u-nm} - \frac{q_\psi\cdot MK}{t-r_v-mk} - \frac{q_\tau\cdot NK}{t-r_w-nk}\end{aligned}$$

Proof. We compute the arithmetic complexity by adding up the complexities of each stage. We use Claim 3.11 to compute the complexities of the two initial transformations $\varphi_l(\vec{A})$ and $\psi_l(\vec{B})$, and of the final transformation $\tau_l(\tilde{C})$. Using Claim 3.12 we compute the arithmetic complexity of $ALG_{\langle U_\varphi, V_\psi, W_\tau\rangle}(\tilde{A}, \tilde{B})$. Adding up all terms yields the expression above. □

Corollary 3.14. *The leading coefficient of DRB is:*

$$\frac{q_{u_\varphi}}{r_u} + \frac{q_{v_\psi}}{r_v} + \frac{q_{w_\tau}}{r_w} + 1$$

3.6 IO-Complexity

We next analyze the IO-Complexity of the fast recursive transformations (Subsection 3.1). Our analysis corresponds to the sequential model with two memory levels, where the fast memory is of size M. In this model the IO Complexity captures the number of transfers between the memory hierarchy, namely to and from the fast memory. Results of computations can be written out directly to the main memory without necessitating transfers from fast memory.

Claim 3.15. *(Generalization of [18]). Let R be a ring, and let $\varphi_1 : R^{s_1} \to R^{s_2}$ be a linear transformation. Denote by q_φ the additive complexity of φ_1. Let $l \in \mathbb{N}$ and denote $S_1 = (s_1)^l$, $S_2 = (s_2)^l$. Let $v \in R^{S_1}$ be a vector. Let $f = \log_{s_1}\frac{S_1}{\sqrt{\frac{M}{2}}}$. The IO-Complexity of computing $\varphi_l(v)$ is:*

$$IO_\varphi(S_1, M) = \left(\frac{3q_\varphi}{s_1 - s_2}\cdot\left(\left(\frac{s_1}{s_2}\right)^f - 1\right)\right)\cdot S_2 + \left(2\cdot\frac{M + (s_2)^f}{M}\right)\cdot S_1$$

Proof. The proof is similar to that of the arithmetic complexity (Claim 3.11), the main difference being the halting criteria and the complexity at the base-case. φ_l is computed recursively. At each step, φ_{l-1} is applied s_1 times to vectors of size $\frac{S_1}{s_1}$, producing vectors of size $\frac{S_2}{s_2}$. When $2S_1 \le M$, two input blocks fit inside the fast memory, requiring only M read operations, and the output is written out requiring S_2 writes. When the problem does not fit inside fast memory, each addition requires at most 3 data transfers: 2 reads for the inputs, and one write for the output. Therefore:

$$IO_\varphi(S_1, M) \le \begin{cases} s_1\cdot IO_\varphi\left(\frac{S_1}{s_1}, M\right) + 3q_\varphi\cdot\frac{S_2}{s_2} & 2S_1 > M\\ M + S_2 & 2S_1 \le M\end{cases}$$

Solving the recurrence we obtain:

$$IO_\varphi(S_1, M) \leq \sum_{r=0}^{f-1} \left(3q_\varphi \cdot \frac{(s_1)^r}{(s_2)^{l-r-1}}\right) + (M + (s_2)^f) \cdot \frac{2S_1}{M}$$
$$= 3q_\varphi \cdot (s_2)^{l-1} \sum_{r=0}^{f-1} \left(\frac{s_1}{s_2}\right)^r + (M + (s_2)^f) \cdot \frac{2S_1}{M}$$
$$= \left(\frac{3q_\varphi}{s_1 - s_2} \cdot \left(\left(\frac{s_1}{s_2}\right)^f - 1\right)\right) \cdot S_2 + \left(2 \cdot \frac{M + (s_2)^f}{M}\right) \cdot S_1$$

□

4 FULL DECOMPOSITION

In Section 3.3 we demonstrated a decomposition in which each of the encoding/decoding matrices of an $\langle n, m, k; t\rangle$-algorithm is split into a *pair* of matrices. We refer to such a decomposition as a *first-order decomposition*. First-order decompositions allowed a reduction of the leading coefficient, at the cost of introducing new low-order monomials. The same approach can then be repeatedly applied to the output of the decomposition, thus also reducing the coefficients of low-order monomials (see Figure 2).

Definition 4.1. Let $Q \in R^{t \times s}$ be an encoding or decoding matrix of an $\langle n, m, k; t\rangle$-algorithm. The c-order decomposition of Q is defined as:

$$Q = Q_\varphi \cdot \varphi^{(1)} \cdot \varphi^{(2)} \cdot \ldots \cdot \varphi^{(c)} = Q_\varphi \cdot \prod_{i=1}^{c} \varphi^{(i)}$$

Where $\varphi^{(i)} \in R^{h_i \times h_{i+1}}$, and $\forall i \in [c] : h_i > h_{i+1}$. Furthermore, $h_c = s$.

Interestingly, full decompositions may result in zero coefficients for some of the lower-order monomials. In the first-order decomposition, the decomposition level (see Definition 3.5) determines the degree of the lower-order monomial; higher decomposition levels yield lower-degree monomial incurred by the transformation cost (Claim 3.11). In a full decomposition, some lower-order monomials might cancel out altogether, as their transformation costs may cancel out some terms telescopically. See Table 2 for an example of the full decomposition of the $\langle 3, 3, 3; 23\rangle$-algorithm.

5 LOWER BOUNDS

5.1 Optimal Decomposition

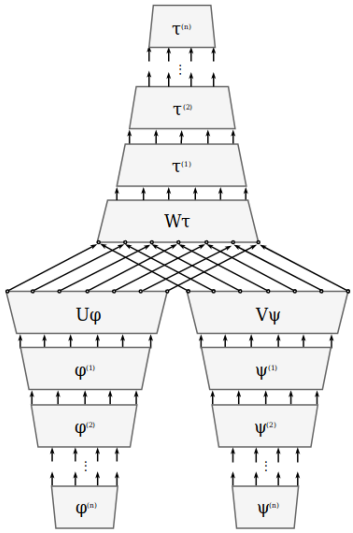
We decomposed the matrices corresponding to several matrix multiplication algorithms. Some algorithms exhibited an *optimal* decomposition, namely the leading coefficient of their arithmetic complexity is 2. This is optimal, as shown in the following claim:

Claim 5.1. *Let U,V,W be the encoding/decoding matrices of an $\langle n, m, k; t\rangle$-algorithm. W.l.o.g, none of U,V,W contain an all-zero row* [2]*. The leading coefficient of the arithmetic complexity of DRB (Algorithm 2) is at least 2.*

Proof. Let $U_\varphi, V_\psi, W_\tau, \varphi, \psi, \tau$ be a decomposition of U, V, W with levels r_u,r_v,r_w, as in Definition 3.5. By Claim 2.11, the additive

[2] Otherwise an $\langle n, m, k; t-1\rangle$-algorithm is automatically implied, see Claim 5.2

Figure 2: Full Decomposition Scheme



complexities satisfy:

$$q_{u_\varphi} = \text{nnz}(U_\varphi) + \text{nns}(U_\varphi) - \text{rows}(U_\varphi)$$
$$q_{v_\psi} = \text{nnz}(V_\psi) + \text{nns}(V_\psi) - \text{rows}(V_\psi)$$
$$q_{w_\tau} = \text{nnz}(W_\tau) + \text{nns}(W_\tau) - \text{cols}(W_\tau)$$

As U,V,W do not have all-zero rows, neither can U_φ, V_ψ, W_τ. Consequently, U_φ, V_ψ, W_τ all have at least one non-zero element in every row:

$$\text{nnz}(U_\varphi) \geq \text{rows}(U_\varphi)$$
$$\text{nnz}(V_\psi) \geq \text{rows}(V_\psi)$$
$$\text{nnz}(W_\tau) \geq \text{rows}(W_\tau)$$

Therefore:

$$q_{u_\varphi} \geq 0, \quad q_{v_\psi} \geq 0$$
$$q_{w_\tau} \geq \text{rows}(W_\tau) - \text{cols}(W_\tau) = r_w$$

The proof now follows from Corollary 3.14. □

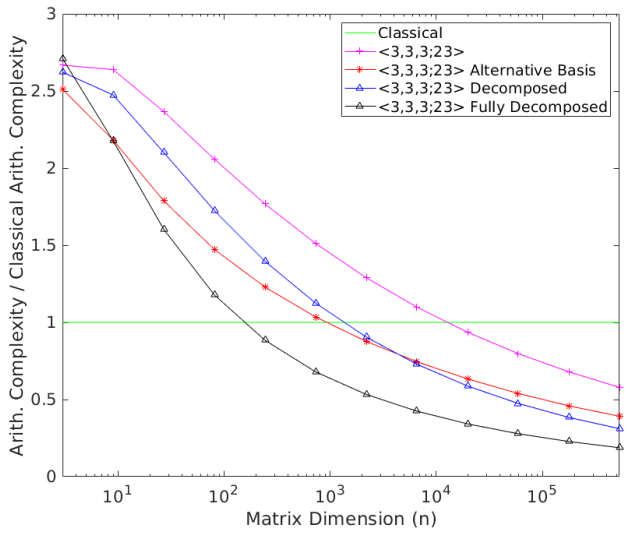
We note that all classical multiplication algorithms optimally decompose. However, the leading coefficient of classical algorithms is already 2 without decompositions, the minimal leading coefficient. Therefore, their decomposition does not allow for any further acceleration.

5.2 A Lower-Bound on the Leading Coefficient of $\langle 2, 2, 2; 7\rangle$-algorithms

Probert [27] and Bshouty [7] showed that 15 additions are necessary for any $\langle 2, 2, 2; 7\rangle$-algorithm, assuming the input and output are given in the standard basis. Karstadt and Schwartz proved a lower-bound of 12 arithmetic operations for any $\langle 2, 2, 2; 7\rangle$-algorithm, regardless of the input and output bases, thus showing their algorithm obtains the optimum.

In the decomposed matrix multiplication regime, the input and output are given in bases of a different *dimension*. This could have

Figure 3: Comparing the Arithmetic Complexity of the classical algorithm, $\langle 3,3,3;23\rangle$-algorithm [3, 17, 20], alternative basis $\langle 3,3,3;23\rangle$-algorithm [18], decomposed $\langle 3,3,3;23\rangle$-algorithm [here], and fully decomposed $\langle 3,3,3;23\rangle$-algorithm [here]



allowed for sidestepping the aforementioned lower bound, by requiring a smaller number of linear operations and thus, perhaps, a smaller leading coefficient. We prove that this is not the case. Namely, we prove that while 12 arithmetic operations are *not* required in this model (indeed 4 suffice), the leading coefficient of any $\langle 2,2,2;7\rangle$-algorithm remains at least 5, regardless of the decomposition level used.

Lemma 5.2. *Let Q be an encoding/decoding matrix of a $\langle 2,2,2;7\rangle$-algorithm. Q has no all-zero rows.*

Proof. Winograd showed [38] the minimal number of multiplications for any $\langle 2,2,2\rangle$-algorithm is 7. Assume towards a contradiction that Q is an encoding matrix with an all zeros row. Thus, the corresponding multiplicand is zero, allowing the output to be computed using only 6 multiplications, in contradiction to Winograd's lower bound. Similarly, if Q is a decoding matrix with an all zeros row, the corresponding multiplicand would always be discarded, once again allowing 6 multiplications, in contradiction to Winograd's lower bound. □

Corollary 5.3. *Q_φ has no all-zero rows, since a zero row in Q_φ implies such a row in Q.*

Lemma 5.4. *([16]) Let Q be an encoding/decoding matrix of a $\langle 2,2,2;7\rangle$-algorithm. Q has no duplicate rows.*

Corollary 5.5. *Q_φ has no duplicate rows, since duplicate rows in Q_φ imply duplicates in Q.*

Claim 5.6. *Let ALG be a $\langle 2,2,2;7\rangle$-algorithm. The leading coefficient of ALG is 5.*

Proof. Let $U,V,W \in R^{7\times 4}$ be the encoding/decoding matrices of *ALG*. Denote their decomposition as follows:

$$\begin{array}{lll} U_\varphi \in R^{7\times(t-r_u)} & \varphi \in R^{(7-r_u)\times 4} & r_u \in [3] \\ V_\psi \in R^{7\times(t-r_v)} & \psi \in R^{(7-r_v)\times 4} & r_v \in [3] \\ W_\tau \in R^{7\times(t-r_w)} & \tau \in R^{(7-r_w)\times 4} & r_w \in [3] \end{array}$$

For $r = 3$, the matrices φ,ψ,τ are square, therefore this case is identical to the Alternative Basis model, in which each encoding/decoding matrix must have at-least 10 non-zero elements [18], therefore:

$$\begin{aligned} q_{u_\varphi} &= \mathrm{nnz}(U_\varphi) - \mathrm{rows}(U_\varphi) \geq 10 - 7 = 3 \\ q_{v_\psi} &= \mathrm{nnz}(V_\psi) - \mathrm{rows}(V_\psi) \geq 10 - 7 = 3 \\ q_{w_\tau} &= \mathrm{nnz}(W_\tau) - \mathrm{cols}(W_\tau) \geq 10 - 4 = 6 \end{aligned}$$

Next, we handle the decomposition level $r = 2$. Let Q be an encoding/decoding matrix of a $\langle 2,2,2;7\rangle$-algorithm, and let $Q = Q_\varphi \cdot \varphi$, where $Q_\varphi \in R^{7\times 5}$, $\varphi \in R^{5\times 4}$. By Corollary 5.3, each of Q_φ's rows contain at least a single non-zero element. However, by Corollary 5.5, there are at most 5 such rows, therefore the remaining two rows must contain at-least two non-zero elements. Consequently:

$$\mathrm{nnz}(Q_\varphi) \geq 5 + 2 + 2 = 9$$

Thus, the corresponding additive complexities satisfy:

$$\begin{aligned} q_{u_\varphi} &= \mathrm{nnz}(U_\varphi) - \mathrm{rows}(U_\varphi) \geq 9 - 7 = 2 \\ q_{v_\psi} &= \mathrm{nnz}(V_\psi) - \mathrm{rows}(V_\psi) \geq 9 - 7 = 2 \\ q_{w_\tau} &= \mathrm{nnz}(W_\tau) - \mathrm{cols}(W_\tau) \geq 9 - 5 = 4 \end{aligned}$$

Lastly, we handle the decomposition level $r = 1$. Let Q be an encoding/decoding matrix of a $\langle 2,2,2;7\rangle$-algorithm. $Q = Q_\varphi \cdot \varphi$, where $Q_\varphi \in R^{7\times 6}$, $\varphi \in R^{6\times 4}$. Once again, by Corollaries 5.3 and 5.5, Q_φ has no duplicate rows, and at least one non-zero element in each row. Therefore, 6 of Q_φ's rows have at least one non-zero element, and the remaining row must contain at least 2 non-zeros. Therefore $\mathrm{nnz}(Q_\varphi) \geq 8$, and:

$$\begin{aligned} q_{u_\varphi} &= \mathrm{nnz}(U_\varphi) - \mathrm{rows}(U_\varphi) \geq 8 - 7 = 1 \\ q_{v_\psi} &= \mathrm{nnz}(V_\psi) - \mathrm{rows}(V_\psi) \geq 8 - 7 = 1 \\ q_{w_\tau} &= \mathrm{nnz}(W_\tau) - \mathrm{cols}(W_\tau) \geq 8 - 6 = 2 \end{aligned}$$

Putting the above terms together, we observe that irrespective of the decomposition dimension, the arithmetic costs satisfy:

$$\begin{aligned} cost(U_\varphi) &= \frac{q_{u_\varphi}}{r_u} \underset{r_u=3}{=} \frac{3}{3} \underset{r_u=2}{=} \frac{2}{2} \underset{r_u=1}{=} \frac{1}{1} = 1 \\ cost(V_\psi) &= \frac{q_{v_\psi}}{r_v} \underset{r_v=3}{=} \frac{3}{3} \underset{r_v=2}{=} \frac{2}{2} \underset{r_v=1}{=} \frac{1}{1} = 1 \\ cost(W_\tau) &= \frac{q_{w_\tau}}{r_w} \underset{r_w=3}{=} \frac{6}{3} \underset{r_w=2}{=} \frac{4}{2} \underset{r_w=1}{=} \frac{2}{1} = 2 \end{aligned}$$

Thus by Corollary 3.14, in all cases the leading coefficient is:

$$cost(U_\varphi) + cost(V_\psi) + cost(W_\tau) + 1 = 5$$ □

6 FINDING SPARSE DECOMPOSITIONS

As the additive complexity of an $\langle n, m, k; t\rangle$-algorithm is determined by the number of non-zero and non-singleton elements in its encoding/decoding matrices (see Claim 2.11), we seek *sparse* decompositions of the aforementioned matrices, preferably containing only singletons.

Formally, let $Q \in R^{t\times n}$ be an encoding or decoding matrix, and let $r \in [t-n]$. We seek a decomposition of Q into $Q_\varphi \in R^{t\times(t-r)}$, $\varphi \in R^{(t-r)\times n}$, satisfying:

$$\begin{aligned} &minimize: && \mathrm{nnz}(Q_\varphi) + \mathrm{nns}(Q_\varphi) \\ &subject\ to: && Q = Q_\varphi \cdot \varphi \end{aligned}$$

We focus here on minimizing non-zeros, for two main reasons. First, many encoding/decoding matrices contain only singleton values, and moreover our resulting decompositions have only singletons. Furthermore, minimizing the number of non-zeros also bounds the number of non-singletons, as $\mathrm{nns}(A) \le \mathrm{nnz}(A)$.

The optimization problem above whose objective is minimizing only the number of non-zeros is known as the Dictionary Learning problem, which is NP-Hard and hard to approximate [14, 35] within a factor of $2^{\log^{1-\epsilon} m}$, $\forall \epsilon > 0$ (unless $NP \subseteq DTIME\left(m^{poly(\log m)}\right)$). Nevertheless, due to the relatively small dimensions of many practical $\langle n, m, k; t\rangle$-algorithm base cases, the aforementioned problem can feasibly be tackled with currently available computational power.

Claim 6.1. *Let Q be an encoding or decoding matrix of an $\langle n, m, k; t\rangle$-algorithm, and let $r \in \mathbb{N}$ be the level decomposition we seek to obtain for Q. If Q has no all-zero rows, then Q_φ has non-zeros in every row and every column.*

Proof. By Corollary 5.3, if Q does not contain zero rows, neither does Q_φ. Assume towards a contradiction there exists an all-zero column in Q_φ. Then an $r-1$ decomposition is implied, since:

$$\begin{pmatrix} & i & \\ & | & \\ U'_\varphi & 0 & U''_\varphi \\ & | & \end{pmatrix} \begin{pmatrix} & \varphi' & \\ - & v_i & - \\ & \varphi'' & \end{pmatrix} = \begin{pmatrix} U'_\varphi & U''_\varphi \end{pmatrix} \begin{pmatrix} \varphi' \\ \varphi'' \end{pmatrix}$$

Thus Q_φ has non-zeros in every row and every column. □

The sparsest structure with non-zeros in every row and every column is a (possibly permuted) diagonal matrix $D_{(t-r)}$. Since we seek to minimize both the number of non-zeros and the number of non-singletons, we assume Q_φ contains a (possibly permuted) identity matrix. Let P_π be the permutation matrix which permutes Q_φ's rows such that the first $t-r$ rows contain the identity matrix. Then multiplying by P_π we get:

$$P_\pi \cdot Q = P_\pi \cdot Q_\varphi \cdot \varphi = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ \hline & \dots & \end{pmatrix} \cdot \varphi$$

Thus $\forall i \in [t-r] : \varphi_i = (P_\pi \cdot Q)_i$, and therefore φ is uniquely determined by the location of the identity matrix's rows. Put together, the sparsification process works as follows:

(1) Choose the location of the identity matrix rows in Q_φ
(2) Compute φ based on the above selection
(3) For every remaining row of v_i of Q_φ, solve:

$$v_i = \underset{\vec{x}\in R^{t-r}}{\mathrm{argmin}}\left(\left\{\mathrm{nnz}(\vec{x}) : \varphi^T \cdot x^T = (Q_i)^T\right\}\right)$$

The latter optimization problem is known as the Compressed Sensing problem, which is NP-Hard [13, 14]. Nevertheless, many algorithms (such as Basis Pursuit [9] and Orthogonal Matching Pursuit [26]) attempt to solve relaxations of the above problem. While these algorithms' optimization goals are different to that of Compressed Sensing, their outputs may converge under some conditions (i.e., the null-space property).

Due to the relatively small dimensions of the encoding/decoding matrices, we iterated through all possible placements of non-zero elements in $\vec{x}$, solving the corresponding least-squares instance for each such choice. This approach, while far slower than the aforementioned algorithms, resulted in far sparser solutions, as quite large portions of the solution-space were enumerated.

7 DISCUSSION

Our method for reducing the hidden constant of the arithmetic complexity of fast matrix multiplication utilizes a richer set of decompositions compared to [18], allowing for even faster practical algorithms. We inherit the same asymptotic complexity of the original fast matrix multiplication algorithms, while significantly improving their leading coefficients (Table 1).

Highly optimized implementations of the "classical" algorithm often outperform fast matrix multiplication algorithms for sufficiently small matrices. We have obtained fast matrix multiplication algorithms whose leading coefficients match that of the classical algorithm (Subsection 5.1), and may therefore outperform the classical algorithm even for relatively small matrices.

Iteratively applying the decomposition scheme (recall Section 4) allows us to reduce the coefficients of lower-order monomials. For algorithms in which the degrees of lower-order monomials are quite close to that of the leading monomial, this further optimization can significantly improve the arithmetic complexity (see Table 2 and Figure 3).

Our algorithm relies on a recursive divide-and-conquer strategy. Thus, the straight-forward serial recursive implementation (cf. [12]) matches the communication cost lower-bounds. For parallel implementations, the BFS-DFS method can be used to attain these lower bounds [2, 22, 23].

An optimal decomposition (see Section 5.1) of the $\langle 3, 3, 3; 23\rangle$-algorithm [3, 17, 20] can be found in Appendix A. Thanks to its leading coefficient of 2, the decomposed $\langle 3, 3, 3; 23\rangle$-algorithm can outperform the $\langle 2, 2, 2; 7\rangle$-algorithm on small matrices, despite its larger exponent. The $\langle 3, 3, 3; 23\rangle$-algorithm for which we present an optimal decomposition is due to Ballard and Benson [3]. All three encoding/decoding matrices contain duplicate rows, and thus optimally decompose (Subsection 5.1). In contrast, the $\langle 3, 3, 3; 23\rangle$-algorithm due to Laderman [20] contains no duplicate rows in any of the matrices, and therefore exhibits a leading coefficient of at-least 5 for any level of decomposition. Johnson and McLoughlin [17] described a parametric family of $\langle 3, 3, 3; 23\rangle$-algorithms. Any choice of parameters results in duplicate rows in the encoding matrices,

and moreover choosing $x = y = z = 1$ yields duplicate rows in all three matrices, thus resulting in an optimally decomposing algorithm, similar to that of Ballard and Benson.

In addition to Smirnov's $\langle 6, 3, 3; 40\rangle$-algorithm [30], we decomposed a $\langle 6, 3, 3; 40\rangle$-algorithm, of Tichavskỳ and Kováč [34]. The original algorithm has a leading coefficient of 79.28. We improved it to 7 (a reduction by 91.1%), the same leading coefficient we obtained for Smirnov's algorithm.

Future work includes exploring Pan's Trilinear Aggregating [24, 25] methods and their generated algorithms. We have optimally decomposed several of the aforementioned algorithms, including $\langle 70, 70, 70; 143640\rangle$-algorithm and $\langle 44, 44, 44; 39952\rangle$-algorithm. We intend to apply the same approach to more of Pan's algorithms, including $\langle 44, 44, 44; 36133\rangle$-algorithm [25].

Our current sparsification efforts were performed using modest computational power, thereby restricting the depth of decomposition explored. In future work, we intend to harness larger computational power in order to apply our methods and find additional practical matrix multiplication algorithms.

ACKNOWLEDGMENTS

We thank Grey Ballard for referring us to Tichavskỳ and Kováč's $\langle 6, 3, 3; 40\rangle$-algorithm. Research was supported by grants 1878/14, and 1901/14 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities). This research was supported by a grant from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel. This work was supported by The Federmann Cyber Security Center in conjunction with the Israel national cyber directorate.

REFERENCES

[1] Valerii Alekseev and Alexey Smirnov. 2013. On the exact and approximate bilinear complexities of multiplication of 4×2 and 2×2 matrices. *Proceedings of the Steklov Institute of Mathematics* 282, 1 (2013), 123–139.

[2] Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. 2012. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 193–204.

[3] Austin R Benson and Grey Ballard. 2015. A framework for practical parallel fast matrix multiplication. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 42–53.

[4] Dario Andrea Bini, Milvio Capovani, Francesco Romani, and Grazia Lotti. 1979. $O(n^{2.7799})$ Complexity for nxn approximate matrix multiplication. *Information processing letters* 8, 5 (1979), 234–235.

[5] Marco Bodrato. 2010. A Strassen-like matrix multiplication suited for squaring and higher power computation. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*. ACM, 273–280.

[6] Richard P Brent. 1970. *Algorithms for matrix multiplication.* Technical Report. Stanford University CA Dept. of Computer Science.

[7] Nader H Bshouty. 1995. On the additive complexity of 2×2 matrix multiplication. *Information processing letters* 56, 6 (1995), 329–335.

[8] Murat Cenk and M Anwar Hasan. 2017. On the arithmetic complexity of Strassen-like matrix multiplications. *Journal of Symbolic Computation* 80 (2017), 484–501.

[9] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. 2001. Atomic decomposition by basis pursuit. *SIAM review* 43, 1 (2001), 129–159.

[10] Don Coppersmith and Shmuel Winograd. 1982. On the asymptotic complexity of matrix multiplication. *SIAM J. Comput.* 11, 3 (1982), 472–492.

[11] Don Coppersmith and Shmuel Winograd. 1990. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation* 9, 3 (1990), 251–280.

[12] Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran. 2012. Cache-oblivious algorithms. *ACM Transactions on Algorithms (TALG)* 8, 1 (2012), 4.

[13] Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA.

[14] Lee-Ad Gottlieb and Tyler Neylon. 2010. Matrix sparsification and the sparse null space problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 205–218.

[15] John Hopcroft and Jean Musinski. 1973. Duality applied to the complexity of matrix multiplication and other bilinear forms. *SIAM J. Comput.* 2, 3 (1973), 159–173.

[16] John E Hopcroft and Leslie R Kerr. 1971. On minimizing the number of multiplications necessary for matrix multiplication. *SIAM J. Appl. Math.* 20, 1 (1971), 30–36.

[17] Rodney W Johnson and Aileen M McLoughlin. 1986. Noncommutative Bilinear Algorithms for 3×3 Matrix Multiplication. *SIAM J. Comput.* 15, 2 (1986), 595–603.

[18] Elaye Karstadt and Oded Schwartz. 2017. Matrix multiplication, a little faster. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 101–110.

[19] Julian Laderman, Victor Pan, and Xuan-He Sha. 1992. On practical algorithms for accelerated matrix multiplication. *Linear Algebra and Its Applications* 162 (1992), 557–588.

[20] Julian D Laderman. 1976. A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. *Bull. Amer. Math. Soc.* 82, 1 (1976), 126–128.

[21] François Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*. ACM, 296–303.

[22] Benjamin Lipshitz, Grey Ballard, James Demmel, and Oded Schwartz. 2012. Communication-avoiding parallel Strassen: Implementation and performance. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–11.

[23] William F McColl and Alexandre Tiskin. 1999. Memory-efficient matrix multiplication in the BSP model. *Algorithmica* 24, 3-4 (1999), 287–297.

[24] Victor Y. Pan. 1978. Strassen's algorithm is not optimal trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations. In *Foundations of Computer Science, 1978., 19th Annual Symposium on*. IEEE, 166–176.

[25] Victor Ya Pan. 1982. Trilinear aggregating with implicit canceling for a new acceleration of matrix multiplication. *Computers & Mathematics with Applications* 8, 1 (1982), 23–34.

[26] Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. 1993. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*. IEEE, 40–44.

[27] Robert L Probert. 1976. On the additive complexity of matrix multiplication. *SIAM J. Comput.* 5, 2 (1976), 187–203.

[28] Francesco Romani. 1982. Some properties of disjoint sums of tensors related to matrix multiplication. *SIAM J. Comput.* 11, 2 (1982), 263–267.

[29] Arnold Schönhage. 1981. Partial and total matrix multiplication. *SIAM J. Comput.* 10, 3 (1981), 434–455.

[30] Alexey Smirnov. 2013. The bilinear complexity and practical algorithms for matrix multiplication. *Computational Mathematics and Mathematical Physics* 53, 12 (2013), 1781–1795.

[31] Andrew James Stothers. 2010. On the complexity of matrix multiplication. (2010).

[32] Volker Strassen. 1969. Gaussian elimination is not optimal. *Numerische mathematik* 13, 4 (1969), 354–356.

[33] Volker Strassen. 1986. The asymptotic spectrum of tensors and the exponent of matrix multiplication. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 49–54.

[34] Petr Tichavskỳ and Teodor Kováč. 2015. Private communication with Ballard and Benson, see [3] for benchmarking. (2015).

[35] Andreas M Tillmann. 2015. On the computational intractability of exact and approximate dictionary learning. *IEEE Signal Processing Letters* 22, 1 (2015), 45–49.

[36] Charles F Van Loan. 2000. The ubiquitous Kronecker product. *Journal of computational and applied mathematics* 123, 1-2 (2000), 85–100.

[37] Virginia Vassilevska Williams. 2012. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. ACM, 887–898.

[38] Shmuel Winograd. 1971. On multiplication of 2×2 matrices. *Linear algebra and its applications* 4, 4 (1971), 381–388.

A OPTIMAL DECOMPOSITION OF <3,3,3;23>-ALGORITHM [3, 17, 20]

The following is a decomposition of the $\langle 3, 3, 3; 23\rangle$-algorithm due to Ballard and Benson [3]. See Section 7 for a brief discussion on the differences between this algorithm and the algorithms due to Laderman [20] and Johnson and McLoughlin [17].

$U_\varphi \in \mathbb{R}^{23\times 21}$

$$\begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&-1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}$$

$V_\psi \in \mathbb{R}^{23\times 21}$

$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&-1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}$$

$W_\tau \in \mathbb{R}^{23\times 21}$

$$\begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}$$

$\varphi \in \mathbb{R}^{21\times 9}$

$$\begin{pmatrix}
0&-1&0&0&0&0&0&0&0\\
0&0&-1&0&0&0&0&0&0\\
0&0&1&1&0&0&0&0&0\\
0&0&0&1&-1&0&1&-1&0\\
-1&0&-1&0&0&0&-1&0&0\\
0&0&0&1&-1&0&0&0&1\\
0&0&0&0&0&0&1&0&0\\
0&0&-1&-1&1&1&0&0&0\\
0&0&0&0&0&0&0&0&-1\\
0&1&0&-1&0&0&0&0&0\\
0&0&0&-1&0&-1&0&0&-1\\
0&0&0&1&-1&0&0&-1&0\\
0&0&0&0&0&-1&0&0&-1\\
1&-1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&0\\
1&-1&0&0&0&0&1&0&0\\
0&0&0&1&-1&0&0&0&0\\
-1&1&0&0&0&0&-1&1&1\\
0&0&0&0&1&0&0&1&0\\
0&0&0&0&0&0&0&0&1
\end{pmatrix}$$

$\psi \in \mathbb{R}^{21\times 9}$

$$\begin{pmatrix}
0&0&1&0&0&1&0&1&-1\\
1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0\\
0&0&0&0&0&-1&0&1&0\\
-1&-1&1&0&0&0&0&0&0\\
0&0&-1&0&0&-1&0&0&1\\
1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&-1&0&0&1\\
0&1&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&-1&1\\
1&0&0&0&-1&1&0&0&0\\
0&0&0&0&0&0&-1&0&0\\
1&0&0&0&0&0&0&0&0\\
0&0&0&1&1&-1&0&0&0\\
0&1&-1&0&1&-1&0&-1&1\\
1&1&0&0&0&1&0&0&0\\
0&0&0&0&-1&0&0&1&0\\
0&0&0&0&0&1&0&0&0\\
1&0&0&1&0&0&0&0&0\\
0&0&0&0&-1&1&1&1&0
\end{pmatrix}$$

$\tau \in \mathbb{R}^{21\times 9}$

$$\begin{pmatrix}
1&0&0&0&0&0&0&0&0\\
0&1&0&0&1&1&0&0&0\\
0&0&0&0&0&0&1&-1&0\\
0&0&-1&0&0&0&0&0&0\\
0&0&0&1&0&1&-1&1&0\\
0&0&-1&0&0&0&0&0&1\\
0&1&1&0&1&1&0&0&0\\
0&-1&-1&0&0&0&1&1&0\\
0&0&0&0&1&1&0&0&0\\
0&0&0&0&0&1&0&0&-1\\
0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&-1&0&0\\
0&0&0&1&0&0&0&0&0\\
0&0&0&0&-1&0&0&1&0\\
1&-1&0&0&0&0&0&0&0\\
0&0&0&-1&0&0&1&0&0\\
0&1&0&0&1&0&0&0&0\\
0&0&-1&0&0&0&0&1&1\\
0&0&0&-1&1&0&1&-1&0\\
0&0&0&0&0&0&0&1&1\\
0&0&0&1&0&0&0&0&0\\
0&0&0&-1&0&0&1&0&0
\end{pmatrix}$$

B ILLUSTRATION OF ENCODING/DECODING DIMENSIONS

Figure 4: Dimensions of Encoding/Decoding Transformations of Recursive-Bilinear, Alternative Basis [18], Decomposed [Here] and Fully Decomposed [Here] algorithms