Alternative Basis Matrix Multiplication is Fast and Stable

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

Sivan Toledo
Tel Aviv University
Tel Aviv, Israel
stoledo@mail.tau.ac.il

Noa Vaknin
The Hebrew University of Jerusalem
Jerusalem, Israel
noa.vaknin1@mail.huji.ac.il

Gal Wiernik
Tel Aviv University
Tel Aviv, Israel
galwiernik@mail.tau.ac.il

***Abstract*—Alternative basis matrix multiplication algorithms are the fastest matrix multiplication algorithms in practice to date. However, are they numerically stable?**

We obtain the first numerical error bound for alternative basis matrix multiplication algorithms, demonstrating that their error bounds are asymptotically identical to the standard fast matrix multiplication algorithms, such as Strassen's. We further show that arithmetic costs and error bounds of alternative basis algorithms can be simultaneously and independently optimized. Particularly, we obtain the first fast matrix multiplication algorithm with a 2-by-2 base case that simultaneously attains the optimal leading coefficient for arithmetic costs and optimal asymptotic error bound, effectively beating the Bini and Lotti (1980) speed-stability trade-off for fast matrix multiplication. We provide high-performance parallel implementations of our algorithms with benchmarks that show our algorithm is on par with the best in class for speed and with the best in class for stability. Finally, we show that diagonal scaling stability improvement techniques for fast matrix multiplication are as effective for alternative basis algorithms, both theoretically and empirically. These findings promote the use of alternative basis matrix multiplication algorithms in practical applications.

***Index Terms*—Fast Matrix Multiplication, Alternative Basis Matrix Multiplication, Numerical Stability**

I. Introduction

Matrix multiplication is a fundamental computation kernel in many parallel and sequential scientific applications. Strassen [1] obtained the first sub-cubic algorithm for matrix-matrix multiplication, reducing the complexity of matrix multiplication from $\Theta(n^3)$ to $\Theta(n^{\log_2 7})$. Since then, the study of fast (sub-cubic) matrix multiplication algorithms has diverged into two main branches.

The focus of the first branch is on obtaining asymptotic improvements by minimizing the exponent of the arithmetic costs (see [2]–[9]). However, these algorithms are often associated with huge hidden constants, making them impractical. In addition, the relevant algorithms are generally only suitable for matrices with enormous dimensions, which limits their practical use. The second branch focuses on obtaining algorithms that maintain low hidden costs and are more likely to have practical applications, even if their asymptotics are sub-cubic but not optimal (cf. [10]–[23]).

Strassen's algorithm computes a 2×2 matrix product using seven multiplications instead of eight. Winograd [24] provided a variant of this algorithm that reduces the number of additions required from 18 to 15, thus reducing the leading coefficient of the arithmetic costs from 7 to 6. Probert [25] and Bshouty [26] demonstrated that 15 additions are necessary for any $\langle 2,2,2;7\rangle$-algorithm[1], thereby leading to the conclusion that the leading coefficient of Strassen-Winograd is optimal for the 2×2 base case. Extensive research followed this result, designed to reduce the leading coefficient of the arithmetic costs of fast matrix multiplication algorithms [10], [11], [13], [15], [27], [28]. More recently, so-called alternative basis methods were shown to be even faster in practice and to beat the lower bound; we describe them in Section I-B.

A. Stability of Fast Matrix Multiplication

For practical use, matrix multiplication must be numerically stable, not only fast. Modern numerical computations use floating-precision representation, which introduces numerical errors. The error bound of classical algorithms is component-wise, meaning that for every output element, the error bound depends solely on the corresponding row and column elements of the input matrices [29], [30].

Conversely, the error bound of fast algorithms is known as norm-wise [29], [30], meaning the error bound depends on the overall properties of the input matrices since they involve computations with entries not present in a given dot product (although their contributions cancel out in exact arithmetic). Miller [31] showed that element-wise stability is unattainable for fast algorithms since any polynomial algorithm achieving such bounds necessitates at least n^3 multiplications.

Bini and Lotti [32] provided a framework to obtain error bounds for the set of all fast matrix multiplication methods with a 2×2 base case and coefficients in $\mathbb{H} = \left\{0, \pm 2^i : i \in \mathbb{N}\right\}$. They define an algorithm's stability vector (see Definition III.1) and define an equivalence relation based on the stability vector, which divides the aforementioned set into 25 classes. Furthermore, they prove that the maximal entry of the stability vector, denoted the *stability factor*, determines the asymptotic error bound of a fast matrix multiplication algorithm. They conclude that Strassen's algorithm obtains the minimal stability factor in this set, while Winograd's variant yields a weaker bound despite having the fewest additions of all algorithms in the set. Higham [33] [34, chap. 23] provides a tighter analysis of the error bounds for both Strassen's

[1]See Section II-A for definition.

and Winograd's algorithms, demonstrating the same trade-off between their stability and arithmetic costs.

Castrapel and Gustafson [35] and D'Alberto [36] suggest reducing the stability factor of algorithms by using *non-stationary, non-uniform* matrix multiplication algorithms [37], that is, algorithms that vary recursive rules across different recursive levels and within each level.

Bini and Lotti's [32] bound holds for all coefficient matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ with elements in $\mathbb{H}$, and assumes that multiplication by entries of these matrices is error-free. Demmel et al. [38] prove a general error bound that assumes that round-off errors are introduced by every execution of any arithmetic operation at the cost of a looser stability factor (multiplied by $\|\mathbf{U}\| \cdot \|\mathbf{V}\| \cdot \|\mathbf{W}\|$ in max-norm). Ballard et al. [39] extend the work of Demmel et al. and prove a practical bound for the error analysis of algorithms. They also assume round-off errors in every arithmetic operation. They redefine the stability vector to consider the size of each entry in the encoding/decoding matrices to provide a tighter result. In addition, they consider rectangular matrices and the number of recursion levels before invoking classical matrix multiplication. Like Bini and Lotti, they also compare trade-offs between an algorithm's arithmetic cost and stability factor. Building on the work of [34], [40]–[42], Ballard et al. also offer a detailed approach to improving the stability of fast matrix multiplication algorithms using diagonal scaling techniques.

B. Alternative Basis Matrix Multiplication

Cenk and Hasan [15] reduced the leading coefficient of Strassen-Winograd's algorithmZ [24] by using a non-uniform divide-and-conquer pattern that allows to reuse sums. However, their method increases memory footprint and communication costs. Karstadt and Schwartz [10], [11] obtained a new $\langle 2,2,2;7\rangle$-algorithm that requires only 12 additions instead of 15, thereby presenting a leading coefficient of 5 (compared to 6 of Winograd [24]), when computing the multiplication in an alternative basis. While this result seemingly defies Probert [25] and Bshouty's [26] lower bounds, their bound implicitly assumes the input and output are in the standard basis. Karstadt and Schwartz extended these lower bounds by allowing an alternative basis and proved that their $\langle 2,2,2;7\rangle$-algorithm obtains an optimal leading coefficient of 5 for the 2×2 base case. Beniamini and Schwartz [12], [13] reduced the leading coefficient of various fast matrix multiplication algorithms even further. They introduced the decomposed recursive bilinear framework, which allows the basis transformations to be in a different dimension than the original matrix, thus creating a sparser bilinear operator. They proved new lower bounds that match these algorithms. Moran and Schwartz [28] lowered the arithmetic cost of multiplying small blocks by providing a method to replace scalar multiplications with additions.

C. High-performance Alternative Basis Matrix Multiplication

For many algorithms and hardware settings, communication costs are significantly higher than computational costs. Thus, high-performance parallel alternative basis matrix multiplication requires load-balancing of the computation and minimizing the communication costs, both between processors and within the memory hierarchy. Extensive research has been conducted in designing and implementing communication-effective classical and fast matrix multiplication algorithms [42]–[52]. Recently, Karppa and Kaski [53] implemented an alternative basis variant of Strassen's algorithm for binary and Boolean matrices on a GPU. Schwartz and Vaknin [48] implemented a variant for general matrix-matrix multiplication, outperforming Intel's MKL on small and large matrices.

Schwartz and Vaknin [48] experimentally demonstrated that their alternative basis algorithm and Strassen-Winograd's [24] algorithm exhibit the same numerical errors. However, it was not clear whether alternative basis matrix multiplication algorithms satisfy the same type of norm-wise error bounds that conventional standard-basis fast algorithms do. Additionally, it was not clear whether Bini and Lotti's [32] speed-stability trade-off for fast matrix multiplication algorithms applies to alternative basis algorithms as well. Furthermore, it was unknown whether existing stability improvement methods such as diagonal scaling [34], [39]–[42] and non-uniform approaches such as those of Castrapel and Gustafson [35] and D'alberto [36] can be applied to alternative basis algorithms.

D. Our Contribution

1) Error bounds. We provide the first error bound for alternative basis algorithms and show that they obtain the asymptotically identical error bounds as their standard basis counterparts, with a small increase in the multiplicative factor of the error bound. We prove that this applies to all known versions of the alternative basis methods, including those that maintain matrix dimensions [10]–[12], and those that compute over spaces of higher dimensions [13]. To this end, we prove the following theorem:

Theorem I.1. *Let $\mathbf{A},\mathbf{B} \in \mathbb{R}^{N\times N}$ and let $\mathbf{C}=\mathbf{A}\cdot\mathbf{B}$. Consider an alternative basis matrix multiplication algorithm ALG with $N_0 \times N_0$ base case. Consider a run of ALG with $L=\log_{N_0} N$ recursive steps. Let $\hat{\mathbf{C}} = ALG(\mathbf{A},\mathbf{B})$. Then*

$$\left\|\hat{\mathbf{C}}-\mathbf{C}\right\| \leq f_{ALG}(N)\,\|A\|\,\|B\|\,\epsilon + O\left(\epsilon^2\right), \qquad (1)$$

where $f_{ALG}(N) = \left(1+Q\cdot\log_{N_0} N\right)\cdot N^{\log_{N_0} E}$, E is the stability factor (see Definition III.2), Q is the prefactor (see Definition III.4), and $\|\cdot\|$ is the max-norm.

The theorem generalizes to rectangular matrices and the common non-stationary case, where classical matrix multiplication is used in the base case of the recursion (see Section III for details). Due to space limitations, we provide a shorter proof with a slightly looser multiplicative factor Q' (see Definition III.5). The tighter bound and its generalization to fully decomposed algorithms of [13] will appear in the full version of this paper.

TABLE I: Arithmetic costs and error bounds of $\langle 2,2,2;7\rangle$-algorithms. n is the matrix dimension. For large input matrices, the dominant term in the error bound is the stability factor, and the dominant term in the arithmetic cost is its leading coefficient. The boldface values are the best in their category. Our algorithm is the only one that achieves both the optimal leading coefficient of 5 and the optimal stability factor of 12.

Algorithm	Arithmetic cost	Error bound
Strassen [1]	$7n^{\log_2 7}-6n^2$	$(1+8\log_2 n)n^{\mathbf{\log_2 12}}$
Winograd [24]	$6n^{\log_2 7}-5n^2$	$(1+10\log_2 n)n^{\log_2 18}$
Karstadt & Schwartz [11]	$\mathbf{5}n^{\log_2 7}-4n^2+3n^2\log_2 n$	$(1+16\log_2 n)n^{\log_2 18}$
Schwartz & Vaknin [48]	$\mathbf{5}n^{\log_2 7}-4n^2+\frac{3}{2}n^2\log_2 n$	$(1+16\log_2 n)n^{\log_2 18}$
Ours	$\mathbf{5}n^{\log_2 7}-4n^2+\frac{9}{4}n^2\log_2 n$	$(1+15\log_2 n)n^{\mathbf{\log_2 12}}$

2) Beyond the Bini and Lotti trade-off. We obtain a fast matrix multiplication algorithm with 2×2 base case and coefficients in $\mathbb{H}=\left\{0,\pm 2^i : i\in\mathbb{N}\right\}$ with the arithmetic cost leading coefficient of 5 (optimal for algorithms with 2×2 base case, see lower bound in [11]) and stability factor of 12 (best known for algorithms with 2×2 base case). This seemingly contradicts Bini and Lotti's [32] classification of all 2×2 algorithms with coefficients in $\mathbb{H}$, as they exhibit a trade-off between the stability factor and the arithmetic costs leading coefficient (see Table I). To this end, we show how to decouple arithmetic cost and error bound using the alternative basis method, allowing for simultaneous optimization of both. Our method may increase the arithmetic cost but only by lower-order terms, which may be a worthwhile cost for improving the error bound. Furthermore, we demonstrate arithmetic costs and error bounds for a selection of existing and new alternative basis algorithms, with improved arithmetic costs and maintained asymptotic error bounds in Table II.

3) Diagonal Scaling. We show that diagonal scaling techniques are as effective for alternative basis algorithms as for standard basis fast matrix multiplication algorithms. In contrast, we show that Castrapel and Gustafson's [35] and D'Alberto's [36] methods do not readily extend to alternative basis algorithms in the full version of this paper.

4) Parallization, performance, and stability benchmarks. We provide a suite of experimental results that go in line with our theoretical predictions. Particularly, we provide a high-performance parallel implementation of our $\langle 2,2,2;7\rangle$-algorithm. We also provide memory hierarchy and inter-process communication cost analysis for our algorithm. Our algorithm is shown to be on par with the best-in-class for both runtime and numerical error. We provide performance benchmarks for both the alternative methods in [10]–[12] and the more general method in [13], all align with the theoretical predictions. Finally, our benchmarks demonstrate that diagonal scaling is an effective tool for improving the stability of alternative basis algorithms.

E. Paper Organization

We provide preliminaries, notations, and algorithms review in Section II. In Section III, we provide the error analysis for the alternative basis method. Section IV presents our method for simultaneously optimizing speed and stability. In Section V, we discuss combining the alternative basis with stability improvement methods such as diagonal scaling. We provide experimental results in Section VI and conclude in Section VII.

II. Preliminaries

In this section, we provide preliminaries regarding recursive bilinear algorithms, alternative basis matrix multiplication, and rounded arithmetic. We denote matrices and vectors in bold lettering, e.g., $\mathbf{A}$, for clarity, and we denote matrix elements in lowercase, e.g., a_{ij}.

A. Recursive Bilinear Algorithms

A recursive bilinear $\langle M_0,K_0,N_0;R\rangle$-algorithm multiplies two matrices $\mathbf{A}\in\mathbb{R}^{M\times K}$ and $\mathbf{B}\in\mathbb{R}^{K\times N}$ by partitioning $\mathbf{A}$ into $M_0\times K_0$ sub-matrices of size $(M/M_0)\times(K/K_0)$ and $\mathbf{B}$ into $K_0\times N_0$ sub-matrices of size $(K/K_0)\times(N/N_0)$, and then applying the following to the submatrix blocks:

$$\begin{aligned}\mathbf{S}_r=\sum\nolimits_{i=1}^{M_0K_0}u_{ir}\mathbf{A}_i,\quad \mathbf{T}_r=\sum\nolimits_{j=1}^{K_0N_0}v_{jr}\mathbf{B}_j,\\ \mathbf{M}_r=\mathbf{S}_r\cdot\mathbf{T}_r,\quad \mathbf{C}_k=\sum\nolimits_{r=1}^{R}w_{kr}\mathbf{M}_r,\end{aligned}\tag{2}$$

for $k=1,\ldots,M_0N_0$, where $(\cdot)$ signifies a recursive call to the algorithm, and u_{ir},v_{jr},w_{kr} are the elements of an $M_0K_0\times R$ matrix $\mathbf{U}$, a $K_0N_0\times R$ matrix $\mathbf{V}$, and an $M_0N_0\times R$ matrix $\mathbf{W}$. We use single subscripts on matrices as an index for the major ordering of matrix block columns or rows. We assume that M,K, and N are powers of M_0, K_0, and N_0; otherwise, we can pad the matrices with zeros, and the same analysis will hold. We do not assume that the algorithm recurses to a base case of dimension 1 but do assume that it performs L recursive levels and then invokes the classical matrix multiplication algorithm as the base case. Therefore such an algorithm is defined by the set of coefficients $\langle\mathbf{U},\mathbf{V},\mathbf{W}\rangle$, which we call the encoding/decoding matrices, and the number of recursion steps, L. We generally denote a recursive bilinear algorithm that multiplies an $M_0\times K_0$ matrix with an $K_0\times N_0$ matrix with R multiplications as an $\langle M_0,K_0,N_0;R\rangle$-algorithm.

B. Alternative Basis Matrix Multiplication Algorithms

The number of linear operations incurred by an $\langle M_0,K_0,N_0;R\rangle$-algorithm can be reduced by transforming the basis of the input and output matrices to one in which the corresponding encoding/decoding matrices are sparser [10], [11], [13]. We follow the notations of Beniamini and Schwartz [13] that generalize the notations of [10], [11] and allow the decomposition to be in a different dimension than that of the original input matrices.

Definition II.1. [13] Let $D_1,D_2\in\mathbb{N}$, and let $\varphi^1:\mathbb{R}^{D_1}\to\mathbb{R}^{D_2}$ be a linear transformation. Let $L\in\mathbb{N}$. Let $\mathbf{v}\in\mathbb{R}^{D_1^L}$, and denote $\forall i\in[D_1]:\mathbf{v}^i=\left(\mathbf{v}_{D_1^{L-1}\cdot(i-1)+1},\ldots,\mathbf{v}_{D_1^{L-1}\cdot i}\right)$.

TABLE II: A sample of fast matrix multiplication algorithms and their alternative basis versions. The number of linear operations determines the leading coefficient of their arithmetic cost. Both the algorithm and its alternative basis version have the same stability factor. We note that [12] provides an alternative basis version of Strassen's algorithm that obtains the same stability bounds and arithmetic costs as our algorithm.

	Number of Additions		**Leading Coefficient**		**Error Bound**	
Algorithm	Original	Alt. Base	Original	Alt. Base	Original	Alt. Base
$\langle 2,2,2;7\rangle$ [12]	7	5	18	12	$(1+15\log_2 n)\,n^{\log_2 12}$	$(1+15\log_2 n)\,n^{\log_2 12}$
$\langle 3,2,3;15\rangle$ [39]	55	39	8.28	6.17	$(1+10\log_2 n)\,n^{\log_2 20}$	$(1+19\log_2 n)\,n^{\log_2 20}$
$\langle 3,3,3;23\rangle$ [22]	84	68	7	5.86	$(1+13\log_3 n)\,n^{\log_3 31}$	$(1+19\log_3 n)\,n^{\log_3 31}$
$\langle 4,4,2;23\rangle$ [39]	178	100	12.8	7.56	$(1+26\log_4 n)\,n^{\log_4 102}$	$(1+35\log_4 n)\,n^{\log_4 102}$
$\langle 3,4,5;47\rangle$ [54]	298	238	10.52	8.55	$(1+21\log_4 n)\,n^{\log_4 112}$	$(1+31\log_4 n)\,n^{\log_4 112}$

The linear mapping $\varphi^L : \mathbb{R}^{D_1^L} \to \mathbb{R}^{D_2^L}$ is defined recursively as follows:

$$\varphi^L(\mathbf{v}) = \varphi^1\left(\varphi^{L-1}\left(\mathbf{v}^1\right), \ldots, \varphi^{L-1}\left(\mathbf{v}^{D_1}\right)\right).$$

We utilize the $D_1 \times D_2$ matrix representation[2] of φ, which we also denote by φ, and which maps $\mathbf{v}$ recursively by

$$\forall j \in [D_2]: \quad \varphi^L(\mathbf{v})_j = \sum_{i=1}^{D_1} \varphi_{ij} \cdot \varphi^{L-1}\left(\mathbf{v}^i\right), \tag{3}$$

where φ^0 is the identity function. We drop the superscript L and write $\varphi(\mathbf{v})$ when evident from the context.

Definition II.2 (Alternative Basis Matrix Multiplication [13])**.** Let $\langle \mathbf{U}, \mathbf{V}, \mathbf{W}\rangle$ be the encoding/decoding matrices of a recursive bilinear algorithm. Let

$$\mathbf{U} = \phi \cdot \mathbf{U}_\phi, \mathbf{V} = \psi \cdot \mathbf{V}_\psi, \mathbf{W} = \nu \cdot \mathbf{W}_\nu \tag{4}$$

be decompositions of the aforementioned matrices[2], such that $\phi \in \mathbb{R}^{M_0K_0 \times D_U}$, $\psi \in \mathbb{R}^{K_0N_0 \times D_V}$, $\nu \in \mathbb{R}^{M_0N_0 \times D_W}$, and $\mathbf{U}_\phi \in \mathbb{R}^{D_U \times R}$, $\mathbf{V}_\psi \in \mathbb{R}^{D_V \times R}$, $\mathbf{W}_\nu \in \mathbb{R}^{D_W \times R}$ where D_U, D_V, D_W are the dimensions of each alternative basis and may be larger than the original dimensions of the matrices. Let $ALG_{\langle\phi,\psi,\nu\rangle}$ be a recursive-bilinear algorithm defined by the encoding and decoding matrices $\mathbf{U}_\phi, \mathbf{V}_\psi, \mathbf{W}_\nu$. The Alternative Basis Matrix Multiplication Algorithm is defined as follows by Algorithm 1.

Algorithm 1 Alternative Basis Matrix Multiplication

Input: $\mathbf{A} \in \mathbb{R}^{M \times K}, \mathbf{B} \in \mathbb{R}^{K \times N}$
Output: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$

1: **function** $ALG_{ABMM}(\mathbf{A}, \mathbf{B})$
2: $\quad \tilde{\mathbf{A}} = \phi(\mathbf{A})$ ▷ *Transform the first input*
3: $\quad \tilde{\mathbf{B}} = \psi(\mathbf{B})$ ▷ *Transform the second input*
4: $\quad \tilde{\mathbf{C}} = ALG_{\langle\phi,\psi,\nu\rangle}\left(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}\right)$ ▷ *Recursive-bilinear phase*
5: $\quad \mathbf{C} = \nu^T(\tilde{\mathbf{C}})$ ▷ *Transform the output*
6: $\quad$ **return** $\mathbf{C}$

[2]Our notation is transposed to that of [10], [11], [13] to match that of [32], [38], [39].

Claim II.3. [55], [56] Given a fast matrix multiplication algorithm represented by the encoding/decoding matrices $\langle \mathbf{U}, \mathbf{V}, \mathbf{W}\rangle$, and any non-singular matrices $\mathbf{Q}_1 \in \mathbb{R}^{K_0 \times K_0}$, $\mathbf{Q}_2 \in \mathbb{R}^{M_0 \times M_0}$, and $\mathbf{Q}_3 \in \mathbb{R}^{N_0 \times N_0}$, it holds that

$$\left\langle \left(\mathbf{Q}_1 \otimes \mathbf{Q}_2^{-T}\right)\mathbf{U}, \left(\mathbf{Q}_1^{-T} \otimes \mathbf{Q}_3\right)\mathbf{V}, \left(\mathbf{Q}_2 \otimes \mathbf{Q}_3{}^{-T}\right)\mathbf{W} \right\rangle \tag{5}$$

are also encoding/decoding matrices of a fast matrix multiplication algorithm.

C. Arithmetic Model and Notation

We adopt the classical model of rounded arithmetic, used by Demmel et al. and Ballard et al. [38], [39]. In this model, each arithmetic operation incurs a small relative error denoted ϵ that is bounded by the machine precision. That is, for each arithmetic operation $\mathrm{op}(a,b)$, the computed value is $\mathrm{op}(a,b)(1+\theta)$, with some arbitrary θ such that $|\theta| < \epsilon$. The classical arithmetic operations are $\{+,-,\cdot\}$. We denote computed values by the conventional hat $\hat{\cdot}$ or $fl(\cdot)$ notation, for example, a computed operation $\mathrm{op}(a,b)$ is $fl(\mathrm{op}(a,b))$ and a computation of a matrix block $\mathbf{C}$ is $\hat{\mathbf{C}}$.

For simplicity, let $\Theta = \{\theta \mid |\theta| < \epsilon\}$ be the set of all errors bounded by the machine precision ϵ, and let $\Delta = \{1+\theta \mid \theta \in \Theta\}$. We use standard set operation notation: for sets A, B and operator $\mathrm{op} \in \{+,-,\cdot\}$, we have $A \,\mathrm{op}\, B := \{a \,\mathrm{op}\, b \mid a \in A, b \in B\}$. We use the notation $A^j = \overbrace{A \cdot A \cdots A}^{j \text{ terms}}$ and note that $\Delta^j \subseteq \Delta^{j+1}$ as $1 \in \Delta$. This allows the useful notation of $fl(\mathrm{op}(a,b)) \in \mathrm{op}(a,b)\Delta$.

Under this arithmetic model, we use the following general summation fact:

$$fl\left(\textstyle\sum_{i=1}^N fl(c_i \cdot a_i)\right) \in \left(\textstyle\sum_{i=1}^N c_i \cdot a_i\right)\Delta^N, \tag{6}$$

where the summing algorithm sequentially accumulates a_i terms. By using a serialized divide-and-conquer summation, we obtain

$$fl\left(\textstyle\sum_{i=1}^N fl(c_i \cdot a_i)\right) \in \left(\textstyle\sum_{i=1}^N c_i \cdot a_i\right)\Delta^{1+\lceil\log_2 N\rceil}. \tag{7}$$

We use the looser bound in Equation (6) for simplicity. Our results may be adjusted for the tighter error bounds in

Equation (7). We also use the following property:

$$fl\left(\sum_{i=1}^{N} c_i \Delta^{a_i}\right) \in \left(\sum_{i=1}^{N} c_i\right) \Delta^{N+\max_i a_i}. \quad (8)$$

III. Error Analysis of Alternative Basis Matrix Multiplication Algorithms

In this section, we provide stability-related definitions and analyze the error bound for alternative basis matrix multiplication algorithms. We compare it to the bound of standard basis fast matrix multiplication algorithms (i.e., the bounds in [38], [39]). Finally, we discuss the implications of the bound.

A. Principal Quantities of Fast Matrix Multiplication Error

The primary quantifier for the error of a fast matrix multiplication algorithm is the stability vector [32], [38], [39]. When bounding the numerical error, the maximal entry in the stability vector translates into the exponent of the numerical error bound.

Definition III.1 (Stability vector of fast matrix multiplication [32], [38], [39])**.** Let $\langle \mathbf{U}, \mathbf{V}, \mathbf{W} \rangle$ be the encoding/decoding matrices of a fast matrix multiplication algorithm. Let

$$\forall r \in [R]: \; a_r := \sum_{i=1}^{M_0 K_0} |u_{ir}| \quad b_r := \sum_{j=1}^{K_0 N_0} |v_{jr}|. \quad (9)$$

Then the stability vector $\mathbf{e} \in \mathbb{R}^{M_0 N_0}$ is defined entry-wise by $e_k := \sum_{r=1}^{R} a_r \cdot b_r \cdot |w_{kr}|$. The *stability factor* is $E := \max_k e_k$.

We next generalize Definition III.1 for the case of an alternative basis algorithm.

Definition III.2 (Stability vector for alternative basis)**.** Let $\langle \mathbf{U}_\phi, \mathbf{V}_\psi, \mathbf{W}_\nu \rangle$, ϕ, ψ, ν be the encoding/decoding matrices and basis transformation matrices of an alternative basis algorithm. The stability vector $\mathbf{e} \in \mathbb{R}^{M_0 N_0}$ is defined by the standard basis representation of the algorithm, that is, by the stabiliy vector of $\langle \phi \cdot \mathbf{U}_\phi, \psi \cdot \mathbf{V}_\psi, \nu \cdot \mathbf{W}_\nu \rangle$, and the *stability factor* is $E := \max_k e_k$.

Ballard et al. [39] defined the prefactor vector $\mathbf{q} \in \mathbb{R}^{M_0 N_0}$ of a fast matrix multiplication algorithm, which captures how many arithmetic operations are performed on the way to compute $\mathbf{C}_{ij}$. The more operations that are performed, the higher the entry in the prefactor vector. The maximal entry in the prefactor vector determines the constant factor of the error bound. Formally:

Definition III.3 (Prefactor vector of a recursive bilinear algorithm [39])**.** Let $\langle \mathbf{U}_\phi, \mathbf{V}_\psi, \mathbf{W}_\nu \rangle$ be the encoding/decoding matrices of a recursive bilinear algorithm. Define

$$\alpha_r := \sum_{i=1}^{D_U} \mathbb{I}\left(u_{ir}^{\phi}\right), \; \beta_r := \sum_{j=1}^{D_V} \mathbb{I}\left(v_{jr}^{\psi}\right), \; \gamma_k := \sum_{r=1}^{R} \mathbb{I}\left(w_{kr}^{\nu}\right),$$

where $\mathbb{I}$ is the indicator function, that is, $\mathbb{I}(x) = 1$ if $x \neq 0$ and $\mathbb{I}(x) = 0$ otherwise. The prefactor vector $\mathbf{q} \in \mathbb{R}^{M_0 N_0}$ is defined entry-wise by $q_k = \gamma_k + \max_r (\alpha_r + \beta_r) \mathbb{I}(w_{kr}^{\nu})$, and the *prefactor* is $Q_B := \max_k q_k$.

We next generalize Definition III.3 for the case of an alternative basis algorithm.

Definition III.4 (Prefactor vector for alternative basis)**.** Let ALG be an alternative basis algorithm as in Definition II.2. For a basis transformation $\varphi : \mathbb{R}^{D_1} \to \mathbb{R}^{D_2}$, define $\mathbf{q}^{\varphi} \in \mathbb{R}^{D_2}$ element-wise as $q_j^{\varphi} = \sum_{i=1}^{D_1} \mathbb{I}(\varphi_{ij})$, and denote $Q^{\varphi} := \max q_j^{\varphi}$. Define the following vectors:

$$q_j^{\phi} = \sum_{i=1}^{M_0 K_0} \mathbb{I}(\phi_{ij}), q_j^{\psi} = \sum_{i=1}^{K_0 N_0} \mathbb{I}(\psi_{ij}), q_i^{\nu} := \sum_{j=1}^{D_W} \mathbb{I}(\nu_{ij}). \quad (10)$$

Note that we use the notation $\mathbf{q}^{\nu}$ to describe $\mathbf{q}^{\nu^T}$ for simplicity. Also denote:

$$y_r = \alpha_r + \max_i \left(q_i^{\phi} \cdot \mathbb{I}\left(u_{ir}^{\phi}\right)\right), z_r = \beta_r + \max_i \left(q_i^{\psi} \cdot \mathbb{I}\left(v_{ir}^{\psi}\right)\right).$$

The prefactor vector $\mathbf{q} \in \mathbb{R}^{M_0 N_0}$ is defined entry-wise by:

$$q_j = q_j^{\nu} + \max_k \left(\gamma_k + \max_r (y_r + z_r) \mathbb{I}(w_{kr}^{\nu})\right) \mathbb{I}(\nu_{kj}).$$

The *prefactor* is $Q := \max_j q_j$.

We also define a slightly looser prefactor:

Definition III.5. For an alternative basis algorithm as in Definition II.2 and $\mathbf{q}^{\phi}, \mathbf{q}^{\psi}, \mathbf{q}^{\nu}$ as in Equation (10). Define the prefactor Q' as $Q' = Q_B + Q^{\phi} + Q^{\psi} + Q^{\nu}$, where Q_B is the prefactor for the bilinear phase (see Definition III.3).

Remark III.6. Q' is looser than Q, that is, $Q \leq Q'$.

B. Error Analysis of Alternative Basis Algorithms

In this section, we provide the error bound for the alternative basis matrix algorithm. Ballard et al. [39] provided the following bound for the standard basis fast matrix multiplication algorithm:

Theorem III.7. *[39] Let $\mathbf{A} \in \mathbb{R}^{M \times K}$ and $\mathbf{B} \in \mathbb{R}^{K \times N}$, and let $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$. Consider the fast matrix multiplication algorithm ALG in Equation* (2) *with L recursive steps and with the classical algorithm used to multiply the matrices at the base cases of the recursion. Let $\hat{\mathbf{C}} = ALG(\mathbf{A}, \mathbf{B})$. Then*

$$\left\| \hat{\mathbf{C}} - \mathbf{C} \right\| \leq f_{ALG}(K, L) \|\mathbf{A}\| \|\mathbf{B}\| \epsilon + O\left(\epsilon^2\right),$$

where $f_{ALG}(K, L) = \left(K/K_0^L + Q_B L\right)\left(K/K_0^L\right) E^L$, E and Q_B are the stability factor and prefactor, respectively, and $\|\cdot\|$ is the max-norm.

We next generalize this bound to alternative basis algorithms. However, due to space limitations, we provide here a slightly weaker bound which is shorter to prove. The proof of Theorem I.1 appears in an upcoming journal version.

Theorem III.8. *Let $\mathbf{A} \in \mathbb{R}^{M \times K}$ and $\mathbf{B} \in \mathbb{R}^{K \times N}$ and let $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$. Consider an alternative basis matrix multiplication algorithm ALG with L recursive steps and with the classical algorithm used to multiply the matrices at the base cases of the recursion. Let $\hat{\mathbf{C}} = ALG(\mathbf{A}, \mathbf{B})$. Then*

$$\left\|\hat{\mathbf{C}}-\mathbf{C}\right\| \leq f_{ALG}\left(K,L\right)\left\|\mathbf{A}\right\|\left\|\mathbf{B}\right\|\epsilon+O\left(\epsilon^{2}\right),$$

where $f_{ALG}\left(K,L\right)=\left(K/K_0^L+Q'L\right)\left(K/K_0^L\right)E^L$*,* E *is the stabiliy factor and* Q' *is the prefactor.*

Theorem I.1 provides a similar bound with the tighter prefactor (see Definition III.4), that is with $f_{ALG}\left(K,L\right)=\left(K/K_0^L+QL\right)\left(K/K_0^L\right)E^L$. When the transformation matrices ϕ,ψ,ν are the identity matrix, the algorithm identifies with traditional fast matrix multiplication, and our definition of Q' almost identifies with that of [39]. Since our proof assumes that round-off errors are introduced by every execution of any arithmetic operation, particularly by the basis transformations, whereas [39] analyzes only a recursive bilinear algorithm, it increases the prefactor by at least 1 for each basis transformation, bringing our prefactor to a slightly higher value of Q_B+3. The original prefactor could be easily matched in our analysis by extending it to explicitly account for multiplications by elements in $\{\pm1\}$, which do not introduce errors.

The proof of Theorem III.8 follows that of [39] with the following changes: We first analyze the accumulated relative errors Δ^x of the initial basis transformations (steps 2,3 in Algorithm 1). We then analyze the bilinear phase (step 4 in Algorithm 1), similar to the proof in [39]. Finally, we analyze the accumulated error of the last basis transformation (step 5 in Algorithm 1), and then bound the total error.

We show that although the basis transformations increase the accumulated error Δ, they do not alter the key characteristics of the algorithm's stability compared to its standard basis counterpart. Formally, we obtain the following corollary:

Corollary III.9. *The error bound of an alternative basis matrix multiplication algorithm has the same exponent as that of its standard basis counterpart.*

Proof. The alternative basis algorithm has the same stability vector as its standard basis counterpart (recall Definition III.2). In particular, they also have the same stablity factor E. By Theorem III.8 and Theorem III.7, the exponent of the error bound is determined by the stability factor E for both algorithms, which is identical. □

We analyze the error of basis transformations.

Claim III.10 (Error of basis transformations). Let φ be a recursive linear transformation (recall Definition II.1) with input $\mathbf{A}$. Then $\hat{\varphi}^L\left(\mathbf{A}\right)\in\varphi^L\left(\mathbf{A}\right)\Delta^{Q^\varphi L}$.

Proof. We prove the claim by induction. The computation of $\hat{\varphi}\left(\mathbf{A}\right)_j$ at the base case (recall Equation (3)) sums exactly q_j^φ elements. By Equation (6), the error is

$$\forall j\in[D_2]:\quad \hat{\varphi}\left(\mathbf{A}\right)_j\in\varphi\left(\mathbf{A}\right)\Delta^{q_j^\varphi}\in\varphi\left(\mathbf{A}\right)\Delta^{Q^\varphi}. \tag{11}$$

Next, assume that the claim holds for $1\leq l\leq L-1$. By Equation (3), the computation of $\hat{\varphi}^L\left(\mathbf{A}\right)_j$ sums q_j^φ elements of $\hat{\varphi}^{L-1}\left(\mathbf{A}\right)$. By Equation (8) and the induction hypothesis, the accumulated error is

$$\forall j\in[D_2]:\ \hat{\varphi}^L\left(\mathbf{A}\right)_j\in\sum_{i=1}^{D_1}\varphi_{ij}\cdot\varphi^{L-1}\left(\mathbf{A}_i\right)\Delta^{q_j^\varphi+Q^\varphi(L-1)}.$$

Since $q_j^\varphi\leq Q^\varphi$, we obtain the desired result. □

We further prove a claim for the case in which the input has some error.

Claim III.11. Let φ be a recursive linear transformation (recall Definition II.2). Assume the input matrix $\mathbf{A}$ is given with some error Δ^X. Then $\hat{\varphi}\left(\mathbf{A}\right)\in\hat{\varphi}\left(\mathbf{A}\right)\Delta^{Q^\varphi L+X}$.

Proof. We follow the proof of Claim III.10, with the following changes: In the bottom level of the recursion, each involved block has error Δ^X. By Equation (8), the summation in Equation (11) results in an error increased by exactly Δ^X. Applying Equation (8) in every layer of recursion results in a total error increase of exactly Δ^X. □

Theorem III.7 provides an absolute error bound for matrix multiplication algorithms. We next restate it as a claim that analyzes the accumulated error Δ of any recursive bilinear algorithm (i.e Equation (2)) and generalize it, allowing the input matrices of the bilinear phase to be given with errors.

Claim III.12. Let ALG be a recursive bilinear algorithm with encoding/decoding matrices $\langle\mathbf{U}_\phi,\mathbf{V}_\psi,\mathbf{W}_\nu\rangle$ and input matrices $\mathbf{A},\mathbf{B}$ with errors Δ^X,Δ^Y repectively. Then the computation of ALG accumulates the following error:

$$fl\left(ALG\right)\left(\mathbf{A},\mathbf{B}\right)_k\in ALG\left(\mathbf{A},\mathbf{B}\right)_k\Delta^{K/K_0^L+Q_BL+X+Y}.$$

Here Q_B is the prefactor of the recursive bilinear algorithm (i.e Definition III.3).

Proof. [39] provides a proof for Theorem III.7 that does not make any assumption about $\langle\mathbf{U}_\phi,\mathbf{V}_\psi,\mathbf{W}_\nu\rangle$ satisfying the triple product condition. It, therefore, applies to any recursive bilinear algorithm. We next account for the given error. By Equation (2), the algorithm starts by recursively forming the $\mathbf{S}_r$ (and $\mathbf{T}_r$) matrices. In the first level of recursion, each block of $\mathbf{A}$ is given with Δ^X error, which by Equation (8), increases the error of the $\mathbf{S}_r$ matrices by the same factor. Repeatedly applying Equation (8) for the summation of the $\mathbf{S}_r$ matrices in each level of recursion, the additional factor of Δ^X propagates through the recursion to an additional factor of Δ^X for the $\mathbf{S}_r$ matrices at the bottom level. The same holds for $\mathbf{T}_r$ and Δ^Y. Using classical multiplication in the bottom level to multiply each $\mathbf{S}_r$ and $\mathbf{T}_r$ into $\mathbf{M}_r$ joins their respective accumulated errors with an additional factor of Δ^{X+Y}. Finally, the $\mathbf{C}_r$ matrices are recursively formed by summation of the $\mathbf{M}_r$ matrices. Applying Equation (8) in each level, the additional Δ^{X+Y} propagates through each recursion level, increasing the accumulated error of each block $\mathbf{C}_k$ by that factor. At the end of the recursion, the accumulated error adds up to the original analysis, Δ^{LQ_B+K/K_0^L}, with an additional Δ^{X+Y}. □

We next detail the correctness claim of the alternative basis algorithm provided by [10], [11] and generalized by [13], which links the exact arithmetic of the alternative basis algorithm to that of the standard basis.

Claim III.13 (Computation of alternative basis matrix multiplication [10], [11], [13]). Let ALG be an alternative basis algorithm. For a linear operator $\mathbf{M}$, denote $\mathbf{M}^L := \otimes_L \mathbf{M}$ where $\otimes$ is the $\otimes_L$ is the L-th tensor product. Assuming exact arithmetic, the computation of ALG is same as for the standard basis algorithm, meaning

$$ALG(\mathbf{A},\mathbf{B}) = \left(\left(\mathbf{A}\mathbf{U}^L\right) \odot \left(\mathbf{B}\mathbf{V}^L\right)\right)\left(\mathbf{W}^T\right)^L,$$

where $\odot$ is element-wise multiplication. This means that for every $k \in \left[M_0^L N_0^L\right]$, the result block is:

$$\mathbf{C}_k = \sum_{r=1}^{R^L} w_{kr}^L \left(\left(\sum_{i=1}^{M_0^L K_0^L} u_{ir}^L \mathbf{A}_i\right) \cdot \left(\sum_{j=1}^{K_0^L N_0^L} v_{jr}^L \mathbf{B}_j\right)\right).$$

We now have the tools to prove Theorem III.8.

Proof of Theorem III.8. We follow the accumulated error with the steps of the algorithm. By Claim III.10, the accumulated error of steps 2,3 (basis transformations) is

$$\hat{\phi}(\mathbf{A}) \in \phi(\mathbf{A})\,\Delta^{LQ^\phi}, \quad \hat{\psi}(\mathbf{B}) \in \psi(\mathbf{B})\,\Delta^{LQ^\psi}.$$

By Claim III.12, the accumulated error of step 4 (bilinear phase) is $\hat{\tilde{\mathbf{C}}} \in \tilde{\mathbf{C}} \cdot \Delta^{K/K_0^L + Q_B L + Q^\phi L + Q^\psi L}$.

Next, by Claim III.11, the accumulated error of step 5 (final basis transformation) is $\hat{\mathbf{C}} \in \mathbf{C} \cdot \Delta^\delta$, where $\delta := K/K_0^L + \left(Q_B + Q^\phi + Q^\psi + Q^{\nu^T}\right) L$.

We next compute the forward error bound. To determine the largest computed block, we apply Claim III.13 and bound the size of the exact computation:

$$\begin{aligned}
|\mathbf{C}_k| &\le \sum_{r=1}^{R^L} \left| w_{kr}^L \left(\left(\sum_{i=1}^{M_0^L K_0^L} u_{ir}^L \mathbf{A}_i\right) \cdot \left(\sum_{j=1}^{K_0^L N_0^L} v_{jr}^L \mathbf{B}_j\right)\right)\right| \\
&\le \sum_{r=1}^{R^L} \left|w_{kr}^L\right| \cdot \sum_{i=1}^{M_0^L K_0^L} \left|u_{ir}^L\right| |\mathbf{A}_i| \cdot \sum_{j=1}^{K_0^L N_0^L} \left|v_{jr}^L\right| |\mathbf{B}_j| \\
&\le \sum_{r=1}^{R^L} \left|w_{kr}^L\right| \sum_{i=1}^{M_0^L K_0^L} \left|u_{ir}^L\right| \sum_{j=1}^{K_0^L N_0^L} \left|v_{jr}^L\right| \cdot \left(K/K_0^L\right) \|\mathbf{A}\| \|\mathbf{B}\|.
\end{aligned}$$

To compute the maximum over all blocks $\mathbf{C}_k$, denote $\xi_k = \sum_{r=1}^{R^L} \left|w_{kr}^L\right| \sum_{i=1}^{M_0^L K_0^L} \left|u_{ir}^L\right| \sum_{j=1}^{K_0^L N_0^L} \left|v_{jr}^L\right|$, and compute the maximum over all k:

$$\begin{aligned}
\max \xi_k &= \max \sum_{r=1}^{R^L} \left|w_{kr}^L\right| \cdot \sum_{i=1}^{M_0^L K_0^L} \left|u_{ir}^L\right| \cdot \sum_{j=1}^{K_0^L N_0^L} \left|v_{jr}^L\right| \\
&= \left(\max \sum_{r=1}^{R} |w_{kr}| \cdot \sum_{i=1}^{M_0 K_0} |u_{ir}| \cdot \sum_{j=1}^{K_0 N_0} |v_{jr}|\right)^L = E^L,
\end{aligned}$$

where E is given in Definition III.2. Finally, we have $\mathbf{E}_k = \hat{\mathbf{C}}_k - \mathbf{C}_k \in \mathbf{C}_k \Theta^\delta$. Therefore, the total error is

$$|\mathbf{E}_k| \le |\mathbf{C}_k| \epsilon^\delta + O\left(\epsilon^2\right) \le E^L \left(K/K_0^L\right) \|\mathbf{A}\| \|\mathbf{B}\| \delta\epsilon + O\left(\epsilon^2\right).$$

Plugging in δ, the theorem follows. □

IV. Fast and Stable Algorithms

In this section, we present two approaches for finding algorithms that attain both speed and stability. Roughly speaking, the first approach is to stabilize an existing fast algorithm, and the second approach is to speed up a stable one.

A. Stabilizing Alternative Basis Algorithms

Given an alternative basis algorithm, one can traverse between stability classes without adding costs to its bilinear phase. Formally:

Claim IV.1. Let ALG be an alternative basis algorithm. Let $\mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_3$ be invertible matrices as in Claim II.3. Define ALG' to be the algorithm with the same bilinear phase as ALG and the following basis transformations:

$$\left(\mathbf{Q}_1 \otimes \mathbf{Q}_2^{-T}\right)\phi, \left(\mathbf{Q}_1^{-T} \otimes \mathbf{Q}_3\right)\psi, \left(\mathbf{Q}_2 \otimes \mathbf{Q}_3{}^{-T}\right)\nu. \quad (12)$$

Then ALG' is an alternative basis algorithm with the same bilinear phase as ALG.

Proof. Denote the encoding/decoding matrices of ALG' in the standard basis by $\langle \mathbf{U}', \mathbf{V}', \mathbf{W}' \rangle$. We have

$$\mathbf{U}' = \left(\mathbf{Q}_1 \otimes \mathbf{Q}_2^{-T} \cdot \phi\right) \cdot \mathbf{U}_\phi = \left(\mathbf{Q}_1 \otimes \mathbf{Q}_2^{-T}\right) \mathbf{U}, \quad (13)$$

and similarly for $\mathbf{V}', \mathbf{W}'$. By Claim II.3, $\langle \mathbf{U}', \mathbf{V}', \mathbf{W}' \rangle$ are encoding/decoding matrices of a fast matrix multiplication algorithm. □

Corollary IV.2. *The stability factor of ALG' is determined by $\langle \mathbf{U}', \mathbf{V}', \mathbf{W}' \rangle$.*

Proof. By Definition III.2, the stability factor of ALG' is determined by its standard basis representation. □

Corollary IV.3. *ALG' has the same leading coefficient of its arithmetic and communication costs (see Definition A.1) as ALG.*

Proof. The leading coefficient for both the arithmetic and communication costs is determined by the bilinear phase, which is identical for ALG and ALG'. □

We apply the above to Schwartz and Vaknin's [48] high-performance $\langle 2,2,2;7 \rangle$-algorithm. That is, we modify its basis transformation matrices such that we obtain an optimal stability factor while maintaining its performance. Note that the basis transformations have more operations than [48], causing a slight increase to the arithmetic and communication costs, but only in lower order monomials (see Table III). We also maintain the same memory footprint since the new basis transformations are computed in place. We provide a complete communication costs analysis in Appendix A.

TABLE III: Communication costs of $\langle 2,2,2;7\rangle$-algorithms. n is the matrix dimension and M is the local memory size. Our algorithm, up to low order monomials, obtains the same communication costs as [48] and has improved stability (see Table I).

ALG	Memory footprint	Communication costs
[1]	$\left(8\frac{2}{3}+o\left(1\right)\right)n^2$	$50.21\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7}M-18n^2$
[24]	$3n^2$ [57]	$28.05\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7}M-15n^2$
[10]	$3n^2$ [48]	$23.37\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7}M-6n^2+12.73n^2\log_2\left(\frac{n}{\sqrt{M}}\right)$
[48]	$\left(2\frac{2}{3}+o\left(1\right)\right)n^2$	$18.82\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7}M-6.42n^2+2.81n^2\log_2\left(\frac{n}{\sqrt{M}}\right)$
Ours	$\left(2\frac{2}{3}+o\left(1\right)\right)n^2$	$18.82\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7}M-3.92n^2+5.75n^2\log_2\left(\frac{n}{\sqrt{M}}\right)$

Every $\langle 2,2,2;7\rangle$-algorithm with coefficients in $\mathbb{Z}$ can be obtained under the action of Equation (12) (cf. [55], [56]), allowing us to obtain the best-known stability factor in the algorithm class using such a transformation. However, this does not generally hold for other classes of algorithms, which we explore next.

B. Speeding Up Stable Algorithms

To speed up a stable algorithm, one can sparsify its operators using methods of [10]–[13]. The resulting algorithm has fewer additions than the standard basis algorithm in its bilinear phase and the same stability factor by Corollary III.9, albeit at the minor cost of a higher prefactor. Table II lists existing algorithms, their stability bounds, and their arithmetic complexities before and after decomposition. The alternative basis algorithms are either taken from the algorithms obtained in [12] or computed using one of their three heuristics (specifically, the one utilizing Z3 [12]). The $\langle 3,2,3;15\rangle$-algorithm and $\langle 4,4,2;7\rangle$-algorithm are the stable algorithms demonstrated in [39]. The $\langle 3,3,3;23\rangle$-algorithm [22] is the most stable algorithm demonstrated in [39]. We decompose these algorithms to obtain faster algorithms with the same stability factor. We obtain here the first decomposition of the $\langle 3,4,5;47\rangle$-algorithm of [54], which is the fastest $\langle 3,4,5;47\rangle$-algorithm to date. Our alternative basis version is faster than the original algorithm by a factor approaching $^{298}/_{238}$ and with the same stability factor.

C. Trade-off Between Speed and Stability

We consider further algorithm classes, starting with $\langle 3,3,3;23\rangle$-algorithms (see Figure 1). In the standard basis, Smirnov's algorithm [22] obtains the least number of additions and has the lowest stability factor of all the compared algorithms. However, in an alternative basis, some algorithms have fewer additions in their bilinear phase than the alternative basis version of Smirnov's algorithm. The 152-nnz algorithm by [14], [17], [19], when in an alternative basis [12], has the lowest number of additions and the second best stability factor. Unlike the class of $\langle 2,2,2;7\rangle$-algorithms, where we obtain an algorithm that is optimal for both speed and stability

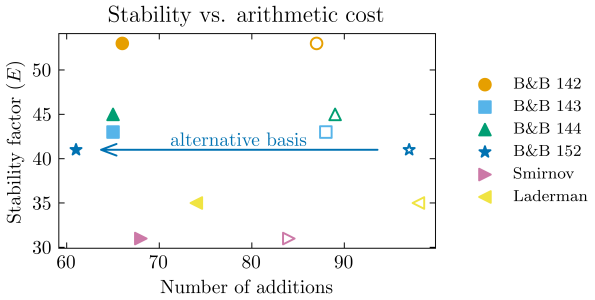

Fig. 1: Scatter plot of stability factor and number of additions for various $\langle 3,3,3;23\rangle$ algorithms. The empty faces represent standard basis algorithms, and the full faces represent the respective alternative basis versions. Notably, the stability factor remains the same, but the number of additions in the bilinear phase is reduced. Hence, the new algorithms are faster.

properties, the $\langle 3,3,3;23\rangle$-algorithms class may offer a trade-off between speed and stability. However, this set has over 17 thousand algorithms [58], and we leave the search for better algorithms to future work.

V. Stability Improvement Methods

This section examines several methods for improving stability. We show that diagonal scaling methods can be effectively applied to alternative basis algorithms. In contrast, we demonstrate that non-uniform approaches such as those provided by [35], [36] do not readily extend to the alternative basis method in the full version of this paper.

A. Scaling

Regrettably, the bound in Theorem III.8 allows for a relatively large error when $|c_{ij}|$ is small compared to $\|\mathbf{A}\|\cdot\|\mathbf{B}\|$. This issue is common in fast matrix multiplication algorithms. We address such vulnerabilities using techniques known as *scaling* [34], [39]–[42], [46]. The concept that underlies these techniques relies on a simple observation:

$$\mathbf{C}=\mathbf{D}_A\left(\mathbf{D}_A^{-1}\mathbf{A}\mathbf{D}\right)\left(\mathbf{D}^{-1}\mathbf{B}\mathbf{D}_B^{-1}\right)\mathbf{D}_B, \tag{14}$$

for any non-singular scaling matrices (i.e., diagonal matrices) $\mathbf{D}_A,\mathbf{D}_B$, and $\mathbf{D}$. By employing scaling matrices $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}$ that can be easily applied to pre-process $\mathbf{A}$ and $\mathbf{B}$ and to post-process $\mathbf{C}$, it is possible to tighten the norm-wise bound in Theorem III.8 at a negligible cost of $O\left(MK+KN+MN\right)$.

Ballard et al. [39] provides a comprehensive review of scaling techniques, including *outside scaling* and *inside scaling*. *Outside scaling* was first suggested by [40], which provides the following diagonal matrices:

$$\mathbf{D}_A=\mathrm{diag}\left(\max_j|a_{ij}|\right),\quad \mathbf{D}_B=\mathrm{diag}\left(\max_i|b_{ij}|\right).$$

To use outside scaling, one first computes $\mathbf{A}'\leftarrow\mathbf{D}_A^{-1}\mathbf{A}$ and $\mathbf{B}'\leftarrow\mathbf{D}_B^{-1}\mathbf{B}$, and then computes $\mathbf{C}'\leftarrow\mathbf{A}'\cdot\mathbf{B}'$ using a fast algorithm. Finally, we compute $\mathbf{C}\leftarrow\mathbf{D}_A\mathbf{C}'\mathbf{D}_B$. *Inside*

scaling was first presented by Brent [30] and was more carefully referenced by [34], [41], [42], [46]. The diagonal values were first explicitly stated in [39]:

$$\mathbf{D} = \operatorname{diag}\left(\sqrt{\max_j |b_{kj}| / \max_i |a_{ik}|}\right).$$

To apply inside scaling, we first compute $\mathbf{A}' \leftarrow \mathbf{AD}$, $\mathbf{B}' \leftarrow \mathbf{D}^{-1}\mathbf{B}$, and then return $\mathbf{C} = \mathbf{A}' \cdot \mathbf{B}'$ with a fast algorithm. Ballard et al. [39] also suggest repeated application of outside and inside scaling in alternating order, denoted *repeated outside-inside scaling* (R-O-I). When peforming only one iteration, performing outside scaling first is denoted outside-inside (O-I) scaling, while starting with inside scaling is denoted inside-outside (I-O) scaling. They prove several propositions and theorems that bound the relative error of the output of a fast algorithm when using these scaling techniques. We next provide the following theorem, which is a direct corollary of their results:

Theorem V.1. *Let ALG be an alternative basis algorithm. Then, every claim for the effectiveness of scaling for improving stability from [39] also applies to ALG. This holds, in particular, to Propositions 9,11,12 of [39].*

See Appendix B for the details of these claims.

Proof. The only assumption that [39] makes is that ALG exhibits an error bound of the form

$$|\hat{c}_{ij} - c_{ij}| \leq f_{ALG}(K) \|\mathbf{A}\| \|\mathbf{B}\| \epsilon + O\left(\epsilon^2\right), \tag{15}$$

where f_{ALG} is a polynomial function at worst. By Theorem III.8, this holds for ALG as well. □

In practice, the error in computing c_{ij} depends on which elements of $\mathbf{A}$ and $\mathbf{B}$ are involved in its computation and their order of magnitude compared to c_{ij}. We next show that when such an error occurs in a standard basis fast algorithm, it also occurs in its alternative basis counterpart. Formally:

Claim V.2. Let ALG_A be an alternative basis algorithm with matrices $\mathbf{U}_\phi, \mathbf{V}_\psi, \mathbf{W}_\nu$, ϕ, ψ, ν, and let ALG_S be the standard basis version of ALG_A, that is, with encoding/decoding matrices $\langle \mathbf{U}, \mathbf{V}, \mathbf{W} \rangle = \langle \phi \cdot \mathbf{U}_\phi, \psi \cdot \mathbf{V}_\psi, \nu \cdot \mathbf{W}_\nu \rangle$. Consider an output block $\mathbf{C}_{ij}$. Then every block of $\mathbf{A}$ and $\mathbf{B}$ that is involved in the computation of $\mathbf{C}_{ij}$ by ALG_S is also involved in computing $\mathbf{C}_{ij}$ by ALG_A.

Proof. For simplicity, we assume a single level of recursion. The general case is similar. Let $(m,k) \in [M_0] \times [K_0]$ such that $\mathbf{A}_{mk}$ is involved in $\mathbf{C}_{ij}$ by ALG_S. By Equation (2), there exists an r such that $u_{(m,k),r} \neq 0$ and $w_{(i,j),r} \neq 0$. We refer by $u_{(m,k),r}$ to the element in the r'th column corresponding with the index (m,k) of the vectorization of $\mathbf{A}$, and similarly for $w_{(i,j),r}$. Since $u_{(m,k),r} = \left\langle \phi_{(m,k),:}, \mathbf{U}^{\phi}_{:r} \right\rangle \neq 0$, there exists some $p \in [D_U]$ such that $\phi_{(m,k),p} \neq 0$ and $u^{\phi}_{pr} \neq 0$. Similarly, since $w_{(i,j)r} = \left\langle \nu_{(i,j),:}, w^{\nu}_{:r} \right\rangle \neq 0$, there exists some $q \in [D_W]$ such that $\nu_{(i,j),q} \neq 0$ and $w^{\nu}_{qr} \neq 0$. By definition of ALG_A, we obtain that $\mathbf{A}_{(m,k)}$ is involved in the computation $\phi(\mathbf{A})_p$, $\phi(\mathbf{A})_p$ is involved in $\nu(\mathbf{C})_q$, and finally, $\nu(C)_q$ is involved in the computation of $\mathbf{C}_{(i,j)}$. The proof for $\mathbf{B}$ is identical. □

We note that the contrary may not hold since the computation of c_{ij} by ALG_A may involve elements of $\mathbf{A}, \mathbf{B}$ that are canceled out in basis transformations and are not involved in the computation of c_{ij} by ALG_S. Diagonal scaling methods potentially balance the sizes of the entries invloved in the computation of c_{ij}. We conclude that diagonal scaling can be beneficial for alternative basis algorithms whenever it is beneficial for their standard basis counterparts. Our experiments demonstrate that scaling is effective in practice for alternative basis matrix multiplication. We repeat the scaling experiments of [39] and apply them for alternative basis algorithms as well in Section VI-C.

VI. Experimental Results

In this section, we provide numerical experiments that empirically support our theoretical findings.

A. A fast and stable algorithm

We provide an experiment measuring the speed and absolute error of several $\langle 2,2,2;7 \rangle$-algorithms, including ours.

1) Experiment Settings. We implemented our high-performance algorithm in C++17 based on the optimizations and parallelization techniques presented by [48]. Roughly speaking, the implementation uses fully vectorized loops that are parallelized using OpenMP directives to compute linear combinations. We use a block recursive data layout, and when reaching sufficiently small matrices, we invoke Intel's MKL DGEMM using all available threads. Our algorithm utilizes the exact bilinear phase implementation as that of [48], and new basis transformations (see Appendix A), which are optimized using the same techniques as of [48]. We also implemented Strassen's algorithm [1] with a naïve schedule and Winograd's improvement [24] with the schedule provided by [57], utilizing the same parallelization techniques and optimizations, making these implementations faster than existing ones and therefore comparable to our algorithm and to that of [48]. We compare the runtimes and numerical errors to that of Intel's MKL DGEMM [59]. The experiments were conducted on a dual-socket Intel Gold 6248 processor, Cascade Lake microarchitecture, with a base frequency of 2.50GHz and 20 cores per socket (see Appendix C for full specifications). We run the algorithms in a parallel setting with different numbers of recursion steps and matrix sizes and measure the average runtime over 100 runs. We measure the maximal absolute error over 100 runs for multiplying two matrices of size 4096×4096, with random entries distributed uniformly in $(-1,1)$ and in $(0,1)$, and compared to classical matrix multiplication in quadruple precision.

2) Results. Figure 2 (A) plots the average runtime for an increasing matrix size, normalized by the runtime of Intel's MKL DGEMM [59]. It shows that the performance of our algorithm is on par with Schwartz and Vaknin's high-performance parallel implementation. Figure 2 (B) plots the average runtime for a constant matrix size 8192×8192

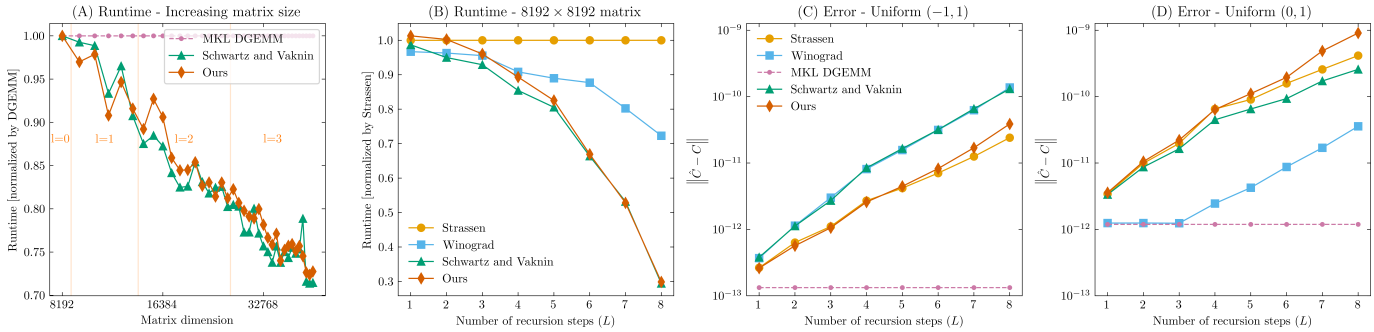

Fig. 2: Runtime and Error of different algorithms. Our algorithm is comparable to the best in class in both runtime and error, except for the uniform $(0, 1)$ distribution, where Winograd's variant has the least error.

and a varying number of recursion steps. Our algorithm and Schwartz and Vaknin's have the lowest average runtime, followed by Winograd's and then by Strassen's, thus matching our theoretical results in Table I. Figure 2 (C) and (D) plot the maximal absolute errors of algorithms over 100 runs for uniform distributions $(-1, 1)$ and $(0, 1)$. Figure 2 (C) shows that our algorithm and Strassen's, both with a stability factor $E = 12$, obtain more accurate results, whereas Winograd's and Schwartz and Vaknin's, both with a stability factor of $E = 18$, are less accurate. Figure 2 (D) shows generally larger errors for all algorithms, including DGEMM. However, their order differs, with Winograd's variant being the most stable. Both phenomena are known and explained by D'Alberto [36]. Higham [34] and Ballard et al. [39] also notice that for non-negative matrices, algorithms' errors correlate less with their stability factors and more with the number of non-zero entries in their encoding/decoding matrices (for which, Winograd's is lowest).

B. Error of other alternative basis algorithms

We next provide an experiment demonstrating Theorem III.8 for the general decomposed recursive-bilinear framework [13]. We compare several $\langle 3, 3, 3; 23\rangle$-algorithms that compute their bilinear phase in the standard basis, in an alternative basis, in higher dimensions, and in a full decomposition [13].

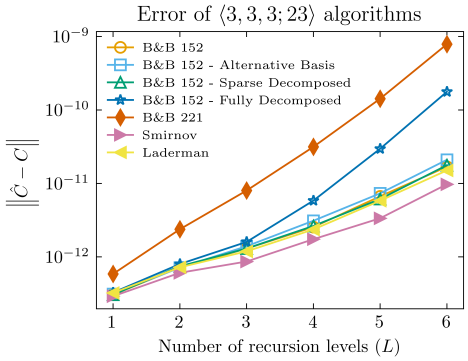

Fig. 3: Errors of different $\langle 3, 3, 3; 23\rangle$ algorithms. There is a correlation between the stability factors and the errors of the algorithms.

1) Experiment Settings. We provide different implementations of the 152-non-zeros algorithm by [14], [17], [19] (denoted B&B 152). We implemented an alternative basis version [12], a higher-dimension decomposed and fully decomposed version [13]. For comparative purposes, we implemented B&B 221 [14], Smirnov's [22] and Laderman's [19] $\langle 3, 3, 3; 23\rangle$-algorithms. The bilinear phases of the algorithms were implemented using an automatic code generation tool based on the framework of [14]. All the experiments were conducted on the same machine as in Section VI-A. The experiment was conducted by multiplying two matrices of size 2187×2187, with random entries distributed uniformly in $(-1, 1)$.

2) Results. Figure 3 shows the maximal error over 100 runs. The stability factors dominate the algorithms' errors- B&B 152 (with stability $E = 41$) is less accurate than Smirnov's algorithm (with $E = 31$) and more accurate than B&B 221 (with $E = 139$). The ordering of accuracy within the decompositions correlates with their prefactors (the generalization will appear in the full version of this paper): the standard version is most accurate ($Q = 13$), followed by the higher-dimension decomposed version ($Q = 16$), its alternative basis version ($Q = 21$), and its fully decomposed version ($Q = 61$).

C. Scaling with the alternative basis method

We next measure the relative errors of our algorithm compared to Strassen's algorithm, combined with different scaling methods. We replicate the experiments of [14] with different adversarial distributions that show scaling methods are effective for alternative basis algorithms. Ballard and Benson [14] presented several adversarial examples for comparing the effectiveness of each scaling type. We take these adversarial distributions and apply them to Strassen's algorithm and an alternative basis version of Strassen's algorithm.

1) Experiment Settings. The implementation of the bilinear phases of the algorithms and the scaling methods are based on the framework of [14], [39]. The experiments were conducted on the same machine as in Section VI-A. We sampled 100 matrices of size 2048×2048 from 3 distributions that are defined in [39]. Distrbution 1 is uniform from $(0, 1)$ i.i.d for every a_{ij} and b_{ij}. Distribution

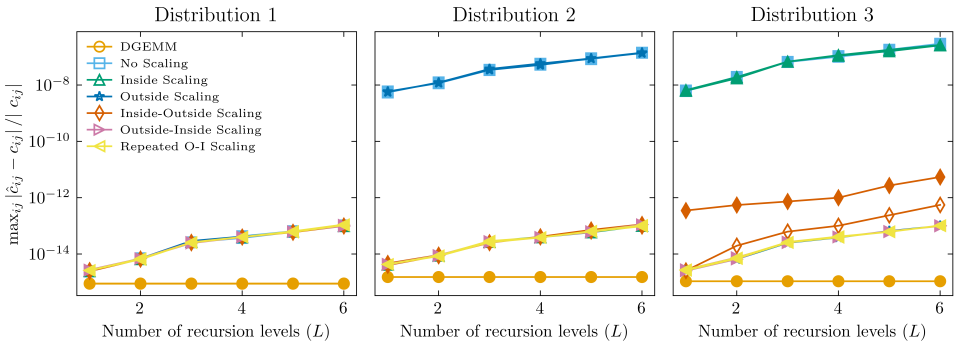

Fig. 4: The relative stability of Strassen's algorithm and an alternative basis version of it. The markers with blank faces represent the standard basis algorithm, and markers with colored faces are their alternative basis counterparts. In the graphs, they almost completely overlap.

2 is $a_{ij} \sim \text{Uniform}\left(0, 1/N^2\right)$ if $j > N/2$, otherwise, $a_{ij} \sim \text{Uniform}(0,1)$; $b_{ij} \sim \text{Uniform}\left(0, 1/N^2\right)$ if $i < N/2$, otherwise $b_{ij} \sim \text{Uniform}(0,1)$. Distribution 3 is $a_{ij} \sim \text{Uniform}\left(0, N^2\right)$ if $i < N/2$ and $j > N/2$, otherwise, $a_{ij} \sim \text{Uniform}(0,1)$; $b_{ij} \sim \text{Uniform}\left(0, 1/N^2\right)$ if $j < N/2$, otherwise $b_{ij} \sim \text{Uniform}(0,1)$. The error reported was the maximum value of $\max_{ij}|\hat{c}_{ij} - c_{ij}|/|c_{ij}|$ over the 100 samples, where c_{ij} were computed with the classical algorithm, using quadruple precision.

2) Results. Figure 4 shows the alternative basis version of Strassen's algorithm identifies almost entirely with the errors of Strassen's algorithm. Errors for distribution 1 are well-behaved for fast matrix multiplication algorithms. For distribution 2, which is designed such that outside scaling will be ineffective [39], Strassen's algorithm and its alternative basis version remain stable with inside scaling. Distribution 3 is designed so inside scaling will be ineffective [39], and we can see that both standard and alternative bases perform well with inside scaling. Thus, with no prior knowledge of the distribution, repeated inside-outside scaling remains a safe and effective choice for maintaining low errors for both standard and alternative basis matrix multiplication.

VII. Conclusion

This study shows that alternative basis algorithms are stable and have the same asymptotic error bound as their standard basis counterparts. While the accuracy of alternative basis matrix multiplication is not as good as that of classical matrix multiplication, it can play a significant role in various modern use cases, such as most machine learning algorithms and many simulation applications, where speed is prioritized, and the error has negligible or no effect at all. Our work shows that alternative basis matrix multiplication is a plausible kernel for such cases. For a given fast matrix multiplication class of algorithms, the performance is dominated by the number of additions in the bilinear phase of the algorithms, and the error bound depends on their stability factor. These properties were previously thought to exhibit a trade-off. We demonstrate that by using the alternative basis method, one can simultaneously optimize both.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 818252). This project has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101113120, 101138056). Research was supported by grants No. 1354/23 and 1919/19 of the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities). This research has been supported by the Science Accelerator and by the Frontiers in Science initiative of the Ministry of Innovation, Science and Technology. Research was also supported by the Minerva Foundation. This work was supported by computing time awarded on the Cyclone supercomputer of the High Performance Computing Facility of The Cyprus Institute under project ID 170.

Appendix A
Communication Costs Analysis

Definition A.1 (Communication Costs [42])**.** The communication costs of an algorithm are the number of read and write operations between the cache and main memory in a shared memory model, and the number of send and receive messages between processors, in a distributed memory model. The communication costs are measured as a function of the number of processors P, the local memory size M in words, and the matrix dimension n.

A. Basis Transformations of Our Algorithm

The basis transformations of our $\langle 2,2,2;7\rangle$-algorithm are given by:

$$\phi = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \psi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix},$$

$$\nu^{-1} = \begin{pmatrix} 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 \end{pmatrix}.$$

B. Communcation Costs Analysis

Our basis transformations are computed in place, utilizing avx512 intrinsics. While the claim in [48] assumes ± 1 on the main diagonal of each basis transformation, it is possible to compute these basis transformations in place.

The communication costs for the basis transfers, with loop merging, are:

$$IO_{\psi}(n,m,M) \leq \left(\frac{|R_{\psi}|+|W_{\psi}|}{n_0^2}\right) \cdot n^2 \log_{n_0}\left(\sqrt{\frac{|R_{\psi}|}{n_0^2}} \cdot \frac{n}{\sqrt{M}}\right) + \left(\frac{|R_{\psi}|+|W_{\psi}|}{n_0^2}\right) n^2.$$

The communication costs for the bilinear phase are the same as for [48]:

$$18.82\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M - 9.67n^2. \tag{16}$$

For each basis transformation, we have:

$$IO_{\phi}(n,m,M) \leq 2n^2 \log_{n_0}\left(\frac{n}{\sqrt{M}}\right) + 2n^2$$

$$IO_{\psi}(n,m,M) \leq \frac{7}{4}n^2 \log_{n_0}\left(\frac{n}{\sqrt{M}}\right) + \frac{7}{4}n^2$$

$$IO_{\nu}(n,m,M) \leq 2n^2 \log_{n_0}\left(\frac{n}{\sqrt{M}}\right) + 2n^2,$$

for a total of

$$18.82\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M - 3.92n^2 + 5.75n^2 \log_2\left(\frac{n}{\sqrt{M}}\right).$$

We provide a naive implementation of Strassen's algorithm [1]. Its communication costs are given by

$$\begin{aligned} IO_{ALG}(n,M) \leq & 3\left(8\frac{2}{3} + o(1)\right)^{\frac{\log_2 7}{2}-1}\left(1+\frac{18}{7-4}\right) \\ & \cdot \left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M - 3\left(\frac{18}{7-4}\right)n^2 \\ = & 50.21 \cdot \left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M - 3\left(\frac{18}{7-4}\right)n^2. \end{aligned}$$

APPENDIX B
EXISTING SCALING CLAIMS

In the following claims from [39], we treat f_{ALG} as a constant factor, that is, $f_{ALG}(K,L) \cdot \epsilon = O(\epsilon)$.

Claim B.1 (Effectiveness of outside scaling [39]). Using outside scaling with input matrices $\mathbf{A}$ and $\mathbf{B}$, we obtain:

$$|c_{ij} - \hat{c}_{ij}| \leq O(\epsilon) \|a_{i,:}\| \|b_{:,j}\|. \tag{17}$$

Claim B.2 (Effectiveness of inside scaling [39]). Using inside scaling with input matrices $\mathbf{A}$ and $\mathbf{B}$, we obtain:

$$\|\hat{\mathbf{C}} - \mathbf{C}\| \leq O(\epsilon) \max_{i,k,j} |a_{ik}| \, |b_{kj}|. \tag{18}$$

Claim B.3 (Effectiveness of repeated outside-inside scaling). Let t be the number of steps of outside-inside scaling that we complete. The computed product satisfies

$$|c_{ij} - \hat{c}_{ij}| \leq O(\epsilon) r_i^{(t)} s_j^{(t)} \left\|\mathbf{A}^{(t)}\right\| \left\|\mathbf{B}^{(t)}\right\|, \tag{19}$$

where $r_i^{(t)}$ and $s_j^{(t)}$ denote the diagonal elements of $\mathbf{D}_A$ and $\mathbf{D}_B$, respectively, after t steps.

APPENDIX C
EXPERIMENT PROCESSOR SPECIFICATIONS

All our experiments were conducted on an Intel Xeon Gold 6248 processor. Microarchitecture: Cascade Lake; Base frequency: 2.50GHz; Number of cores per socket: 20; Hyperthreads per core: 1; SIMD vector width: 512bit; L1d cache per core: Size: 32KB, Associativity: 8, Cache line size: 64; L2 cache per core: Size: 1024KB, Associativity: 16, Cache line size: 64; L3 cache per socket: Size: 28160KB, Associativity: 11, Cache line size: 64; Memory: DDR4- 2933Mhz 6 channels; Memory bandwidth: 140.8GB/s; Compiler: GCC 11.2.0; Intel MKL version: 2022.1.0.

REFERENCES

[1] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, no. 4, pp. 354–356, Aug. 1969. [Online]. Available: http://link.springer.com/10.1007/BF02165411

[2] J. Alman and V. V. Williams, "A Refined Laser Method and Faster Matrix Multiplication," in *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '21. USA: Society for Industrial and Applied Mathematics, 2021, pp. 522–539, event-place: Virtual Event, Virginia.

[3] D. Bini, "Relations between exact and approximate bilinear algorithms. Applications," *Calcolo*, vol. 17, no. 1, pp. 87–97, Jan. 1980. [Online]. Available: http://link.springer.com/10.1007/BF02575865

[4] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251–280, Mar. 1990. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0747717108800132

[5] H. Cohn and C. Umans, "A group-theoretic approach to fast matrix multiplication," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* Cambridge, MA, USA: IEEE Computer. Soc, 2003, pp. 438–449. [Online]. Available: http://ieeexplore.ieee.org/document/1238217/

[6] A. M. Davie and A. J. Stothers, "Improved bound for complexity of matrix multiplication," *Proceedings of the Royal Society of Edinburgh: Section A Mathematics*, vol. 143, no. 2, pp. 351–369, Apr. 2013. [Online]. Available: https://www.cambridge.org/core/product/identifier/S0308210511001648/type/journal_article

[7] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. Kobe Japan: ACM, Jul. 2014, pp. 296–303. [Online]. Available: https://dl.acm.org/doi/10.1145/2608628.2608664

[8] A. Schönhage, "Partial and Total Matrix Multiplication," *SIAM Journal on Computing*, vol. 10, no. 3, pp. 434–455, Aug. 1981. [Online]. Available: http://epubs.siam.org/doi/10.1137/0210032

[9] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. New York New York USA: ACM, May 2012, pp. 887–898. [Online]. Available: https://dl.acm.org/doi/10.1145/2213977.2214056

[10] E. Karstadt and O. Schwartz, "Matrix Multiplication, a Little Faster," in *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*. Washington DC USA: ACM, Jul. 2017, pp. 101–110. [Online]. Available: https://dl.acm.org/doi/10.1145/3087556.3087579

[11] ——, "Matrix Multiplication, a Little Faster," *Journal of the ACM*, vol. 67, no. 1, pp. 1–31, Feb. 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3364504

[12] G. Beniamini, N. Cheng, O. Holtz, E. Karstadt, and O. Schwartz, "Sparsifying the Operators of Fast Matrix Multiplication Algorithms," 2020, publisher: arXiv Version Number: 1. [Online]. Available: https://arxiv.org/abs/2008.03759

[13] G. Beniamini and O. Schwartz, "Faster Matrix Multiplication via Sparse Decomposition," in *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*. Phoenix AZ USA: ACM, Jun. 2019, pp. 11–22. [Online]. Available: https://dl.acm.org/doi/10.1145/3323165.3323188

[14] A. R. Benson and G. Ballard, "A framework for practical parallel fast matrix multiplication," in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. San Francisco CA USA: ACM, Jan. 2015, pp. 42–53. [Online]. Available: https://dl.acm.org/doi/10.1145/2688500.2688513

[15] M. Cenk and M. A. Hasan, "On the arithmetic complexity of Strassen-like matrix multiplications," *Journal of Symbolic Computation*, vol. 80, pp. 484–501, May 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0747717116300359

[16] J. E. Hopcroft and L. R. Kerr, "On Minimizing the Number of Multiplications Necessary for Matrix Multiplication," *SIAM Journal on Applied Mathematics*, vol. 20, no. 1, pp. 30–36, Jan. 1971. [Online]. Available: http://epubs.siam.org/doi/10.1137/0120004

[17] R. W. Johnson and A. M. McLoughlin, "Noncommutative Bilinear Algorithms for 3 x 3 Matrix Multiplication," *SIAM Journal on Computing*, vol. 15, no. 2, pp. 595–603, May 1986. [Online]. Available: http://epubs.siam.org/doi/10.1137/0215043

[18] J. Laderman, V. Pan, and X.-H. Sha, "On practical algorithms for accelerated matrix multiplication," *Linear Algebra and its Applications*, vol. 162-164, pp. 557–588, Feb. 1992. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/002437959290393O

[19] J. D. Laderman, "A noncommutative algorithm for multiplying 3 x 3 matrices using 23 multiplications," *Bulletin of the American Mathematical Society*, vol. 82, pp. 126–128, 1976. [Online]. Available: https://api.semanticscholar.org/CorpusID:121295009

[20] V. Y. Pan, "Strassen's algorithm is not optimal trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations," *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pp. 166–176, 1978. [Online]. Available: https://api.semanticscholar.org/CorpusID:14348408

[21] V. Pan, "Trilinear aggregating with implicit canceling for a new acceleration of matrix multiplication," *Computers & Mathematics with Applications*, vol. 8, no. 1, pp. 23–34, 1982. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0898122182900372

[22] A. V. Smirnov, "The bilinear complexity and practical algorithms for matrix multiplication," *Computational Mathematics and Mathematical Physics*, vol. 53, no. 12, pp. 1781–1795, Dec. 2013. [Online]. Available: http://link.springer.com/10.1134/S0965542513120129

[23] ——, "Several bilinear algorithms for matrix multiplication," Tech. Rep., 2017.

[24] S. Winograd, "On multiplication of 2 × 2 matrices," *Linear Algebra and its Applications*, vol. 4, no. 4, pp. 381–388, Oct. 1971. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0024379571900097

[25] R. L. Probert, "On the Additive Complexity of Matrix Multiplication," *SIAM Journal on Computing*, vol. 5, no. 2, pp. 187–203, Jun. 1976. [Online]. Available: http://epubs.siam.org/doi/10.1137/0205016

[26] N. H. Bshouty, "On the additive complexity of 2 × 2 matrix multiplication," *Information Processing Letters*, vol. 56, no. 6, pp. 329–335, Dec. 1995. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/002001909500176X

[27] M. Bodrato, "A Strassen-like Matrix Multiplication Suited for Squaring and Higher Power Computation," in *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ser. ISSAC '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 273–280, event-place: Munich, Germany. [Online]. Available: https://doi.org/10.1145/1837934.1837987

[28] Y. Moran and O. Schwartz, "Multiplying 2 × 2 Sub-Blocks Using 4 Multiplications," in *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 379–390, event-place: Orlando, FL, USA. [Online]. Available: https://doi.org/10.1145/3558481.3591083

[29] R. P. Brent, "Algorithms for matrix multiplication," Stanford University, Technical Report TR-CS-70-157, Mar. 1970.

[30] ——, "Error analysis of algorithms for matrix multiplication and triangular decomposition using Winograd's identity," *Numerische Mathematik*, vol. 16, no. 2, pp. 145–156, Nov. 1970. [Online]. Available: http://link.springer.com/10.1007/BF02308867

[31] W. Miller, "Computational Complexity and Numerical Stability," *SIAM Journal on Computing*, vol. 4, no. 2, pp. 97–107, Jun. 1975, publisher: Society for Industrial & Applied Mathematics (SIAM). [Online]. Available: https://doi.org/10.1137%2F0204009

[32] D. Bini and G. Lotti, "Stability of fast algorithms for matrix multiplication," *Numerische Mathematik*, vol. 36, no. 1, pp. 63–72, Mar. 1980. [Online]. Available: http://link.springer.com/10.1007/BF01395989

[33] N. J. Higham, "Exploiting fast matrix multiplication within the level 3 BLAS," *ACM Transactions on Mathematical Software*, vol. 16, no. 4, pp. 352–368, Dec. 1990. [Online]. Available: https://dl.acm.org/doi/10.1145/98267.98290

[34] ——, *Accuracy and stability of numerical algorithms*, 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics, 2002.

[35] R. R. Castrapel and J. L. Gustafson, "Precision improvement method for the Strassen/Winograd matrix multiplication method," US Patent Patent 7,209,939, Apr., 2007. [Online]. Available: http://www.google.com/patents/US7209939

[36] P. D'Alberto, "The Better Accuracy of Strassen-Winograd Algorithms (FastMMW)," *Advances in Linear Algebra & Matrix Theory*, vol. 04, no. 01, pp. 9–39, 2014. [Online]. Available: http://www.scirp.org/journal/doi.aspx?DOI=10.4236/alamt.2014.41002

[37] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Graph expansion and communication costs of fast matrix multiplication," *Journal of the ACM*, vol. 59, no. 6, pp. 1–23, Dec. 2012. [Online]. Available: https://dl.acm.org/doi/10.1145/2395116.2395121

[38] J. Demmel, I. Dumitriu, O. Holtz, and R. Kleinberg, "Fast matrix multiplication is stable," *Numerische Mathematik*, vol. 106, no. 2, pp. 199–224, Feb. 2007, publisher: Springer Science and Business Media LLC. [Online]. Available: https://doi.org/10.1007%2Fs00211-007-0061-6

[39] G. Ballard, A. R. Benson, A. Druinsky, B. Lipshitz, and O. Schwartz, "Improving the Numerical Stability of Fast Matrix Multiplication," *SIAM Journal on Matrix Analysis and Applications*, vol. 37, no. 4, pp. 1382–1418, Jan. 2016, publisher: Society for Industrial & Applied Mathematics (SIAM). [Online]. Available: https://doi.org/10.1137%2F15m1032168

[40] B. Dumitrescu, "Improving and estimating the accuracy of Strassen's algorithm," *Numerische Mathematik*, vol. 79, no. 4, pp. 485–499, Jun. 1998. [Online]. Available: http://link.springer.com/10.1007/s002110050348

[41] J. Demmel, I. Dumitriu, and O. Holtz, "Fast linear algebra is stable," *Numerische Mathematik*, vol. 108, no. 1, pp. 59–91, Oct. 2007. [Online]. Available: http://link.springer.com/10.1007/s00211-007-0114-x

[42] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Communication-optimal parallel algorithm for strassen's matrix multiplication," in *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*. Pittsburgh Pennsylvania USA: ACM, Jun. 2012, pp. 193–204. [Online]. Available: https://dl.acm.org/doi/10.1145/2312005.2312044

[43] F. Desprez and F. Suter, "Impact of Mixed-Parallelism on Parallel Implementations of the Strassen and Winograd Matrix Multiplication Algorithms: Research Articles," *Concurr. Comput. : Pract. Exper.*, vol. 16, no. 8, pp. 771–797, Jul. 2004, place: GBR Publisher: John Wiley and Sons Ltd.

[44] B. Grayson and R. Van De Geijn, "A High Performance Parallel Strassen Implementation," *Parallel Processing Letters*, vol. 06, no. 01, pp. 3–12, Mar. 1996. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0129626496000029

[45] Q. Luo and J. B. Drake, "A Scalable Parallel Strassen's Matrix Multiplication Algorithm for Distributed-Memory Computers," in *Proceedings of the 1995 ACM Symposium on Applied Computing*, ser. SAC '95. New York, NY, USA: Association for Computing Machinery, 1995, pp. 221–226, event-place: Nashville, Tennessee, USA. [Online]. Available: https://doi.org/10.1145/315891.315965

[46] B. Lipshitz, G. Ballard, J. Demmel, and O. Schwartz, "Communication-Avoiding Parallel Strassen: Implementation and Performance," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Washington, DC, USA: IEEE Computer Society Press, 2012, event-place: Salt Lake City, Utah.

[47] W. F. McColl and A. Tiskin, "Memory-Efficient Matrix Multiplication in the BSP Model," *Algorithmica*, vol. 24, no. 3-4, pp. 287–297, Jul. 1999. [Online]. Available: http://link.springer.com/10.1007/PL00008264

[48] O. Schwartz and N. Vaknin, "Pebbling Game and Alternative Basis for High Performance Matrix Multiplication," *SIAM Journal on Scientific Computing*, 2023, to appear.

[49] L. E. Cannon, "A Cellular Computer to Implement the Kalman Filter Algorithm," PhD Thesis, Montana State University, USA, 1969.

[50] R. A. Van De Geijn and J. Watts, "SUMMA: scalable universal matrix multiplication algorithm," *Concurrency: Practice and Experience*, vol. 9, no. 4, pp. 255–274, Apr. 1997. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/(SICI)1096-9128(199704)9:4⟨255::AID-CPE250⟩3.0.CO;2-2

[51] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar, "A three-dimensional approach to parallel matrix multiplication," *IBM Journal of Research and Development*, vol. 39, no. 5, pp. 575–582, Sep. 1995. [Online]. Available: http://ieeexplore.ieee.org/document/5389455/

[52] E. Solomonik and J. Demmel, "Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms," in *Euro-Par 2011 Parallel Processing*, E. Jeannot, R. Namyst, and J. Roman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6853, pp. 90–109, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-642-23397-5_10

[53] M. Karppa and P. Kaski, "Engineering Boolean Matrix Multiplication for Multiple-Accelerator Shared-Memory Architectures," 2019, publisher: arXiv Version Number: 1. [Online]. Available: https://arxiv.org/abs/1909.01554

[54] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, Oct. 2022. [Online]. Available: https://www.nature.com/articles/s41586-022-05172-4

[55] R. W. Brockett and D. Dobkin, "On the optimal evaluation of a set of bilinear forms," *Linear Algebra and its Applications*, vol. 19, no. 3, pp. 207–235, 1978. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0024379578900125

[56] H. F. De Groote, "On varieties of optimal algorithms for the computation of bilinear mappings I. the isotropy group of a bilinear mapping," *Theoretical Computer Science*, vol. 7, no. 1, pp. 1–24, 1978. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0304397578900385

[57] B. Boyer, J.-G. Dumas, C. Pernet, and W. Zhou, "Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm," in *Proceedings of the 2009 international symposium on Symbolic and algebraic computation*. Seoul Republic of Korea: ACM, Jul. 2009, pp. 55–62. [Online]. Available: https://dl.acm.org/doi/10.1145/1576702.1576713

[58] M. J. Heule, M. Kauers, and M. Seidl, "New ways to multiply 3×3-matrices," *Journal of Symbolic Computation*, vol. 104, pp. 899–916, May 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0747717120301139

[59] "Intel® oneAPI math kernel library," 2020. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html