Matrix Multiplication, a Little Faster

Regular Submission

Elaye Karstadt
The Hebrew University of Jerusalem
elayeek@cs.huji.ac.il

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

ABSTRACT

Strassen's algorithm (1969) was the first sub-cubic matrix multiplication algorithm. Winograd (1971) improved its complexity by a constant factor. Many asymptotic improvements followed. Unfortunately, most of them have done so at the cost of very large, often gigantic, hidden constants. Consequently, Strassen-Winograd's $O\left(n^{\log_2 7}\right)$ algorithm often outperforms other matrix multiplication algorithms for all feasible matrix dimensions. The leading coefficient of Strassen-Winograd's algorithm was believed to be optimal for matrix multiplication algorithms with 2×2 base case, due to a lower bound of Probert (1976).

Surprisingly, we obtain a faster matrix multiplication algorithm, with the same base case size and asymptotic complexity as Strassen-Winograd's algorithm, but with the coefficient reduced from 6 to 5. To this end, we extend Bodrato's (2010) method for matrix squaring, and transform matrices to an alternative basis. We prove a generalization of Probert's lower bound that holds under change of basis, showing that for matrix multiplication algorithms with a 2×2 base case, the leading coefficient of our algorithm cannot be further reduced, hence optimal. We apply our technique to other Strassen-like algorithms, improving their arithmetic and communication costs by significant constant factors.

CCS CONCEPTS

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

KEYWORDS

Fast Matrix Multiplication, Bilinear Algorithms

ACM Reference format:
Elaye Karstadt and Oded Schwartz. 2017. Matrix Multiplication, a Little Faster. In *Proceedings of SPAA '17, Washington DC, USA, July 24-26, 2017,* 10 pages.
DOI: 10.1145/3087556.3087579

SPAA '17, Washington DC, USA

978-1-4503-4593-4/17/07…$15.00
DOI: 10.1145/3087556.3087579

1 INTRODUCTION

Strassen's algorithm [37] was the first sub-cubic matrix multiplication algorithm, with complexity $O\left(n^{\log_2 7}\right)$. Winograd [40] reduced the leading coefficient from 7 to 6 by decreasing the number of additions and subtractions from 18 to 15. In practice, Strassen-Winograd's algorithm often performs better than some asymptotically faster algorithms [3] due to these smaller hidden constants. The leading coefficient of Strassen-Winograd's algorithm was believed to be optimal, due to a lower bound on the number of additions[1] for matrix multiplication algorithms with 2×2 base case, obtained by Probert [31].

We obtain a method for improving the practical performance of Strassen and Strassen-like fast matrix multiplication algorithms by improving the hidden constants inside the O-notation. To this end, we extend Bodrato's (2010) method for matrix squaring, and transform matrices to an alternative basis.

1.1 Strassen-like Algorithms

Strassen-like algorithms are a class of divide-and-conquer algorithms which utilize a base $\langle n_0, m_0, k_0; t\rangle$-algorithm: multiplying an $n_0\times m_0$ matrix by an $m_0\times k_0$ matrix using t scalar multiplications, where n_0, m_0, k_0 and t are positive integers. When multiplying an $n \times m$ matrix by an $m \times k$ matrix, the algorithm splits them into blocks (each of size $\frac{n}{n_0} \times \frac{m}{m_0}$ and $\frac{m}{m_0} \times \frac{k}{k_0}$, respectively), and works block-wise, according to the base algorithm. Additions and multiplication by scalar in the base algorithm are interpreted as block-wise additions. Multiplications in the base algorithm are interpreted as block-wise multiplication via recursion. We refer to a Strassen-like algorithm by its base case. Hence, an $\langle n, m, k; t\rangle$-algorithm may refer to either the algorithm's base case or the corresponding block recursive algorithm, as obvious from context.

1.2 Known Strassen-like algorithms

Since Strassen's original discovery, many fast matrix multiplication algorithms followed and improved the asymptotic complexity [6, 12, 13, 26, 29, 32, 33, 36, 38, 39]. Some of these improvements have come at the cost of very large, often gigantic, hidden constants. Le Gall [26] estimated that even if matrix multiplication could be done in $O\left(n^2\right)$ arithmetic operations, it is unlikely to be applicable as the base case sizes would have to be astronomical.

Recursive fast matrix multiplication algorithms with reasonable base case size for both square and rectangular matrices have been discovered [3, 20, 24, 25, 29, 30, 35]. Thus, they have manageable hidden constants, some of which are asymptotically faster than Strassen's algorithm. While many fast matrix multiplication algorithms fail to compete with Strassen's in practice due to their

[1] From here on, when referring to addition and subtraction count we say additions.

hidden constants. However, some have achieved competitive performance (e.g., Kaporin's [21] implementation of Laderman et al.'s algorithm [24]).

Recently, Smirnov presented several fast matrix multiplication algorithms derived by computer aided optimization tools [35], including an $\langle 6, 3, 3; 40\rangle$-algorithm with asymptotic complexity of $O\left(n^{\log_{54} 40^3}\right)$, faster than Strassen's algorithm. Ballard and Benson [3] later presented several additional fast Strassen-like algorithms, found using computer aided optimization tools as well. They implemented several Strassen-like algorithms, including Smirnov's $\langle 6, 3, 3; 40\rangle$-algorithm, on shared-memory architecture in order to demonstrate that Strassen and Strassen-like algorithms can outperform classical matrix multiplication in practice (such as, Intel's MKL), on modestly sized problems (at least up to n=13000), in a shared-memory environment. Their experiments also showed Strassen's algorithm outperforming Smirnov's algorithm in some of the cases.

1.3 Previous work

Bodrato [7] introduced the intermediate representation method, for repeated squaring and for chain matrix multiplication computations. This enables decreasing the number of additions between consecutive multiplications. Thus, he obtained an algorithm with a 2×2 base case, which uses 7 multiplications, and has a leading coefficient of 5 for chain multiplication and for repeated squaring, for every multiplication outside the first one. Bodrato also presented an invertible linear function which recursively transforms a $2^k \times 2^k$ matrix to and from the intermediate transformation. While this is not the first time that linear transformations are applied to matrix multiplication, the main focus of previous research on the subject was on improving asymptotic performance rather than reducing the number of additions [10, 17].

Very recently, Cenk and Hasan [8] showed a clever way to apply Strassen-Winograd's algorithm directly to $n \times n$ matrices by forsaking the uniform divide-and-conquer pattern of Strassen-like algorithms. Instead, their algorithm splits Strassen-Winograd's algorithm into two linear divide-and-conquer algorithms which recursively perform all pre-computations, followed by vector multiplication of their results, and finally performs linear post-computations to calculate the output. Their method enables reuse of sums, resulting in a matrix multiplication algorithm with arithmetic complexity of $5n^{\log_2 7} + 0.5 \cdot n^{\log_2 6} + 2n^{\log_2 5} - 6.5n^2$. However, this comes at the cost of increased communication costs and memory footprint.

1.4 Our contribution

We present the Alternative Basis Matrix Multiplication method, and show how to apply it to existing Strassen-like algorithms (see Sections 3). While basis transformation is, in general, as expensive as matrix multiplications, some can be performed very fast (e.g., Hadamard in $O\left(n^2 \log n\right)$ using FFT [11]). Fortunately, so is the case for our basis transformation (see Section 3.1). Thus, it is a worthwhile trade-off of reducing the leading coefficient in exchange of an asymptotically insignificant overhead (see Section 3.2). We provide analysis as to how these constants are affected and the impact on both arithmetic and IO-complexity.

We discuss the problem of finding alternative bases to improve Strassen-like algorithms (see Section 5), and present several improved variants of existing algorithms, most notable of which are the alternative basis variant of Strassen's $\langle 2, 2, 2; 7\rangle$-algorithm which reduces the number of additions from 15 to 12 (see Section 3.3), and the variant of Smirnov's $\langle 6, 3, 3; 40\rangle$-algorithm with leading coefficient reduced by about 83.2%.[2]

Theorem 1.1 (Probert's lower bound). *[31] 15 additions are necessary for any $\langle 2, 2, 2; 7\rangle$-algorithm.*

Our result seemingly contradicts Probert's lower bound. However, his bound implicitly assumes that the input and output are represented in the standard basis, thus there is not contradiction. We extend Probert's lower bound to account for alternative bases (see Section 4):

Theorem 1.2 (Basis invariant lower bound). *12 additions are necessary for any matrix multiplication algorithm that uses a recursive-bilinear algorithm with a 2×2 base case with 7 multiplications, regardless of basis.*

Our alternative basis variant of Strassen's algorithm performs 12 additions in the base case, matching the lower bound in Theorem 1.2. Hence, it is optimal.

2 PERLIMINARIES

2.1 The communication bottleneck

Fast matrix multiplication algorithms have lower IO-complexity than the classical algorithm. That is, they communicate asymptotically less data within the memory hierarchy and between processors. The IO-compleixty is measured as a function of the number of processors P, the local memory size M, and the matrix dimension n. Namely, the communication costs of a parallel $\langle n_0, n_0, n_0; t\rangle$-algorithm are $\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_{n_0} t} \frac{M}{P}\right)$ [1, 2, 4, 34]. Thus, parallel versions of Strassen's algorithm which minimize communication cost outperform the well tuned classical matrix multiplication in practice, both in shared-memory [3, 14, 23] and distributed-memory architectures [1, 16, 27].

Our $\langle 2, 2, 2; 7\rangle$-algorithm not only reduces the arithmetic complexity by 16.66%, but also the IO-complexity by 20%, compared to Strassen-Winograd's algorithm. Hence, performance gain should be in range of 16-20% on a shared-memory machine.

2.2 Encoding and Decoding matrices

Fact 2.1. Let R be a ring, and let $f : R^n \times R^m \to R^k$ be a bilinear function which performs t multiplications. There exist $U \in R^{t \times n}$, $V \in R^{t \times m}$, $W \in R^{t \times k}$ such that

$$\forall x \in R^n,\ y \in R^m\ f(x, y) = W^T\left((U \cdot x) \odot (V \cdot y)\right)$$

where $\odot$ is element-wise vector product (Hadamard product).

Definition 2.2. (Encoding/Decoding matrices). We refer to the $\langle U, V, W\rangle$ of a recursive-bilinear algorithm as its encoding/decoding matrices (where U, V are the encoding matrices and W is the decoding matrix).

[2] Files containing the encoding/decoding matrices and the corresponding basis transformations can be found at https://github.com/elayeek/matmultfaster.

Table 1: $\langle 2,2,2;7\rangle$-algorithms

Algorithm	Additions	Arithmetic Computations	I/O-Complexity
Strassen [37]	18	$7n^{\log_2 7} - 6n^2$	$6 \cdot \left(\frac{\sqrt{3}\cdot n}{\sqrt{M}}\right)^{\log_2 7} \cdot M - 18n^2 + 3M$
Strassen-Winograd [40]	15	$6n^{\log_2 7} - 5n^2$	$5 \cdot \left(\frac{\sqrt{3}\cdot n}{\sqrt{M}}\right)^{\log_2 7} \cdot M - 15n^2 + 3M$
Ours	12	$5n^{\log_2 7} - 4n^2 + 3n^2 \log_2 n$	$4 \cdot \left(\frac{\sqrt{3}\cdot n}{\sqrt{M}}\right)^{\log_2 7} \cdot M - 12n^2 + 3n^2 \cdot \log_2\left(\sqrt{2}\cdot \frac{n}{\sqrt{M}}\right) + 5M$

Table 2: Alternative Basis Algorithms

Algorithm	Linear Operations	Improved Linear Operations	Arithmetic Leading Coefficient	Improved Leading Coefficient	Computations Saved
$\langle 2,2,2;7\rangle$ [40]	15	12	6	5	16.6%
$\langle 3,2,3;15\rangle$[3]	64	52	15.06	7.94	47.3%
$\langle 2,3,4;20\rangle$[3]	78	58	9.96	7.46	25.6%
$\langle 3,3,3;23\rangle$[3]	87	75	8.91	6.57	26.3%
$\langle 6,3,3;40\rangle$[35]	1246	202	55.63	9.39	83.2%

Definition 2.3. Let R be a ring, and let $A \in R^{n\times m}$ a matrix. We denote the vectorization of A by $\vec{A}$. We use the notation $U_{\ell,(i,j)}$ when referring to the element in the $\ell' th$ row on the column corresponding with the index (i,j) in the vectorization of A. For ease of notation, we sometimes write $A_{i,j}$ rather than $\vec{A}_{(i,j)}$.[3]

Fact 2.4. (Triple product condition). [22] Let R be a ring, and let $U \in R^{t\times n\cdot m}$, $V \in R^{t\times m\cdot k}$, $W \in R^{t\times n\cdot k}$. $\langle U, V, W\rangle$ are encoding/decoding matrices of an $\langle n,m,k;t\rangle$-algorithm if and only if:

$$\forall i_1, i_2 \in [n]\,,\; k_1, k_2 \in [m]\,,\; j_1, j_2 \in [k]$$
$$\sum_{r=1}^{t} U_{r,(i_1,k_1)} V_{r,(k_2,j_1)} W_{r,(i_2,j_2)} = \delta_{i_1,i_2}\delta_{k_1,k_2}\delta_{j_1,j_2}$$

where $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise.

Notation 2.5. Denote the number of nonzero entries in a matrix by $nnz\,(A)$, and the number of rows/columns by $rows\,(A)\,, cols\,(A)$.

Remark 2.6. The number of linear operations used by a bilinear algorithm is determined by its encoding/decoding matrices. The number of additions performed by each of the encoding is:

$$\texttt{AdditionsU} = nnz\,(U) - rows\,(U)$$
$$\texttt{AdditionsV} = nnz\,(V) - rows\,(V)$$

The number of additions performed by the decoding is:

$$\texttt{AdditionsW} = nnz\,(W) - cols\,(W)$$

The number of scalar multiplication performed by each of the encoding/decoding is equal to the total number of matrix entries which are not 1, -1, and 0.

[3]All basis transformations and encoding/decoding matrices assume row-ordered vectorization of matrices.

3 ALTERNATIVE BASIS MATRIX MULTIPLICATION

Fast matrix multiplication algorithms are bilinear computations. The number of operations performed in the linear phases of such algorithms (the application of their encoding/decoding matrices $\langle U, V, W\rangle$ in the case of matrix multiplication, see Definition 2.2) depends on basis of representation. In this section, we detail how alternative basis algorithms work and address the effects of using alternative bases on arithmetic complexity and IO-complexity.

Definition 3.1. Let R be a ring and let ϕ, ψ, υ be automorphisms of $R^{n\cdot m}$, $R^{m\cdot k}$, $R^{n\cdot k}$ (respectively). We denote a Strassen-like algorithm which takes $\phi\,(A)\,,\,\psi\,(B)$ as inputs and outputs $\upsilon\,(A\cdot B)$ using t multiplications by $\langle n,m,k;t\rangle_{\phi,\psi,\upsilon}$-algorithm. If $n = m = k$ and $\phi = \psi = \upsilon$, we use the notation $\langle n,n,n;t\rangle_{\phi}$-algorithm. This notation extends the $\langle n,m,k;t\rangle$-algorithm notation as latter applies when the three basis transformations are the identity map.

Given a recursive-bilinear, $\langle n,m,k;t\rangle_{\phi,\psi,\upsilon}$-algorithm, *ALG*, alternative basis matrix multiplication works as follows:

Algorithm 1 Alternative Basis Matrix Multiplication Algorithm

Input: $A \in R^{n\times m}$, $B^{m\times k}$
Output: $n\times k$ matrix $C = A\cdot B$

```
1: function ABS(A, B)
2:     Ã = φ(A)                 ▷ R^{n×m} basis transformation
3:     B̃ = ψ(B)                 ▷ R^{m×k} basis transformation
4:     C̃ = ALG(Ã, B̃)            ▷ ⟨n,m,k;t⟩_{φ,ψ,υ}-algorithm
5:     C = υ^{-1}(C̃)            ▷ R^{n×k} basis transformation
6: return C
```

Lemma 3.2. *Let R be a ring, let $\langle U, V, W\rangle$ be the encoding/decoding matrices of an $\langle n,m,k;t\rangle$-algorithm, and let ϕ, ψ, υ be automorphisms of $R^{n\cdot m}$, $R^{m\cdot k}$, $R^{n\cdot k}$ (respectively). $\left\langle U\phi^{-1}, V\psi^{-1}, W\upsilon^{T}\right\rangle$ are encoding/decoding matrices of an $\langle n,m,k;t\rangle_{\phi,\psi,\upsilon}$-algorithm.*

Proof. $\langle U, V, W \rangle$ are encoding/decoding matrices of an $\langle n, m, k; t \rangle$-algorithm. Hence, for any $A \in R^{n\times m}$, $B \in R^{m\times k}$

$$W^T \left(\left(U \cdot \vec{A} \right) \odot \left(V \cdot \vec{B} \right) \right) = \overrightarrow{A \cdot B}$$

Hence,

$$\begin{aligned} v\left(\overrightarrow{A \cdot B}\right) &= v\left(W^T \left(U \cdot \vec{A} \odot V \cdot \vec{B}\right)\right) \\ &= \left(Wv^T\right)^T \left(U\phi^{-1} \cdot \phi\left(\vec{A}\right) \odot V\psi^{-1} \cdot \psi\left(\vec{B}\right)\right) \end{aligned}$$

□

Corollary 3.3. *Let R be a ring, and let ϕ, ψ, v be automorphisms of $R^{n\cdot m}$, $R^{m\cdot k}$, $R^{n\cdot k}$ (respectively). $\langle U, V, W \rangle$ are encoding/decoding matrices of an $\langle n, m, k; t \rangle_{\phi,\psi,v}$-algorithm if and only if $\left\langle U\phi, V\psi, Wv^{-T} \right\rangle$ are encoding/decoding matrices of an $\langle n, m, k; t \rangle$-algorithm.*

3.1 Fast basis transformation

Definition 3.4. Let R be a ring and let $\psi_1 : R^{n_0\times m_0} \to R^{n_0\times m_0}$ be a linear map. We recursively define a linear map $\psi_{k+1} : R^{n\times m} \to R^{n\times m}$ (where $n = n_0^{\ell_1}$, $m = m_0^{\ell_2}$ for some $\ell_1, \ell_2 \le k+1$) by $\left(\psi_{k+1}(A)\right)_{i,j} = \psi_k\left(\psi_1(A)\right)_{i,j}$, where $A_{i,j}$ are $\frac{n}{n_0} \times \frac{m}{m_0}$ sub-matrices.

Note that ψ_{k+1} is a linear map. For convenience, we omit the subscript of ψ when obvious from context.

Claim 3.5. *Let R be a ring, let $\psi_1 : R^{n_0\times m_0} \to R^{n_0\times m_0}$ be a linear map, and let $A \in R^{n\times m}$ (where $n = n_0^{k+1}$, $m = m_0^{k+1}$). Define $\tilde{A}$ by $\left(\tilde{A}\right)_{i,j} = \psi_k\left(A_{i,j}\right)$. Then $\psi_1\left(\tilde{A}\right) = \psi_{k+1}(A)$.*

Proof. ψ_1 is a linear map. Hence, for any $i \in [n_0]$, $j \in [m_0]$, $\left(\psi_1(A)\right)_{i,j}$ is a linear sum of elements of A. Therefore, there exist scalars $\left\{x_{r,\ell}^{(i,j)}\right\}_{r\in[n_0],\, \ell\in[m_0]}$ such that

$$\begin{aligned} \left(\psi_{k+1}(A)\right)_{i,j} =& \psi_k\left(\psi_1(A)\right)_{i,j} \\ =& \psi_k\left(\sum_{r,\ell} x_{r,\ell}^{(i,j)} \cdot A_{r,\ell}\right) \end{aligned}$$

By linearity of ψ_k

$$= \sum_{r,\ell} x_{r,\ell}^{(i,j)} \psi_k\left(A_{r,\ell}\right) = \left(\psi_1\left(\tilde{A}\right)\right)_{i,j}$$

□

Claim 3.6. *Let R be a ring, let $\psi_1 : R^{n_0\times m_0} \to R^{n_0\times m_0}$ be an invertible linear map, and let ψ_{k+1} as defined above. ψ_{k+1} is invertible and its inverse is $\left(\psi_{k+1}^{-1}(A)\right)_{i,j} = \psi_k^{-1}\left(\psi_1^{-1}(A)\right)_{i,j}$.*

Proof. Define $\tilde{A}$ by $\left(\tilde{A}\right)_{i,j} = \psi_k\left(A_{i,j}\right)$ and define ψ_{k+1}^{-1} by $\left(\psi_{k+1}^{-1}(A)\right)_{i,j} = \psi_k^{-1}\left(\psi_1^{-1}(A)\right)_{i,j}$. Then:

$$\left(\psi_{k+1}^{-1}\left(\psi_{k+1}(A)\right)\right)_{i,j} = \psi_k^{-1}\left(\psi_1^{-1}\left(\psi_{k+1}(A)\right)\right)_{i,j}$$

By Claim 3.5

$$= \psi_k^{-1}\left(\psi_1^{-1}\left(\psi_1\left(\tilde{A}\right)\right)\right)_{i,j} = \psi_k^{-1}\left(\tilde{A}\right)_{i,j}$$

By definition of $\tilde{A}$

$$= \psi_k^{-1}\left(\psi_k(A)_{i,j}\right) = A_{i,j}$$

□

We next analyze the arithmetic complexity and IO-complexity of fast basis transformations. For convenience and readability, we presented here the square case only. The analysis for rectangular matrices is similar.

Claim 3.7. *Let R be a ring, let $\psi_1 : R^{n_0\times n_0} \to R^{n_0\times n_0}$ be a linear map, and let $A \in R^{n\times n}$ where $n = n_0^k$. The arithmetic complexity of computing $\psi(A)$ is*

$$F_\psi(n) = \frac{q}{n_0^2} n^2 \log_{n_0} n$$

where q is the number of linear operations performed by ψ_1.

Proof. Let $F_\psi(n)$ be the number of additions required by ψ. Each step of the recursion consists of computing n_0^2 sub-problems and performs q additions of sub-matrices. Therefore, $F_\psi(n) = n_0^2 F_\psi\left(\frac{n}{n_0}\right) + q\left(\frac{n^2}{n_0}\right)$ and $F_\psi(1) = 0$. Thus,

$$\begin{aligned} F_\psi(n) &= n_0^2 F_\psi\left(\frac{n}{n_0}\right) + q\left(\frac{n}{n_0}\right)^2 \\ &= \sum_{k=0}^{\log_{n_0}(n)-1} \left(n_0^2\right)^k \left(q\left(\frac{n}{n_0^{k+1}}\right)^2\right) \\ &= \frac{q}{n_0^2} n^2 \cdot \sum_{k=0}^{\log_{n_0}(n)-1} \left(\frac{n_0^2}{n_0^2}\right)^k = \frac{q}{n_0^2} n^2 \cdot \log_{n_0}(n) \end{aligned}$$

□

Claim 3.8. *Let R be a ring and let $\psi_1 : R^{n_0\times n_0} \to R^{n_0\times n_0}$ be a linear map, and let $A \in R^{n\times n}$ where $n = n_0^k$. The IO-complexity of computing $\psi(A)$ is*

$$IO_\psi(n, M) \le \frac{3q}{n_0^2} n^2 \log_{n_0}\left(\sqrt{2}\frac{n}{\sqrt{M}}\right) + 2M$$

where q is the number of linear operations performed by ψ_1.

Proof. Each step of the recursion consists of computing n_0^2 sub-problems and performs q linear operations. The base case occurs when the problem fits entirely in the fast memory (or local memory in parallel setting), namely $2n^2 \le M$. Each addition requires at most 3 data transfers (one of each input and one for writing the output). Hence, a basis transformation which performs q linear operations at each recursive steps has the recurrence:

$$IO_\psi(n, M) \le \begin{cases} n_0^2 IO_\psi\left(\frac{n}{n_0}, M\right) + 3q \cdot \left(\frac{n}{n_0}\right)^2 & 2n^2 > M \\ 2M & otherwise \end{cases}$$

Therefore

$$
\begin{aligned}
IO_\psi(n,M) &\le n_0^2 IO_\psi\left(\frac{n}{n_0},M\right)+3q\left(\frac{n}{n_0}\right)^2\\
&= \sum_{k=0}^{\log_{n_0}\left(\frac{n}{\sqrt{\frac{M}{2}}}\right)-1}\left(n_0^2\right)^k\left(3q\left(\frac{n}{n_0^{k+1}}\right)^2\right)+2M\\
&= \frac{3q}{n_0^2}n^2\cdot\sum_{k=0}^{\log_{n_0}\left(\sqrt{2}\cdot\frac{n}{\sqrt{M}}\right)-1}\left(\frac{n_0^2}{n_0^2}\right)^k+2M\\
&= \frac{3q}{n_0^2}n^2\cdot\log_{n_0}\left(\sqrt{2}\cdot\frac{n}{\sqrt{M}}\right)+2M
\end{aligned}
$$

□

3.2 Computing matrix multiplication in alternative basis

Claim 3.9. *Let ϕ_1,ψ_1,υ_1 be automorphisms of $R^{n_0\times m_0}$, $R^{m_0\times k_0}$, $R^{n_0\times k_0}$ (respectively), and let ALG be an $\langle n_0,m_0,k_0;t\rangle_{\phi_1,\psi_1,\upsilon_1}$ algorithm. For any $A\in R^{n\times m}$, $B\in R^{m\times k}$:*

$$ALG\left(\phi_\ell(A),\psi_\ell(B)\right)=\upsilon_\ell(A\cdot B)$$

where $n=n_0^\ell$, $m=m_0^\ell$, $k=k_0^\ell$.

Proof. Denote $\tilde{C}=ALG\left(\phi_{\ell+1}(A),\psi_{\ell+1}(B)\right)$ and the encoding/decoding matrices of ALG by $\langle U,\ V,\ W\rangle$. We prove by induction on ℓ that $\tilde{C}=\upsilon_\ell(A\cdot B)$. For $r\in[t]$, denote

$$
\begin{aligned}
S_r &= \sum_{i\in[n_0],j\in[m_0]}U_{r,(i,j)}\left(\phi_1(A)\right)_{i,j}\\
T_r &= \sum_{i\in[m_0],j\in[k_0]}V_{r,(i,j)}\left(\psi_1(B)\right)_{i,j}
\end{aligned}
$$

The base case, $\ell=1$, holds by Lemma 3.2 since ALG is an $\langle n_0,m_0,k_0;t\rangle_{\phi_1,\psi_1,\upsilon_1}$-algorithm. Note that this means that for any $i\in[n_0]$, $j\in[k_0]$

$$
\begin{aligned}
\left(\upsilon_1(AB)\right)_{i,j} &= \left(W^T\left(\left(U\cdot\phi_1(A)\right)\odot\left(V\cdot\psi_1(B)\right)\right)\right)_{i,j}\\
&= \sum_{r\in[t]}W_{r,(i,j)}\left(S_r\cdot T_r\right)
\end{aligned}
$$

Next, we assume the claim holds for $\ell\in\mathbb{N}$ and show for $\ell+1$. Given input $\tilde{A}=\phi_{\ell+1}(A)$, $\tilde{B}=\psi_{\ell+1}(B)$, ALG performs t multiplications $P_1,\ldots,P_t$. For each multiplication P_r, its left hand side multiplicand is of the form

$$L_r=\sum_{i\in[n_0],j\in[m_0]}U_{r,(i,j)}\tilde{A}_{i,j}$$

By Definition 3.4, $\left(\phi_{\ell+1}(A)\right)_{i,j}=\phi_\ell\left(\phi_1(A)\right)_{i,j}$. Hence,

$$=\sum_{i\in[n_0],j\in[m_0]}U_{r,(i,j)}\left(\phi_\ell\left(\phi_1(A)\right)_{i,j}\right)$$

From linearity of ϕ_ℓ

$$
\begin{aligned}
&=\phi_\ell\left(\sum_{i\in[n_0],j\in[m_0]}U_{r,(i,j)}\left(\phi_1(A)\right)_{i,j}\right)\\
&=\phi_\ell\left(S_r\right)
\end{aligned}
$$

And similarly, the right hand multiplication R_r is of the form

$$R_r=\psi_\ell\left(T_r\right)$$

Note that for any $r\in[t]$, S_r, T_r are $n_0^\ell\times m_0^\ell$ and $m_0^\ell\times k_0^\ell$ matrices, respectively. Hence, by the induction hypothesis,

$$P_r=ALG\left(\phi_\ell(S_r),\psi_\ell(T_r)\right)=\upsilon_\ell\left(S_r\cdot T_r\right)$$

Each entry in the output $\tilde{C}$ is of the form:

$$
\begin{aligned}
\tilde{C}_{i,j} &= \sum_{r\in[t]}W_{r,(i,j)}P_r\\
&= \sum_{r\in[t]}W_{r,(i,j)}\upsilon_\ell\left(S_r\cdot T_r\right)
\end{aligned}
$$

By linearity of υ_ℓ

$$=\upsilon_\ell\left(\sum_{r\in[t]}W_{r,(i,j)}\left(S_r\cdot T_r\right)\right)$$

And, as noted in the base case:

$$\left(\upsilon_1(A\cdot B)\right)_{i,j}=\left(\sum_{r\in[t]}W_{r,(i,j)}\left(S_r\cdot T_r\right)\right)_{(i,j)}$$

Hence,

$$\tilde{C}_{i,j}=\upsilon_\ell\left(\upsilon_1(A\cdot B)\right)_{i,j}$$

Therefore, by Definition 3.4, $\tilde{C}=\upsilon_{\ell+1}(A\cdot B)$ □

Notation 3.10. When discussing an $\langle n_0,n_0,n_0;t\rangle_{\phi,\psi,\upsilon}$-algorithm, we denote $\omega_0=\log_{n_0}t$.

Claim 3.11. *Let ALG be an $\langle n_0,n_0,n_0;t\rangle_{\phi,\psi,\upsilon}$-algorithm which performs q linear operations at its base case. The arithmetic complexity of ALG is*

$$F_{ALG}(n)=\left(1+\frac{q}{t-n_0^2}\right)n^{\omega_0}-\left(\frac{q}{t-n_0^2}\right)n^2$$

Proof. Each step of the recursion consists of computing t subproblems and performs q linear operations (additions/multiplication by scalar) of sub-matrices. Therefore $F_{ALG}(n)=tF_\psi\left(\frac{n}{n_0}\right)+q\left(\frac{n}{n^0}\right)^2$ and $F_{ALG}(1)=1$. Thus,

$$
\begin{aligned}
F_{ALG}(n) &= \sum_{k=0}^{\log_{n_0}n-1}t^k\cdot q\cdot\left(\frac{n}{n_0^{k+1}}\right)^2+t^{\log_{n_0}n}\cdot F_{ALG}(1)\\
&=\frac{q}{n_0^2}n^2\cdot\sum_{k=0}^{\log_{n_0}n-1}\left(\frac{t}{n_0^2}\right)^k+n^{\omega_0}\\
&=\frac{q}{n_0^2}n^2\left(\frac{\left(\frac{t}{n_0^2}\right)^{\log_{n_0}n}-1}{\frac{t}{n_0^2}-1}\right)+n^{\omega_0}q\left(\frac{n^{\omega_0}-n^2}{t-n_0^2}\right)+n^{\omega_0}
\end{aligned}
$$

□

Claim 3.12. *Let* ALG *be an* $\langle n_0, n_0, n_0; t\rangle_{\phi,\psi,\upsilon}$*-algorithm which performs q linear operations at its base case. The IO-complexity of* ALG *is*

$$IO_{ALG}(n, M) \le \left(\frac{q}{t - n_0^2}\right)\left(M\left(\sqrt{3}\cdot\frac{n}{\sqrt{M}}\right)^{\omega_0} - 3n^2\right) + 3M$$

Proof. Each step of the recursion consists of computing n_0^2 sub-problems and performs q linear operations. The base case occurs when the problem fits entirely in the fast memory (or local memory in parallel setting), namely $3n^2 \le M$. Each addition requires at most 3 data transfers (one of each input and one for writing the output). Hence, a basis transformation which performs q linear operations at each recursive steps has the recurrence:

$$IO_{ALG}(n, M) \le \begin{cases} t \cdot IO_{\psi}\left(\frac{n}{n_0}, M\right) + 3q\cdot\left(\frac{n}{n_0}\right)^2 & 3n^2 > M \\ 3M & otherwise \end{cases}$$

Therefore

$$\begin{aligned} IO_{ALG}(n, M) &\le \sum_{k=0}^{\log_{n_0}\frac{n}{\sqrt{\frac{M}{3}}}-1} t^k \cdot 3q\left(\frac{n}{n_0^{k+1}}\right)^2 + 3M \\ &= \frac{3q}{n_0^2}n^2 \sum_{k=0}^{\log_{n_0}\frac{n}{\sqrt{\frac{M}{3}}}-1}\left(\frac{t}{n_0^2}\right)^k + 3M \\ &= \frac{3q}{n_0^2}n^2\left(\frac{\left(\frac{t}{n_0^2}\right)^{\log_{n_0}\frac{n}{\sqrt{\frac{M}{3}}}} - 1}{\frac{t}{n_0^2} - 1}\right) + 3M \\ &= q\left(\frac{M\left(\sqrt{3}\cdot\frac{n}{\sqrt{M}}\right)^{\omega_0} - 3n^2}{t - n_0^2}\right) + 3M \end{aligned}$$

□

Corollary 3.13. *If* ABS *(Algorithm 1) performs q linear operations at the base case, then its arithmetic complexity is*

$$F_{ABS}(n) = \left(1 + \frac{q}{t - n_0^2}\right)n^{\omega_0} - \left(\frac{q}{t - n_0^2}\right)n^2 + O\left(n^2 \log n\right)$$

Proof. The number of flops performed by the algorithm is the sum of: (1) the number of flops performed by the basis transformations (denoted ϕ, ψ, υ) and (2) the number of flops performed by the recursive bilinear algorithms *ALG*.

$$F_{ABS}(n) = F_{ALG}(n) + F_{\phi}(n) + F_{\psi}(n) + F_{\upsilon}(n)$$

The result immediately follows from Claim 3.7 and Claim 3.11 □

Corollary 3.14. *If* ABS *(Algorithm 1) performs q linear operations at the base case, then its IO-complexity is*

$$\begin{aligned} IO_{ALG}(n, M) \le &\left(\frac{3q}{t - n_0^2}\right)\left(M\left(\frac{\sqrt{3}\cdot n}{\sqrt{M}}\right)^{\omega_0} - n^2\right) \\ &+ 3M + O\left(n^2 \log\frac{n}{\sqrt{M}}\right) \end{aligned}$$

Proof. The IO-complexity is the sum of: (1) the IO-complexity of the recursive bilinear algorithms *ALG* and (2) the IO-complexity of the basis transformations (denoted ϕ, ψ, υ).

$$\begin{aligned} IO_{ABS}(n, M) = &IO_{ALG}(n, M) + IO_{\phi}(n, M) \\ &+ IO_{\psi}(n, M) + IO_{\upsilon}(n, M) \end{aligned}$$

The result immediately follows from Claim 3.8 and Claim 3.12. □

3.3 Optimal $\langle 2, 2, 2; 7\rangle$-algorithm

We now present a basis transformation $\psi_{opt} : R^4 \to R^4$ and an $\langle 2, 2, 2; 7\rangle_{\psi}$-algorithm which performs only 12 linear operations.

Notation 3.15. Let, ψ_{opt} refer to the following transformation:

$$\psi_{opt} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad \psi_{opt}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & -1 & 1 & 1 \end{pmatrix}$$

For convenience, when applying ψ to matrices, we omit the vectorization and refer to it as $\psi : R^{2\times 2} \to R^{2\times 2}$:

$$\psi_{opt}(A) = \psi_1\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} A_{1,1} & A_{1,2} - A_{2,1} + A_{2,2} \\ A_{21} - A_{2,2} & A_{1,2} + A_{2,2} \end{pmatrix}$$

Where $A_{i,j}$ can be ring elements or sub-matrices. ψ_{opt}^{-1} is defined analogously. Both ψ_{opt} and ψ_{opt}^{-1} extend recursively as in Definition 3.4.

$$\left\langle U_{opt}, V_{opt}, W_{opt}\right\rangle =$$

$$\left\langle \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & -1 & 0 \\ -1 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & -1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \right\rangle$$

Figure 1: $\langle U, V, W\rangle$ are the encoding/decoding matrices of our $\langle 2, 2, 2; 7\rangle_{\psi_{opt}}$-algorithm which performs 12 linear operations

Claim 3.16. $\left\langle U_{opt}, V_{opt}, W_{opt}\right\rangle$ *are encoding/decoding matrices of an* $\langle 2, 2, 2; 7\rangle_{\psi_{opt}}$*-algorithm.*

Proof. Observe that

$$\left\langle U_{opt}\cdot\psi_{opt}, V_{opt}\cdot\psi_{opt}, W_{opt}\cdot\psi_{opt}^{-T}\right\rangle =$$

$$\left\langle \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & -1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 1 & -1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & -1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 1 & -1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & -1 & -1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \right\rangle$$

It is easy to verify that $\left\langle U_{opt} \cdot \psi_{opt}, V_{opt} \cdot \psi_{opt}, W_{opt} \cdot \psi_{opt}^{-T} \right\rangle$ satisfy the triple product condition in Fact 2.4. Hence, they are encoding/decoding algorithm of an $\langle 2,2,2;7\rangle$-algorithm. By Corollary 3.3, the claim follows. □

Claim 3.17. *Let R be a ring, and let $A \in R^{n\times n}$ where $n = 2^k$. The arithmetic complexity of computing $\psi_{opt}(A)$ is*

$$F_{\psi_{opt}}(n) = n^2 \log_2 n$$

The same holds for computing $\psi_{opt}^{-1}(A)$.

Proof. Both ψ_{opt}, ψ_{opt}^{-1} perform $q = 4$ linear operations at each recursive step and has base case size of $n_0 = 2$. The lemma follows immediately from Claim 3.7. □

Claim 3.18. *Let R be a ring, and let $A \in R^{n\times n}$ where $n = 2^k$. The I/O-complexity of computing $\psi_{opt}(A)$ is*

$$IO_{\psi_{opt}}(n, M) \le 2n^2 \log_2 \left(\sqrt{2}\frac{n}{\sqrt{M}}\right) + 2M$$

Proof. Both ψ_{opt}, ψ_{opt}^{-1} perform $q = 4$ linear operations at each recursive step. The lemma follows immediately from Claim 3.8 with base case $n_0 = 2$. □

Corollary 3.19. *Our $\langle 2,2,2;7\rangle_{\psi_{opt}}$-algorithm's arithmetic complexity is $F_{opt}(n) = 5n^{\log_2 7} - 4n^2$.*

Proof. Our $\langle 2,2,2;7\rangle_{\psi_{opt}}$-algorithm has a 2×2 base case and performs 7 multiplications. Applying Fact 2.6 to its encoding/decoding matrices $\left\langle U_{opt}, V_{opt}, W_{opt} \right\rangle$, we see that it performs 12 linear operations. The result follows immediately from Claim 3.11 □

Corollary 3.20. *Our $\langle 2,2,2;7\rangle_{\psi_{opt}}$-algorithm's IO-complexity is*

$$IO_{opt}(n, M) \le 12 \cdot \left(\sqrt{3} \cdot M \left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} - 3n^2\right) + 3M$$

Proof. Our algorithm has a 2×2 base case and performs 7 multiplications. By applying Fact 2.6 to its encoding/decoding matrices (as shown in Figure 1), we see that it performs 12 linear operations. The result follows immediately from Claim 3.12. □

Corollary 3.21. *The arithmetic complexity of* ABS *(Algorithm 1) with our $\langle 2,2,2;7\rangle_{\psi_{opt}}$-algorithm is*

$$F_{ABS}(n) = 5n^{\log_2 7} - 4n^2 + 3n^2 \log_2 n$$

Proof. The proof is similar to that of Corollary 3.13 □

Corollary 3.22. *The IO-complexity of* ABS *(Algorithm 1) with our $\langle 2,2,2;7\rangle_{\psi_{opt}}$-algorithm is*

$$IO_{ALG}(n, M) \le 4 \cdot \left(\frac{\sqrt{3}\cdot n}{\sqrt{M}}\right)^{\log_2 7} \cdot M - 12n^2 + 3n^2 \cdot \log_2\left(\sqrt{2}\cdot\frac{n}{\sqrt{M}}\right) + 5M$$

Proof. The proof is similar to that of Corollary 3.14 □

Theorem 3.23. *Our $\langle 2,2,2;7\rangle_{\psi_{opt}}$-algorithm's sequential and parallel IO-complexity is bound by $\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0}\frac{M}{P}\right)$ (where P is the number of processors, 1 in the sequential case, and $\omega_0 = \log_2 7$).*

Proof. We refer to the undirected bipartite graph defined by the decoding matrix of our a Strassen-like algorithm as its decoding graph (i.e., the edge (i, j) exists if $W_{i,j} \ne 0$). In [2], Ballard et al. proved that for any square recursive-bilinear Strassen-like algorithm with $n_0 \times n_0$ base case which performs t multiplications, if the decoding graph is connected then these bounds apply with $\omega_0 = \log_{n_0} t$. The decoding graph of our algorithm is connected. Hence, the claim is true. □

4 BASIS-INVARIANT LOWER BOUND ON ADDITIONS FOR 2×2 MATRIX MULTIPLICATION

In this section we prove Theorem 1.2 which says that 12 additions are necessary to compute 2×2 matrix multiplication recursively with base case of 2×2 and 7 multiplications, irrespective of basis. Theorem 1.2 completes Probert's lower bound which says that for standard basis, 15 additions are required.

Definition 4.1. Denote the permutation matrix which swaps row-order for column-order of vectorization of an $I \times J$ matrix by $P_{I\times J}$.

Lemma 4.2. *[19] Let $\langle U, V, W\rangle$ be the encoding/decoding matrices of an $\langle m,k,n;t\rangle$-algorithm. Then $\langle WP_{n\times m}, U, VP_{n\times k}\rangle$ are encoding/decoding matrices of an $\langle n,m,k;t\rangle$-algorithm.*

We use the following results, shown by Hopcroft and Kerr [20]:

Lemma 4.3. *[20] If an algorithm for 2×2 matrix multiplication has k left (right) hand side multiplicands from the set $S = \{A_{1,1}, (A_{1,2} + A_{2,1}), (A_{1,1} + A_{1,2} + A_{2,1})\}$, where additions are done modulo 2, then it requires at least $6 + k$ multiplications.*

Corollary 4.4. *[20] Lemma 4.3 also applies for the following definitions of S:*

(1) $(A_{1,1} + A_{2,1}), (A_{1,2} + A_{2,1} + A_{2,2}), (A_{1,1} + A_{1,2} + A_{2,2})$
(2) $(A_{1,1} + A_{1,2}), (A_{1,2} + A_{2,1} + A_{2,2}), (A_{1,1} + A_{2,1} + A_{2,2})$
(3) $(A_{1,1} + A_{1,2} + A_{2,1} + A_{2,2}), (A_{1,2} + A_{2,1}), (A_{1,1} + A_{2,2})$
(4) $A_{2,1}, (A_{1,1} + A_{2,2}), (A_{1,1} + A_{2,1} + A_{2,2})$
(5) $(A_{2,1} + A_{2,2}), (A_{1,1} + A_{1,2} + A_{2,2}), (A_{1,1} + A_{1,2} + A_{2,1})$
(6) $A_{1,2}, (A_{1,1} + A_{2,2}), (A_{1,1} + A_{1,2} + A_{2,2})$
(7) $(A_{1,2} + A_{2,2}), (A_{1,1} + A_{2,1} + A_{2,2}), (A_{1,1} + A_{1,2} + A_{2,1})$
(8) $A_{2,2}, (A_{1,2} + A_{2,1}), (A_{1,2} + A_{2,1} + A_{2,2})$

Corollary 4.5. *Any 2×2 matrix multiplication algorithm where a left hand (or right hand) multiplicand appears at least twice (modulo 2) requires 8 or more multiplications.*

Proof. Immediate from Lemma 4.3 and Corollary 4.4 since it covers all possible linear sums of matrix elements, modulo 2. □

Fact 4.6. A simple counting argument shows that any 7×4 binary matrix with less than 10 non-zero entries has a duplicate row (modulo 2) or an all zero row.

Lemma 4.7. *Irrespective of basis transformations ϕ, ψ, υ, the encoding matrices U, V, of an $\langle 2,2,2;7\rangle_{\phi,\psi,\upsilon}$-algorithm contain no duplicate rows.*

Proof. Let $\langle U, V, W\rangle$ be encoding/decoding matrices of an $\langle 2,2,2;7\rangle_{\phi,\psi,\upsilon}$-algorithm. By Corollary 3.3, $\left\langle U\phi,\ V\psi,\ W\upsilon^{-T}\right\rangle$ are encoding/decoding matrices of a $\langle 2,2,2;7\rangle$-algorithm. Assume, w.l.o.g, that U contains a duplicate row, which means that $U\phi$ contains a duplicate row as well. In that case, $\left\langle U\psi,\ V\psi,\ W\upsilon^{-T}\right\rangle$ is a $\langle 2,2,2;7\rangle$-algorithm in which one of the encoding matrices contains a duplicate row, in contradiction to Corollary 4.5. □

Lemma 4.8. *Irrespective of basis transformations ϕ, ψ, υ, the encoding matrices U, V, of a $\langle 2,2,2;7\rangle_{\phi,\psi,\upsilon}$-algorithm have at least* 10 *non-zero entries.*

Proof. No row in U, V can be zeroed out since otherwise the algorithm would require less than 7 multiplications. The result then follows from Fact 4.6 and Lemma 4.7. □

Lemma 4.9. *Irrespective of basis transformations ϕ, ψ, υ, the decoding matrix W of an $\langle 2,2,2;7\rangle_{\phi,\psi,\upsilon}$-algorithm has at least* 10 *non-zero entries.*

Proof. Let $\langle U, V, W\rangle$ be encoding/decoding matrices of an $\langle 2,2,2;7\rangle_{\phi,\psi,\upsilon}$-algorithm and suppose by contradiction that W has less than 10 non-zero entries. W is a 7×4 matrix. By Fact 4.6, it either has a duplicate row (modulo 2) or an all zero row. Corollary 3.3 states that $\left\langle U\phi, V\psi, W\upsilon^{-T}\right\rangle$ are encoding/decoding matrices of a $\langle 2,2,2;7\rangle$-algorithm. By Lemma 4.2, because $\left\langle U\phi, V\psi, W\upsilon^{-T}\right\rangle$ define a $\langle 2,2,2;7\rangle$-algorithm, so does $\left\langle W\upsilon^{-T}P_{2\times 2}, U\phi, V\psi P_{2\times 2}\right\rangle$. Hence $W\upsilon^{-T}P_{2\times 2}$ is an encoding matrix of a $\langle 2,2,2;7\rangle$-algorithm which has duplicate or all zero rows, contradicting Corollary 4.5. □

Proof (of Theorem 1.2). Lemma 4.8 and 4.9 then show that each encoding/decoding matrix must contain at least 10 non-zero entries. By Remark 2.6, the number of additions used by each of the encoding matrix is $nnz\,(U) - rows\,(U)$ (and analogously for V) and the number of additions used by the decoding is $nnz\,(W) - cols\,(W)$. Hence, 12 additions are necessary for an $\langle 2,2,2;7\rangle_{\phi,\psi,\upsilon}$-algorithm irrespective of basis transformations ϕ, ψ, υ. □

5 OPTIMAL ALTERNATIVE BASES

To apply our alternative basis method to other Strassen-like matrix multiplication algorithms, we find bases which reduce the number of linear operations performed by the algorithm. As we mentioned in Fact 2.6, the non-zero entries of the encoding/decoding matrices determine the number of linear operations performed by an algorithm. Hence, we want our encoding/decoding matrices to be as sparse as possible, and ideally to have only entries of the form -1, 0 and 1. From Lemma 3.2 and Corollary 3.3 we see that any $\langle n,m,k;t\rangle$-algorithm and dimension compatible basis transformations ϕ, ψ, υ can be composed into an $\langle n,m,k;t\rangle_{\phi,\psi,\upsilon}$-algorithm. Therefore, the problem of finding a basis in which a Strassen-like algorithm performs the least amount of linear operations is closely tied to the Matrix Sparsification problem:

Problem 5.1. Matrix Sparsification Problem (MS): Let U be an $m\times n$ matrix of full rank, find an invertible matrix A such that

$$A = \underset{A\in GL_n}{\mathtt{argmin}}\,(nnz\,(UA))$$

That is, finding basis transformations for a Strassen-like algorithm consists of three independent MS problems. Unfortunately, MS is not only NP-Hard [28] to solve, but also NP-Hard to approximate to within a factor of $2^{\log^{.5-o(1)} n}$ [15] (Over $\mathbb{Q}$, assuming NP does not admit quasi-polynomial time deterministic algorithms). There seem to be very few heuristics for matrix sparsification (e.g., [9]), or algorithms under very limiting assumptions (e.g., [18]). Nevertheless, for existing Strassen-like algorithms with small base cases, the use of search heuristics to find bases which significantly sparsify the encoding/decoding matrices of several Strassen-like algorithms proved useful. Our resulting alternative basis Strassen-like algorithms are summarized in Table 2. Note, particularly, our alternative basis version of Smirnov's $\langle 6,3,3;40\rangle$-algorithm, which is asymptotically faster than Strassen's, where we have reduced the number of linear operations in the bilinear-recursive algorithm from 1246 to 202, thus reducing the leading coefficient by 83.2%.

6 IMPLEMENTATION AND PRACTICAL CONSIDERATIONS

6.1 Recursion cutoff point

Implementations of fast matrix multiplication algorithms often take several recursive steps then call the classical algorithm from a vendor-tuned library. This gives better performance in practice due to two main reasons: (1) the asymptotic improvement of Strassen-like algorithms makes them faster than classical algorithms by margins which increase with matrix size, and (2) vendor-tuned libraries have extensive built-in optimization, which makes them perform better than existing implementations of fast matrix multiplication algorithms on small matrices.

We next present theoretical analysis for finding the optimal number of recursive steps without tuning.

Claim 6.1. *Let* ALG *be an $\langle n,n,n;t\rangle$-algorithm with q linear operations at the base case. The arithmetic complexity of running ALG for ℓ steps, then switching to classical matrix multiplication is:*

$$F_{ALG}\,(n,\ell) = \frac{q}{t-n_0^2}\left(\left(\frac{t}{n_0^2}\right)^{\ell}-1\right)\cdot n^2 + t^{\ell}\left(2\left(\frac{n}{n_0^{\ell}}\right)^3-\left(\frac{n}{n_0^{\ell}}\right)^2\right)$$

Proof. Each step of the recursion consists of computing t subproblems and performs q linear operations. Therefore, $F_{ALG}\,(n,\ell) = t\cdot F_{ALG}\left(\frac{n}{n_0},\ell-1\right)+q\left(\frac{n}{n^0}\right)^2$ and $F_{ALG}\,(n,0) = 2n^3-n^2$. Thus

$$\begin{aligned}F_{ALG}\,(n,\ell) &= \sum_{k=0}^{\ell-1} t^k\cdot q\cdot\left(\frac{n}{n_0^{k+1}}\right)^2 + t^{\ell}\cdot F_{ALG}\left(\frac{n}{n_0^{\ell}}\right)\\ &= \frac{q}{n_0^2}n^2\left(\frac{\left(\frac{t}{n_0^2}\right)^{\ell}-1}{\frac{t}{n_0^2}-1}\right)+t^{\ell}\left(2\left(\frac{n}{n_0^{\ell}}\right)^3-\left(\frac{n}{n_0^{\ell}}\right)^2\right)\\ &= \frac{q}{t-n_0^2}\left(\left(\frac{t}{n_0^2}\right)^{\ell}-1\right)\cdot n^2+t^{\ell}\left(2\left(\frac{n}{n_0^{\ell}}\right)^3-\left(\frac{n}{n_0^{\ell}}\right)^2\right)\end{aligned}$$

□

When running an alternative basis algorithm for a limited number of recursive steps, the basis transformation needs to be computed only for the same number of recursive steps. If the basis transformation is computed for more steps than the alternative basis multiplication, the classical algorithm will compute incorrect results as it does not account for the input being represented in an alternative basis. This introduces a small saving in the runtime of basis transformation.

Claim 6.2. *Let R be a ring and let $\psi_1 : R^{n_0 \times n_0} \to R^{n_0 \times n_0}$ be an invertible linear map, let $A \in R^{n \times n}$ where $n = n_0^k$, and let $\ell \leq k$. The arithmetic complexity of computing $\psi_\ell(A)$ is*

$$F_\psi(n, \ell) = \frac{q}{n_0^2} n^2 \cdot \ell$$

Proof. Let $F_{\psi_\ell}(n)$ be the number of additions required by ψ_ℓ. Each recursive consists of computing n_0^2 sub-problems and performing q linear operations. Therefore, $F_\psi(n, \ell) = n_0^2 \cdot F_\psi\left(\frac{n}{n_0}, \ell - 1\right) + q \cdot \left(\frac{n}{n_0}\right)^2$ and $F_\psi(n, 0) = 0$

$$F_{\psi_\ell}(n) = \sum_{k=0}^{\ell-1} \left(n_0^2\right)^k \cdot q \left(\frac{n}{n_0^{k+1}}\right)^2$$
$$= \frac{q}{n_0^2} n^2 \cdot \sum_{k=0}^{\ell-1} \left(\frac{n_0^2}{n_0^2}\right)^k = \frac{q}{n_0^2} n^2 \cdot \ell$$

□

6.2 Performance experiments

We next present performance results for our $\langle 2, 2, 2; 7 \rangle_{\psi_{opt}}$-algorithm. All experiments were conducted on a single compute node of HLRS's Hazel Hen, with two 12-core (24 threads) Intel Xeon CPU E5-2680 v3 and 128GB of memory.

We used a straightforward implementation of both our algorithm and Strassen-Winograd's [40] algorithm using OpenMP. Each algorithm runs for a pre-selected number of recursive steps before switching to Intel's MKL DGEMM routine. Each DGEMM call uses all threads, matrix additions are always fully parallelized. All results are the median over 6 experiments.

In Figure 2 we see that our algorithm outperforms Strassen-Winograd's, with the margin of improvement increasing with each recursive step and nearing the theoretical improvement.

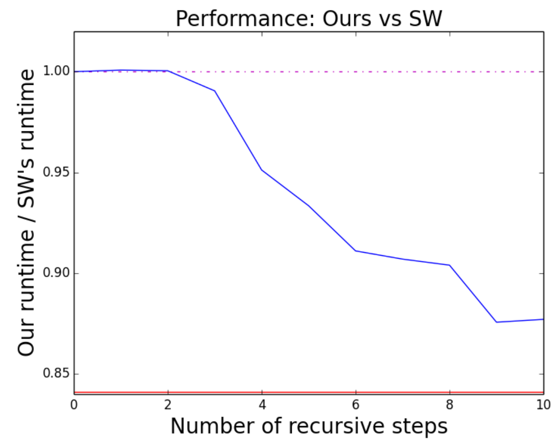

Figure 2: Comparing the performance of our $\langle 2, 2, 2; 7 \rangle_{\psi_{opt}}$-algorithm to Strassen-Winograd's on square matrices of fixed dimension $N = 32768$. The graph shows our algorithm's runtime, normalized by Strassen-Winograd's algorithm's runtime, as a function of the number of recursive steps taken before switching to Intel's MKL DGEMM. The top horizontal (at 1) line represents Strassen-Winograd's performance and the bottom horizontal line (at 0.83) represents the theoretical ratio when taking the maximal number of recursive steps.

7 DISCUSSION

Our method obtained novel variants of existing Strassen-like algorithms, reducing the number of linear operations required. Our algorithm also outperforms Strassen-Winograd's algorithm for any matrix dimension $n \geq 32$. Furthermore, we've obtained an alternative basis algorithm of Smirnov's $\langle 6, 3, 3; 40 \rangle$-algorithm, reducing the number of additions by 83.8%. While the problem of finding bases which optimally sparsify an algorithm's encoding/decoding matrices is NP-Hard (see Section 5), it is still solvable for many fast matrix multiplication algorithms with small base cases. Hence, finding basis transformations could be done in practice using search heuristics, leading to further improvements.

We leave large scale implementations for future research but note that both kernels of our alternative basis algorithms (basis transformation and recursive-bilinear algorithms) are known to be highly parallelizable recursive divide-and-conquer algorithms, and admit various communication minimizing parallelization techniques (e.g., [1, 5]).

ACKNOWLEDGMENTS

Research is supported by grants 1878/14, and 1901/14 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities) and grant 3-10891 from the Ministry of Science and Technology, Israel. Research is also supported by the Einstein Foundation and the Minerva Foundation. This work was supported by the PetaCloud industry-academia consortium. This research was supported by a grant from the United States-Israel Bi-national Science Foundation (BSF), Jerusalem, Israel. This work was supported by the HUJI Cyber Security Research Center in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. We acknowledge PRACE for awarding us access to Hazel Hen at GCS@HLRS, Germany.

REFERENCES

[1] 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.

[2] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. 2012. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM (JACM)* 59, 6 (2012), 32.

[3] Austin R Benson and Grey Ballard. 2015. A framework for practical parallel fast matrix multiplication. *ACM SIGPLAN Notices* 50, 8 (2015), 42–53.

[4] Gianfranco Bilardi and Lorenzo De Stefani. 2016. The I/O complexity of Strassen's matrix multiplication with recomputation. *arXiv preprint arXiv:1605.02224* (2016).

[5] Gianfranco Bilardi, Michele Scquizzato, and Francesco Silvestri. 2012. A Lower Bound Technique for Communication on BSP with Application to the FFT. In *European Conference on Parallel Processing*. Springer, 676–687.

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

[7] 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.

[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] S Frank Chang and S Thomas McCormick. 1992. A hierarchical algorithm for making sparse matrices sparser. *Mathematical Programming* 56, 1 (1992), 1–30.

[10] Henry Cohn and Christopher Umans. 2003. A group-theoretic approach to fast matrix multiplication. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*. IEEE, 438–449.

[11] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965), 297–301.

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

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

[14] Paolo D'alberto, Marco Bodrato, and Alexandru Nicolau. 2011. Exploiting parallelism in matrix-computation kernels for symmetric multiprocessor systems: Matrix-multiplication and matrix-addition algorithm optimizations by software pipelining and threads allocation. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 2.

[15] 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.

[16] Brian Grayson and Robert Van De Geijn. 1996. A high performance parallel Strassen implementation. *Parallel Processing Letters* 6, 01 (1996), 3–12.

[17] Vince Grolmusz. 2008. Modular representations of polynomials: Hyperdense coding and fast matrix multiplication. *IEEE Transactions on Information Theory* 54, 8 (2008), 3687–3692.

[18] Alan J Hoffman and ST McCormick. 1984. A fast algorithm that makes matrices optimally sparse. *Progress in Combinatorial Optimization* (1984), 185–196.

[19] John Hopcroft and Jean Musinski. 1973. Duality applied to the complexity of matrix multiplications and other bilinear forms. In *Proceedings of the fifth annual ACM symposium on Theory of computing*. ACM, 73–87.

[20] 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.

[21] Igor Kaporin. 2004. The aggregation and cancellation techniques as a practical tool for faster matrix multiplication. *Theoretical Computer Science* 315, 2-3 (2004), 469–510.

[22] Donald E Knuth. 1981. The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison-Wesley. *Reading, MA* (1981).

[23] Bharat Kumar, C-H Huang, P Sadayappan, and Rodney W Johnson. 1995. A tensor product formulation of Strassen's matrix multiplication algorithm with memory reduction. *Scientific Programming* 4, 4 (1995), 275–289.

[24] 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.

[25] Julian D Laderman. 1976. A noncommutative algorithm for multiplying 3× 3 matrices using 23 multiplications. In *Am. Math. Soc*, Vol. 82. 126–128.

[26] 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.

[27] Benjamin Lipshitz, Grey Ballard, James Demmel, and Oded Schwartz. 2012. Communication-avoiding parallel strassen: Implementation and performance. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 101.

[28] S Thomas McCormick. 1983. *A Combinatorial Approach to Some Sparse Matrix Problems.* Technical Report. DTIC Document.

[29] V Ya 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.

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

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

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

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

[34] Jacob Scott, Olga Holtz, and Oded Schwartz. 2015. Matrix multiplication I/O-complexity by path routing. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*. ACM, 35–45.

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

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

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

[38] 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.

[39] 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.

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