SAND2015-5246J

IMPROVING THE NUMERICAL STABILITY OF FAST MATRIX MULTIPLICATION ALGORITHMS*

GREY BALLARD†, AUSTIN R. BENSON‡, ALEX DRUINSKY§, BENJAMIN LIPSHITZ¶, AND ODED SCHWARTZ‖

Abstract. Fast algorithms for matrix multiplication, or those that perform asymptotically fewer scalar operations than the classical algorithm, have been considered primarily of theoretical interest. Aside from Strassen's original algorithm, few fast algorithms have been efficiently implemented or used in practical applications. However, there exist many practical alternatives to Strassen's algorithm with varying performance and numerical properties. While fast algorithms are known to be numerically stable, their error bounds are slightly weaker than the classical algorithm.

We argue in this paper that the numerical sacrifice of fast algorithms, particularly for the typical use cases of practical algorithms, is not prohibitive, and we explore ways to improve the accuracy both theoretically and empirically. The numerical accuracy of fast matrix multiplication depends on properties of the algorithm and of the input matrices, and we consider both contributions independently. We generalize and tighten previous error analyses of fast algorithms, compare the properties among the class of known practical fast algorithms, and discuss algorithmic techniques for improving the error guarantees. We also present means for reducing the numerical inaccuracies generated by anomalous input matrices using various forms of diagonal scaling. Finally, we include empirical results that test the various improvement techniques, in terms of both their numerical accuracy and their performance.

1. Introduction. After Strassen's discovery of an algorithm for dense matrix-matrix multiplication in 1969 [24] that reduced the computational complexity from the classical $O(N^3)$ (for multiplying two $N \times N$ matrices) to $O(N^{\log_2 7})$, there has been extensive effort to understand *fast* matrix multiplication, based on algorithms with computational complexity exponent less than 3. From a theoretical perspective, there remains a gap between the best known lower bound [20] and best known upper bound [13] on the exponent. From a practical perspective, it is unlikely that the techniques for obtaining the best upper bounds on the exponent can be translated to *practical* algorithms that will execute faster than the classical one for reasonably sized matrices. In this paper, we are interested in the numerical stability of practical algorithms that have been demonstrated to outperform the classical algorithm (as well as Strassen's in some instances) on modern hardware [3].

Nearly all fast matrix multiplication algorithms are based on recursion, using a recursive rule that defines a method for multiplying matrices of fixed dimension $M_0 \times K_0$ by $K_0 \times N_0$ (we use the notation $\langle M_0, K_0, N_0 \rangle$) with fewer than $M_0 K_0 N_0$ scalar multiplications. For practical algorithms, these fixed dimensions need to be very small, typically $M_0, K_0, N_0 < 10$, as they define the factors by which the dimensions of sub-

*This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

†Sandia National Laboratories, Livermore, California. (gmballa@sandia.gov). This author's work was supported by an appointment to the Sandia National Laboratories Truman Fellowship in National Security Science and Engineering, sponsored by Sandia Corporation (a wholly owned subsidiary of Lockheed Martin Corporation) as Operator of Sandia National Laboratories under its U.S. Department of Energy Contract No. DE-AC04-94AL85000.

‡Institute for Computational and Mathematical Engineering, Stanford University, Stanford, California. (arbenson@stanford.edu). This author's work was supported by an Office of Technology Licensing Stanford Graduate Fellowship.

§Lawrence Berkeley National Laboratory, Berkeley, California. (adruinsky@lbl.gov)

¶Google (benjamin.lipshitz@gmail.com)

‖The Hebrew University odedsc@cs.huji.ac.il

problems are reduced within the recursion. Many such algorithms have been recently discovered [3, 23]. Most fast algorithms share a common bilinear structure and can be compactly represented by three matrices that we denote by $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$, following the notation of [4]. Many key properties of the practicality of an algorithm, including its numerical stability, can be derived quickly from its $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ representation. We also note that, because recursive subproblems are again matrix multiplications, different recursive rules can be combined arbitrarily. Following the terminology of [2], we refer to algorithms that vary recursive rules across different recursive levels and within each level as *non-uniform, non-stationary* algorithms. If an algorithm uses the same rule for every subproblem in a each recursive level but varies the rule across levels, we call it a *uniform, non-stationary* algorithm; those defined by only one rule are called *uniform, stationary* algorithms.

Fast matrix multiplication is known to yield larger numerical errors than the classical algorithm. The forward error guarantee for the classical algorithm is component-wise: the error bound for each entry in the output matrix depends only on the dot product between the corresponding row and column of the input matrices. Fast algorithms perform computations involving other input matrix entries that do not appear in a given dot product (their contributions eventually cancel out), and therefore the error bounds for these algorithms depend on more global properties of the input matrices. Thus, fast algorithms with no modification are known to exhibit "norm-wise stability" [4] while the classical algorithm exhibits the stronger "component-wise stability", which is unattainable for fast algorithms [22]. (Often this distinction is misunderstood as a conclusion that Strassen's and other fast algorithms are completely unstable, which is not true.)

Our main goals in this paper are to explore means for improving the theoretical error bounds of fast matrix multiplication algorithms as well as to test the improvements with numerical experiments, focusing particularly on those algorithms that yield performance benefits in practice. For computing $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$, where $\mathbf{A}$ is $M \times K$ and $\mathbf{B}$ is $K \times N$, norm-wise stability bounds for full recursion take the following form:

$$\|\hat{\mathbf{C}} - \mathbf{C}\| \leq f_{\text{alg}}(K)\|\mathbf{A}\|\|\mathbf{B}\|\epsilon + O(\epsilon^2), \tag{1}$$

where $\|\cdot\|$ is the max-norm, ϵ is the machine precision, and f_{alg} is a polynomial function that depends on the algorithm [4, 11, 15]. For example, $f_{\text{alg}}(K) = K^2$ for the classical algorithm, with no assumption on the ordering of dot product computations. We note that f_{alg} is independent of the input matrices, and $\|\mathbf{A}\|\|\mathbf{B}\|$ is independent of the algorithm. In this paper, we explore ways of improving each factor separately. Our main contributions include:

1. generalizing and tightening previous error analysis of uniform, stationary fast algorithms to bound f_{alg} in terms of the number of recursive steps used and two principal quantities derived from $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$;
2. presenting and comparing the stability quantities of recently discovered practical algorithms;
3. exploring means of improving algorithmic stability through algorithm selection and non-uniform, non-stationary combination of algorithms;
4. presenting diagonal scaling techniques to improve accuracy for inputs with entries of widely varying magnitudes; and
5. showing empirical results of the effects of the various improvement techniques on both error and performance.

The structure of the remainder of the paper is as follows. We describe related work in Section 2 and introduce our notation for fast matrix multiplication algorithms

in Section 3. Section 4 presents the error analysis for bounding f_{alg} for general fast algorithms, and Section 5 discusses the implications of the bounds on known practical algorithms. We present diagonal scaling techniques in Section 6, showing how to reduce the contribution of the input matrices to the error bound.

2. Related Work. Bini and Lotti [4] provide the first general error bound for fast matrix multiplication algorithms, and their analysis provides the basis for our results. Demmel et al. [11] generalize Bini and Lotti's results and show that all fast algorithms are stable. A more complete summary of the numerical stability of fast algorithms, with a detailed discussion of Strassen's algorithm along with Winograd's variant, appears in [15, Chapter 23]. We discuss these previous works in more detail and compare them to our error bounds in Section 4.

Castrapel and Gustafson [7] and D'Alberto [8] discuss means of improving the numerical stability of Strassen's algorithm (and Winograd's variant) using the flexibility of non-uniform, non-stationary algorithms. Castrapel and Gustafson propose general approaches to such algorithms, and D'Alberto provides a particular improvement in the case of two levels of recursion.

Smirnov [23] describes strategies for discovering practical fast algorithms and presents several new algorithms, including a rank-23 algorithm for $\langle 3,3,3\rangle$ with the fewest known nonzeros and an algorithm for $\langle 6,3,3\rangle$ that yields a better exponent than Strassen's. Similar techniques are used by Benson and Ballard [3], and they demonstrate performance improvements over the classical and Strassen's algorithm for both single-threaded and shared-memory multi-threaded implementations. Laderman et al. [19] and later Kaporin [17, 18] consider another form of practical algorithms, ones that can achieve fewer floating point operations than the Strassen-Winograd variant for certain matrix dimensions. Kaporin demonstrates better numerical stability than Strassen-Winograd and shows comparable performance. However, because the base case dimensions proposed are relatively large (*e.g.*, 13 or 20), we suspect that the performance will not be competitive on today's hardware. Further, because the $[\![\mathbf{U},\mathbf{V},\mathbf{W}]\!]$ representations are not readily available, we do not consider these types of algorithms in this work.

Dumitrescu [12] proposes a form of diagonal scaling to improve the error bounds for Strassen's algorithm. We refer to his approach as *outside scaling* and discuss it in more detail in Section 6. Higham [15] points out that *inside scaling* can also affect the error bound but does not propose a technique for improving it. Demmel et al. [10] and Ballard et al. [1] state (without proof) improved error bounds using either inside or outside diagonal scaling, and similar techniques are referenced in [21].

3. Fast Matrix Multiplication Algorithms.

3.1. Base Case Algorithms. A bilinear non-commutative algorithm that computes a product of an $M_0 \times K_0$ matrix and a $K_0 \times N_0$ matrix ($\mathbf{C} = \mathbf{AB}$) using R non-scalar (active) multiplications is determined by a $M_0K_0 \times R$ matrix $\mathbf{U}$, a $K_0N_0 \times R$ matrix $\mathbf{V}$, and a $M_0N_0 \times R$ matrix $\mathbf{W}$ such that

$$(2)\quad c_k = \sum_{r=1}^{R} w_{kr} m_r, \quad \text{where} \quad m_r := s_r \cdot t_r, \quad s_r := \sum_{i=1}^{M_0K_0} u_{ir} a_i, \quad t_r := \sum_{j=1}^{K_0N_0} v_{jr} b_j,$$

for $k = 1, \ldots, M_0N_0$. Here, the single indices of entries of $\mathbf{A}$ and $\mathbf{B}$ assume column-major order, the single indices of entries of $\mathbf{C}$ assume row-major order, and $(\cdot)$ signifies an *active* multiplication. We will refer to the dimensions of such an algorithm with

the notation $\langle M_0, K_0, N_0\rangle$, the rank of the algorithm by R, and the set of coefficients that determine the algorithm with the notation $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$.

3.2. Stationary Algorithms. Now we consider multiplying an $M \times K$ matrix $\mathbf{A}$ by a $K \times N$ matrix $\mathbf{B}$. We will assume that M, K, and N are powers of M_0, K_0, and N_0; otherwise, we can always pad the matrices with zeros and the same analysis will hold. The fast algorithm proceeds recursively by first partitioning $\mathbf{A}$ into $M_0 \times K_0$ submatrices of size $(M/M_0) \times (K/K_0)$ and $\mathbf{B}$ into $K_0 \times N_0$ submatrices of size $(K/K_0) \times (N/N_0)$ and then following (2) by matrix blocks, *i.e.*,

$$\mathbf{C}_k = \sum_{r=1}^{R} w_{kr}\mathbf{M}_r, \text{ where } \mathbf{M}_r := \mathbf{S}_r \cdot \mathbf{T}_r, \ \mathbf{S}_r := \sum_{i=1}^{M_0K_0} u_{ir}\mathbf{A}_i, \ \mathbf{T}_r := \sum_{j=1}^{K_0N_0} v_{jr}\mathbf{B}_j \tag{3}$$

for $k = 1, \ldots, M_0N_0$, where $(\cdot)$ signifies a recursive call to the algorithm. Here, we are using single subscripts on matrices as an index for the column- or row-major ordering of the matrix blocks. The algorithms in this class of fast matrix multiplication are called *stationary algorithms* because they use the same algorithm at each recursive step. However, we do not assume that stationary algorithms recurse all the way to a base case of dimension 1; we assume only that the base case computation (of whatever dimension) is performed using the classical algorithm. Thus, a stationary algorithm is defined by the triplet of matrices $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ along with a number of recursive levels L used before switching to the classical algorithm.

3.3. Uniform, Non-Stationary Algorithms. In contrast to the stationary algorithms discussed above, *uniform, non-stationary algorithms* employ a different fast algorithm, in the sense of (2) and (3), at each recursive level. The fast algorithm is the same at a given recursive level. Specifically, we will consider uniform, non-stationary algorithms with L steps of recursion, so the algorithm is specified by matrices $\mathbf{U}^{[l]}$, $\mathbf{V}^{[l]}$, $\mathbf{W}^{[l]}$ of dimensions $M_0^{[l]}K_0^{[l]} \times R^{[l]}$, $K_0^{[l]}N_0^{[l]} \times R^{[l]}$, $M_0^{[l]}N_0^{[l]} \times R^{[l]}$, for $l = 1, \ldots, L$.

Uniform, non-stationary algorithms are of particular interest for maximizing performance. The fastest algorithm for a particular triplet of dimensions M, K, and N may depend on many factors; the same algorithm may not be optimal for the recursive subproblems of smaller dimensions. Assuming performance is fixed for a given triplet of dimensions, the flexibility of non-stationary algorithms allows for performance optimization over a given set of fast algorithms. However, in parallel and more heterogeneous settings, better performance may be obtained by the greater generality of non-uniform, non-stationary algorithms, described in the next section.

3.4. Non-Uniform, Non-Stationary Algorithms. The final class of matrix multiplication algorithms we consider are *non-uniform, non-stationary algorithms*. In contrast to the previous case, non-uniform, non-stationary algorithms use different algorithms within a single recursive level as well across recursive levels, though we restrict the dimension of the partition by the fast algorithms at a given recursive level to be the same. To define such algorithms, we must specify $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ for every node in the recursion tree, a total of $1 + R^{[1]} + R^{[1]}R^{[2]} + \cdots + \prod_{l=1}^{L-1} R^{[l]}$ recursive rules. We use superscript notation $[l, r_1, r_2, \ldots, r_{l-1}]$ to denote a recursive node at level l, in the top-level subtree r_1, second level subtree r_2, and so on.

We demonstrate in Subsection 4.5 that the flexibility of these algorithms allows for an improvement in the numerical stability of multi-level recursive algorithms. We suspect that they also provide a performance benefit over uniform, stationary algorithm, though this has never been demonstrated empirically.

4. Error Analysis. The work by Bini and Lotti [4] provides the basic framework for the forward error analysis of fast matrix multiplication algorithms. They provide general bounds for any square, stationary bilinear algorithm with power-of-two coefficients (so that there is no error in scalar multiplications), assuming that full recursion is used (a base case of dimension 1). Demmel *et al.* [11] extend the work of Bini and Lotti by (1) accounting for errors induced by the scalar multiplications in bilinear algorithms, (2) analyzing uniform, non-stationary bilinear fast matrix multiplication algorithms (algorithms that use different fast matrix multiplication routines at different levels of recursion), and (3) analyzing group-theoretic fast matrix multiplication algorithms. The bounds provided by Demmel *et al.* also assume square algorithms and that full recursion is used. Higham [15] provides bounds for Strassen's original algorithm as well as Winograd's variant in terms of the base case dimension n_0, where the recursion switches to the classical algorithm. Higham's bounds are also slightly tighter (in the case of Strassen's and Winograd's algorithms) than the general bounds previously mentioned. We note that any matrix multiplication algorithm satisfying the component-wise error bound must perform at least n^3 arithmetic operations [22]. This work also shows that we cannot get the same component-wise error bounds even when using just one step of recursion.

The goal of the error analysis provided in this section is to generalize the previous work in two main directions and to tighten the analysis particularly in the case that nonzeros of $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ are not all ± 1. First, we will consider rectangular fast algorithms; that is, instead of considering recursive rules for multiplying two $k \times k$ matrices, we consider the more general set of rules for multiplying an $m \times k$ matrix by a $k \times n$ matrix. Second, we will state our general bounds in terms of the number of levels of recursion used. Motivated by the results of recently discovered practical algorithms [3, 23], we would like to understand the theoretical error guarantees of an algorithm in terms of its $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ representation. The recent performance results show that rectangular algorithms have practical value (even for multiplying square matrices) and that, for performance reasons, typically only a small number of recursive steps are used in practice. Several recently discovered practical algorithms include fractional power-of-two coefficients (e.g., 1/2, 1/4, 1/8), and we expect that other currently undiscovered useful algorithms will include fractional coefficients that are not powers of two. Therefore, we make no assumptions on the entries of $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$, and we derive principal quantities below that can be tighter than the analogous quantities in the previous works by Bini and Lotti [4] and Demmel *et al.* [11], particularly in the case of fractional coefficients. This sometimes leads to much sharper error bounds (see Example 4). We warn the reader that there are notational similarities and (sometimes subtle) inconsistencies with previous work, as a result of our tightening of the analysis.

4.1. Principal quantities. Following the approach of Bini and Lotti [4], we identify two principal quantities associated with a fast algorithm that, along with the dimensions of the algorithm and the number of levels of recursion, determine its theoretical error bounds. These two quantities can be easily computed from the $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ representation, and we define them in terms of the following vectors:

$$\alpha_r := \sum_{i=1}^{M_0 K_0} \mathbb{I}(u_{ir} \neq 0) \qquad \beta_r := \sum_{j=1}^{K_0 N_0} \mathbb{I}(v_{jr} \neq 0) \qquad \gamma_k := \sum_{r=1}^{R} \mathbb{I}(w_{kr} \neq 0) \tag{4}$$

$$a_r := \sum_{i=1}^{M_0K_0} |u_{ir}| \qquad b_r := \sum_{j=1}^{K_0N_0} |v_{jr}| \tag{5}$$

for $r = 1, \ldots, R$ and $k = 1, \ldots, M_0N_0$, where $\mathbb{I}$ is the Boolean-valued indicator function with value 1 for true and 0 for false. That is, $\boldsymbol{\alpha}$ is the vector of numbers of nonzeros in the columns of $\mathbf{U}$, $\boldsymbol{\beta}$ is the vector of numbers of nonzeros in the columns of $\mathbf{V}$, $\boldsymbol{\gamma}$ is the vector of numbers of nonzeros in the rows of $\mathbf{W}$, $\mathbf{a}$ is the vector of column 1-norms of $\mathbf{U}$, and $\mathbf{b}$ is the vector of column 1-norms of $\mathbf{V}$. When $\mathbf{U}$ and $\mathbf{V}$ have ± 1 entries, $\boldsymbol{\alpha} = \mathbf{a}$ and $\boldsymbol{\beta} = \mathbf{b}$.

DEFINITION 1. *The* prefactor vector $\mathbf{q}$ *is defined entry-wise by*

$$q_k = \gamma_k + \max_r (\alpha_r + \beta_r)\mathbb{I}(w_{kr} \neq 0) \tag{6}$$

for $k = 1, \ldots, M_0N_0$, *and the* prefactor Q *is defined as*

$$Q = \max_k q_k.$$

DEFINITION 2. *The* stability vector $\mathbf{e}$ *is defined entry-wise by*

$$e_k = \sum_{r=1}^{R} a_r \cdot b_r \cdot |w_{kr}| \tag{7}$$

for $k = 1, \ldots, M_0N_0$, *and the* stability factor E *is defined as*

$$E = \max_k e_k.$$

The principal quantities for several fast algorithms are listed in Table 1.

Bini and Lotti [4] provide a definition of $\mathbf{q}$ for two different summation algorithms: sequential summation and serialized divide-and-conquer (see Subsection 4.2). We choose the looser of these two bounds (sequential summation) for generality and simpler notation. However, our results are easily converted to the tighter case. Demmel *et al.* use the serialized divide-and-conquer algorithm in their analysis. Bini and Lotti's analysis does not account for "non-active" multiplication by elements of $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$, so their E parameter depends only on the non-zero structure, rather than the magnitude of the elements in these matrices (cf. (5) and Definition 2). Demmel *et al.* do account for the multiplication by elements of $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$. However, their E parameter is identical to that of Bini and Lotti, and their bound includes an additional factor of $(\|U\|\|V\|\|W\|)^L$, where L is the number of recursive levels and $\|\cdot\|$ is the max-norm.

4.2. Model of arithmetic and notation. We follow the notation of Demmel *et al.* [11]. Let $\Theta = \{\theta \mid |\theta| < \epsilon\}$ be the set of all errors bounded by ϵ (machine precision) and let $\Delta = \{1 + \theta \mid \theta \in \Theta\}$. We assume the standard model of rounded arithmetic where the computed value of $op(a, b)$ is $op(a, b)(1+\theta)$ for some $\theta \in \Theta$. We use the set operation notation: $A+B := \{a+b \mid a \in A,\ b \in B\}$, $A-B := \{a-b \mid a \in A,\ b \in B\}$, and $A \cdot B := \{a \cdot b \mid a \in A,\ b \in B\}$.

We define $A^j = A \cdot A \cdot \ldots \cdot A$ and note that $\Delta^j \subset \Delta^{j+1}$ as $1 \in \Delta$. Furthermore, we will not distinguish between singleton sets and an element when using this notation, *e.g.*, $op(a, b)(1+\theta) \in op(a, b)\Delta$. Finally, we will use the standard hat or $fl(\cdot)$ notation to denote a computed value, *e.g.*, $\hat{\mathbf{C}}$ or $fl(op(a, b)) \in op(a, b)\Delta$.

Under this arithmetic, the following fact for summation will be useful in our analysis

$$fl\left(\sum_{i=1}^{N} fl(c_i \cdot a_i)\right) \in \left(\sum_{i=1}^{N} c_i \cdot a_i\right) \Delta^N, \tag{8}$$

where the algorithm for summation is simply to accumulate the terms a_i one at a time, in sequential order. By using a serialized divide-and-conquer summation, we can also achieve

$$fl\left(\sum_{i=1}^{N} fl(c_i \cdot a_i)\right) \in \left(\sum_{i=1}^{N} c_i \cdot a_i\right) \Delta^{1+\lceil \log_2 N \rceil}. \tag{9}$$

For generality, we will assume the more pessimistic bound in (8). Our results can easily be modified for the error bounds in (9).

We will also use the following property:

$$fl\left(\sum_{i=1}^{N} c_i \Delta^{a_j}\right) \in \left(\sum_{i=1}^{N} c_i\right) \Delta^{N+\max_j a_j}. \tag{10}$$

4.3. Forward error analysis of stationary algorithms. The following theorem states the forward error bound for a stationary algorithm in terms of the principal quantities Q and E defined in Subsection 4.1, which can be readily determined from its $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ representation. The sources of error are floating point error accumulation and possible growth in magnitude of intermediate quantities. The floating point error accumulation depends in part on Q and grows at worst linearly in L. The growth of intermediate quantities depends on E and grows exponentially in L, which typically dominates the bound.

THEOREM 3. *Suppose that* $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$*, where* $\mathbf{A} \in \mathbb{R}^{M \times K}$ *and* $\mathbf{B} \in \mathbb{R}^{K \times N}$ *is computed by using* L *recursive steps of the fast matrix multiplication in (3), with the classical* $O(n^3)$ *algorithm used to multiply the* $(M/M_0^L) \times (K/K_0^L)$ *matrices by the* $(K/K_0^L) \times (N/N_0^L)$ *matrices at the base cases of the recursion. Then the computed matrix* $\hat{\mathbf{C}}$ *satisfies*

$$\|\hat{\mathbf{C}} - \mathbf{C}\| \leq \left(K/K_0^L + Q \cdot L\right)(K/K_0^L) \cdot E^L \|\mathbf{A}\|\|\mathbf{B}\|\epsilon + O(\epsilon^2),$$

where $\|\cdot\|$ *is the max-norm.*

Proof. We begin by analyzing how relative errors propagate as we form the $\mathbf{S}$ and $\mathbf{T}$ matrices. Let a superscript index in brackets denote a matrix formed at the specified level of recursion. Following (8), we have the following error at the first recursive level:

$$\hat{\mathbf{S}}_r^{[1]} \in \sum_{i=1}^{M_0 K_0} u_{ir} \mathbf{A}_i \Delta^{\alpha_r}, \quad \hat{\mathbf{T}}_r^{[1]} \in \sum_{j=1}^{K_0 N_0} v_{jr} \mathbf{B}_j \Delta^{\beta_r},$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are defined in (4).

This error propagates as we recurse. At the lth level of recursion, the inputs to the fast algorithm are given as sums of matrices $\mathbf{A}_\phi$ and $\mathbf{B}_\psi$, each with a possible error of Δ^a and Δ^b, respectively, for some index sets ϕ and ψ and some integers a

and b. Following (3) and (8), the algorithm simply accumulates an additional factor of $\Delta^{\alpha_{r_l}}$ and $\Delta^{\beta_{r_l}}$ before the matrices are passed to the subsequent level of recursion. Thus, at the Lth level of recursion, we have

$$\hat{\mathbf{S}}_r^{[L]} \in \mathbf{S}_r^{[L]} \Delta^{\alpha_{r_1}+\cdots+\alpha_{r_L}}, \quad \hat{\mathbf{T}}_r^{[L]} \in \mathbf{T}_r^{[L]} \Delta^{\beta_{r_1}+\cdots+\beta_{r_L}}, \tag{11}$$

with $r = r_1 + (r_2-1)R + \cdots + (r_L-1)R^{L-1}$. Note that in exact arithmetic,

$$\mathbf{S}_r^{[L]} = \sum_{i=1}^{M_0^L K_0^L} u_{i_1 r_1} \cdots u_{i_L r_L} \mathbf{A}_i, \quad \mathbf{T}_r^{[L]} = \sum_{j=1}^{K_0^L N_0^L} v_{j_1 r_1} \cdots v_{j_L r_L} \mathbf{B}_j, \tag{12}$$

where $i = i_1 + (i_2-1)M_0K_0 + \cdots + (i_L-1)(M_0K_0)^{L-1}$ and $j = i_1 + (j_2-1)K_0N_0 + \cdots + (j_L-1)(K_0N_0)^{L-1}$ represent recursive orderings of the subblocks of $\mathbf{A}$ and $\mathbf{B}$.

We now use the classical algorithm to multiply the computed $\mathbf{S}^{[L]}$ and $\mathbf{T}^{[L]}$ matrices at the leaves of the recursion. Because the inner dimension of each leaf-level matrix multiplication is K/K_0^L, from (8) and (11) we accumulate another factor of Δ^{K/K_0^L} to obtain

$$\hat{\mathbf{M}}_r^{[L]} \in \mathbf{S}_r^{[L]} \mathbf{T}_r^{[L]} \Delta^{\chi_r + K/K_0^L},$$

where $\chi_r = \alpha_{r_1} + \beta_{r_1} + \cdots + \alpha_{r_L} + \beta_{r_L}$ for $1 \le r \le R^L$.

Next, the computed matrices $\mathbf{M}^{[L]}$ are added to form $\mathbf{C}$ following(3). At the lth level of recursion, sums of matrices $\mathbf{M}_\phi^{[L]}$, for appropriate index sets ϕ and including accumulated error Δ^a for some integers a, are added together to form the intermediate computed quantities $\mathbf{M}^{[l]}$. In the final step at the top of the recursion tree, we have

$$\hat{\mathbf{C}}_k \in \sum_{r=1}^{R} w_{kr} \hat{\mathbf{M}}_r^{[1]} \Delta^{\gamma_k},$$

where $\boldsymbol{\gamma}$ is as defined in (4). Following (10), if $\hat{\mathbf{M}}_r^{[1]} \in \mathbf{M}_r^{[1]} \Delta^{x_r}$ for some integers x_r, then

$$\hat{\mathbf{C}}_k \in \sum_{r=1}^{R} w_{kr} \mathbf{M}_r^{[1]} \Delta^{\gamma_k + \max_r x_r \cdot \mathbb{I}(w_{kr} \neq 0)}.$$

Likewise, a factor of $\Delta^{\gamma_{k_l}}$ is accumulated at every recursive step, and the contributed error from the $\mathbf{M}^{[L]}$ matrices comes from the leaf (that is involved in the summation) with maximum error. Leaf matrix $\mathbf{M}_r^{[L]}$ is involved in the summation for $\mathbf{C}_k$ if $w_{k_1 r_1} \cdots w_{k_L r_L} \neq 0$, where $r = r_1 + (r_2-1)R + \cdots (r_L-1)R^{L-1}$ and $k = k_1 + (k_2-1)M_0N_0 + \cdots + (k_L-1)(M_0N_0)^{L-1}$. Thus, we have

$$\hat{\mathbf{C}}_k \in \sum_{r=1}^{R^L} w_{k_1 r_1} \cdots w_{k_1 r_L} \mathbf{M}_r^{[L]} \Delta^{\mu_k + \max_r \chi_r \cdot \mathbb{I}(w_{k_1 r_1} \cdots w_{k_L r_L} \neq 0) + K/K_0^L},$$

where $\mu_k = \gamma_{k_1} + \cdots + \gamma_{k_L}$.

Let $\delta_k = \mu_k + \max_r \chi_r \cdot \mathbb{I}(w_{k_1 r_1} \cdots w_{k_L r_L} \neq 0) + K/K_0^L$. In order to determine

the largest accumulated error, we compute the maximum over all output blocks $\mathbf{C}_k$:

$$\begin{aligned}\max_k \delta_k &= K/K_0^L + \max_{k_1,\ldots,k_L}\left\{\mu_k + \max_{r_1,\ldots,r_L}\chi_r \cdot \mathbb{I}(w_{k_1r_1}\cdots w_{k_Lr_L}\neq 0)\right\}\\ &= K/K_0^L + \max_{k_1}\left\{\gamma_{k_1} + \max_{r_1}(\alpha_{r_1}+\beta_{r_1})\mathbb{I}(w_{k_1r_1}\neq 0)\right\} + \cdots\\ &\qquad + \max_{k_L}\left\{\gamma_{k_L} + \max_{r_L}(\alpha_{r_L}+\beta_{r_L})\mathbb{I}(w_{k_Lr_L}\neq 0)\right\}\\ &= K/K_0^L + \max_{k_1}\left\{\gamma_{k_1} + \max_{r_1}(\alpha_{r_1}+\beta_{r_1})\mathbb{I}(w_{k_1r_1}\neq 0)\right\}\cdot L = K/K_0^L + Q\cdot L,\end{aligned}$$

where Q is given in Definition 1.

We now compute the forward error bound for each block of the output matrix. We have $\mathbf{E}_k = \mathbf{C}_k - \hat{\mathbf{C}}_k \in \sum_r w_{k_1r_1}\cdots w_{k_1r_L}\mathbf{M}_r^{[L]}\Theta^{\delta_k}$, which implies (using (12))

$$\begin{aligned}|\mathbf{E}_k| &\leq \sum_{r=1}^{R^L}\left|w_{k_1r_1}\cdots w_{k_1r_L}\mathbf{S}_r^{[L]}\mathbf{T}_r^{[L]}\right|\delta_k\epsilon + O(\epsilon^2)\\ &\leq \sum_{r=1}^{R^L}|w_{k_1r_1}\cdots w_{k_1r_L}|\sum_{i=1}^{M_0^LK_0^L}|u_{i_1r_1}\cdots u_{i_Lr_L}||\mathbf{A}_i|\sum_{j=1}^{K_0^LN_0^L}|v_{j_1r_1}\cdots v_{j_Lr_L}||\mathbf{B}_j|\delta_k\epsilon\\ &\qquad + O(\epsilon^2)\\ &\leq \sum_{r=1}^{R^L}|w_{k_1r_1}\cdots w_{k_1r_L}|\sum_{i=1}^{M_0^LK_0^L}|u_{i_1r_1}\cdots u_{i_Lr_L}|\sum_{j=1}^{K_0^LN_0^L}|v_{j_1r_1}\cdots v_{j_Lr_L}|\cdot\\ &\qquad (K/K_0^L)\|\mathbf{A}\|\|\mathbf{B}\|\delta_k\epsilon + O(\epsilon^2).\end{aligned}$$

Let $\xi_k = \sum_r|w_{k_1r_1}\cdots w_{k_1r_L}|\sum_i|u_{i_1r_1}\cdots u_{i_Lr_L}|\sum_j|v_{j_1r_1}\cdots v_{j_Lr_L}|$. In order to determine the largest intermediate quantity, we compute the maximum over all output blocks $\mathbf{C}_k$:

$$\begin{aligned}\max_k \xi_k &= \max_{k_1,\ldots,k_L}\sum_{r_1,\ldots,r_L}|w_{k_1r_1}\cdots w_{k_Lr_L}|\sum_{i_1,\ldots,i_L}|u_{i_1r_1}\cdots u_{i_Lr_L}|\sum_{j_1,\ldots,j_L}|v_{j_1r_1}\cdots v_{j_Lr_L}|\\ &= \left(\max_{k_1}\sum_{r_1}|w_{k_1r_1}|\sum_{i_1}|u_{i_1r_1}|\sum_{j_1}|v_{j_1r_1}|\right)\cdots\\ &\qquad \left(\max_{k_L}\sum_{r_L}|w_{k_Lr_L}|\sum_{i_L}|u_{i_Lr_L}|\sum_{j_L}|v_{j_Lr_L}|\right)\\ &= \left(\max_{k_1}\sum_{r_1}|w_{k_1r_1}|\sum_{i_1}|u_{i_1r_1}|\sum_{j_1}|v_{j_1r_1}|\right)^L = E^L,\end{aligned}$$

where E is given in Definition 2.

Computing $\max_k|\mathbf{E}_k|$ by maximizing over δ_k and ξ_k separately, we obtain our result. We note that the two quantities may not achieve their maxima for the same k, but we ignore the possible looseness as the overall bound will typically be dominated by the value of E. ∎

Note that if $L = \log_{K_0} K$ (full recursion), the bound in Theorem 3 becomes

$$\|\hat{\mathbf{C}} - \mathbf{C}\| \leq (1 + Q \cdot L) \cdot E^{\log_{K_0} K} \|\mathbf{A}\|\|\mathbf{B}\|\epsilon + O(\epsilon^2)$$

which is the bound provided by Demmel *et al.* [11], assuming $M_0 = K_0 = N_0$, $M = K = N$, all nonzeros of $\mathbf{U}$ have the same value, all nonzeros of $\mathbf{V}$ have the same value, and all nonzeros of $\mathbf{W}$ have the same value. If $L = 0$ (no recursion), we get the familiar bound

$$\|\hat{\mathbf{C}} - \mathbf{C}\| \leq K^2 \|\mathbf{A}\|\|\mathbf{B}\|\epsilon + O(\epsilon^2).$$

EXAMPLE 4. Because our definition of E (Definition 2) accounts for the magnitude of the entries $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ *in situ*, the bound from Theorem 3 can be tighter than previous analyses [4, 11] when $\mathbf{U}$, $\mathbf{V}$, or $\mathbf{W}$ has entries outside of $\{-1, 0, 1\}$. As an example, we consider a $\langle 4, 4, 2\rangle$ algorithm, where the $\mathbf{U}$ and $\mathbf{W}$ matrices have entries in $\{-0.5, 0.5\}$ [3] (see Appendix C). For this algorithm, E according to Definition 2 is 89, while E according to previous work is 125.

4.4. Forward error analysis of uniform, non-stationary algorithms. Recall that uniform, non-stationary algorithms use a single algorithm at each recursive level. We denote the prefactor vector, stability vector, and partition dimensions of algorithm $\left[\!\left[\mathbf{U}^{[l]}, \mathbf{V}^{[l]}, \mathbf{W}^{[l]}\right]\!\right]$ at level l by $\mathbf{q}^{[l]}$, $\mathbf{e}^{[l]}$, and $M_0^{[l]}$, $K_0^{[l]}$, and $N_0^{[l]}$. Using a similar analysis to Subsection 4.3, we get the following stability bound for this class of algorithms:

THEOREM 5. *Suppose that $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ is computed by a uniform, non-stationary algorithm with L recursive steps of fast matrix multiplication, with the fast algorithm $\left[\!\left[\mathbf{U}^{[l]}, \mathbf{V}^{[l]}, \mathbf{W}^{[l]}\right]\!\right]$ used at level l and the classical algorithm used to multiply the matrices at the base case of the recursion. Then the computed matrix $\hat{\mathbf{C}}$ satisfies*

$$\|\hat{\mathbf{C}} - \mathbf{C}\| \leq \left(\frac{K}{\prod_{l=1}^{L} K_0^{[l]}} + \sum_{l=1}^{L} Q^{[l]}\right)\left(\frac{K}{\prod_{l=1}^{L} K_0^{[l]}}\right) \cdot \left(\prod_{l=1}^{L} E^{[l]}\right) \|\mathbf{A}\|\|\mathbf{B}\|\epsilon + O(\epsilon^2).$$

Proof. The proof is similar to the proof of Theorem 3. The largest accumulation error δ now satisfies

$$\begin{aligned}\max_k \delta_k = \; & \frac{K}{\prod_{l=1}^{L} K_0^{[l]}} + \max_{k_1}\left\{\gamma_{k_1}^{[1]} + \max_{r_1}(\alpha_{r_1}^{[1]} + \beta_{r_1}^{[1]})\mathbb{I}(w_{k_1 r_1}^{[1]} \neq 0)\right\} + \cdots \\ & + \max_{k_L}\left\{\gamma_{k_L}^{[L]} + \max_{r_L}(\alpha_{r_L}^{[L]} + \beta_{r_L}^{[L]})\mathbb{I}(w_{k_L r_L}^{[L]} \neq 0)\right\} = \frac{K}{\prod_{l=1}^{L} K_0^{[l]}} + \sum_{l=1}^{L} Q^{[l]},\end{aligned}$$

and the largest intermediate growth quantity ξ satisfies

$$\begin{aligned}\max_k \xi_k = \; & \left(\max_{k_1} \sum_{r_1=1}^{R^{[1]}} |w_{k_1 r_1}^{[1]}| \sum_{i_1=1}^{M_0^{[1]} K_0^{[1]}} |u_{i_1 r_1}^{[1]}| \sum_{j_1=1}^{K_0^{[1]} N_0^{[1]}} |v_{j_1 r_1}^{[1]}|\right) \cdots \\ & \left(\max_{k_L} \sum_{r_L=1}^{R^{[L]}} |w_{k_L r_L}^{[L]}| \sum_{i_L=1}^{M_0^{[L]} K_0^{[L]}} |u_{i_L r_L}^{[L]}| \sum_{j_L=1}^{K_0^{[L]} N_0^{[L]}} |v_{j_L r_L}^{[L]}|\right) = \prod_{l=1}^{L} E^{[l]}.\end{aligned}$$

∎

4.5. Forward error analysis of non-uniform, non-stationary algorithms. We now consider non-stationary algorithms where the algorithm may be non-uniform at every given recursive level of fast matrix multiplication. That is, at any node in the recursion tree, we may choose a different fast algorithm. For simplicity, we assume that at level l in the recursion tree, all algorithms have the same partitioning scheme and rank (so that the $\left[\!\left[\mathbf{U}^{[l,r_1,\ldots,r_{l-1}]}, \mathbf{V}^{[l,r_1,\ldots,r_{l-1}]}, \mathbf{W}^{[l,r_1,\ldots,r_{l-1}]}\right]\!\right]$ representations have the same dimensions across all values $r_1, \ldots, r_{l-1}$) and that after L levels of recursion, all leaf nodes use the classical algorithm.

In the case of stationary algorithms, one $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ defines the entire algorithm; in the case of uniform non-stationary algorithms, L choices of $\left[\!\left[\mathbf{U}^{[l]}, \mathbf{V}^{[l]}, \mathbf{W}^{[l]}\right]\!\right]$ define the entire algorithm; in this case, we have much more flexibility and can choose $1 + R^{[1]} + R^{[1]}R^{[2]} + \cdots + \Pi_{l=1}^{L-1} R^{[l]}$ different fast algorithms (the number of internal nodes of the recursion tree). Recall that we use the notation $[l, r_1, r_2, \ldots, r_{l-1}]$ as a superscript to refer to the algorithm used at level l in the recursion tree, where r_1 defines subtree membership at level 1, r_2 defines subtree membership at level 2, and so on, and r_{l-1} defines the subtree node at the lth level.

Our analysis of these algorithms is fundamentally the same—we bound the accumulated error (δ) and then bound the number of terms (ξ). However, maximizing over all output blocks is not as straightforward and cannot be simplified as cleanly as in the previous cases. In particular, we define the largest accumulation error δ recursively as $\max_k \delta_k^{[1]}$, where

$$
\begin{aligned}
\delta_k^{[1]} &= \frac{K}{\prod_{l=1}^{L} K_0^{[l]}} + \gamma_{k_1}^{[1]} + \max_{r_1} \delta_k^{[2,r_1]} \cdot \mathbb{I}(w_{k_1 r_1}^{[1]} \neq 0), \\
\delta_k^{[2,r_1]} &= \gamma_{k_2}^{[2,r_1]} + \max_{r_2} \delta_k^{[3,r_1,r_2]} \cdot \mathbb{I}(w_{k_2 r_2}^{[2,r_1]} \neq 0), \\
&\vdots \\
\delta_k^{[l,r_1,\ldots,r_{l-1}]} &= \gamma_{k_l}^{[l,r_1,\ldots,r_{l-1}]} + \max_{r_l} \delta_k^{[l+1,r_1,\ldots,r_l]} \cdot \mathbb{I}(w_{k_l r_l}^{[l,r_1,\ldots,r_{l-1}]} \neq 0), \\
&\vdots \\
\delta_k^{[L,r_1,\ldots,r_{L-1}]} &= \gamma_{k_L}^{[L,r_1,\ldots,r_{L-1}]} + \max_{r_L} \chi_r \cdot \mathbb{I}(w_{k_L r_L}^{[L,r_1,\ldots,r_{L-1}]} \neq 0), \text{ and} \\
\chi_r &= \alpha_{r_1}^{[1]} + \beta_{r_1}^{[1]} + \alpha_{r_2}^{[2,r_1]} + \beta_{r_2}^{[2,r_1]} + \cdots + \alpha_{r_L}^{[L,r_1,\ldots,r_{L-1}]} + \beta_{r_L}^{[L,r_1,\ldots,r_{L-1}]}.
\end{aligned}
$$

This expression does not simplify as before. Note that for block k of the output matrix, node $(r_1, \ldots, r_{l-1})$ at level l of the recursion tree accumulates error for the additions/subtractions required by matrix $\mathbf{W}^{[l,r_1,\ldots,r_{l-1}]}$ at that node plus the maximum accumulated error from any of the combined terms. The expression for χ_r reflects the number of additions and subtractions required to produce the factor matrices $\mathbf{S}_r^{[L]}$ and $\mathbf{T}_r^{[L]}$ at the leaf nodes, and the error accumulated during the classical matrix multiplications is included in the definition of $\delta_k^{[1]}$.

Likewise, the largest intermediate growth quantity ξ is $\max_k \xi_k$, where

$$
\begin{aligned}
\xi_k = \sum_{r_1,\ldots,r_L} &\left| w_{k_1 r_1}^{[1]} w_{k_2 r_2}^{[2,r_1]} \cdots w_{k_L r_L}^{[L,r_1,\ldots,r_{L-1}]} \right| \\
&\cdot \sum_{i_1,\ldots,i_L} \left| u_{i_1 r_1}^{[1]} u_{i_2 r_2}^{[2,r_1]} \cdots u_{i_L r_L}^{[L,r_1,\ldots,r_{L-1}]} \right| \cdot \sum_{j_1,\ldots,j_L} \left| v_{j_1 r_1}^{[1]} v_{j_2 r_2}^{[2,r_1]} \cdots v_{j_L r_L}^{[L,r_1,\ldots,r_{L-1}]} \right|,
\end{aligned}
$$

which we can simplify to

$$\xi_k = \sum_{r_1} \left| w^{[1]}_{k_1 r_1} \right| a^{[1]}_{r_1} b^{[1]}_{r_1} \cdot \sum_{r_2} \left| w^{[2,r_1]}_{k_2 r_2} \right| a^{[2,r_1]}_{r_2} b^{[2,r_1]}_{r_2} \cdots \sum_{r_L} \left| w^{[L,r_1,\ldots,r_{L-1}]}_{k_L r_L} \right| a^{[L,r_1,\ldots,r_{L-1}]}_{r_L} b^{[L,r_1,\ldots,r_{L-1}]}_{r_L},$$

where $\mathbf{a}$ and $\mathbf{b}$ vectors are defined as in (5). Note that we cannot simplify further as in the uniform case.

Example 6. D'Alberto [8] describes a non-uniform, non-stationary approach using Strassen's algorithm that obtains a smaller stability factor than the original uniform, stationary algorithm (for $L \geq 2$). Strassen's algorithm, with $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ as given in (43) of Appendix A, has stability vector $e = \begin{bmatrix} 12 & 4 & 4 & 12 \end{bmatrix}$; two levels of recursion with a uniform, stationary approach yields a two-level stability vector of $e \otimes e$ with maximum entry $12^2 = 144$. D'Alberto shows that, for $L = 2$, a stability factor of 96 can be obtained with a non-uniform approach using one variant of Strassen's algorithm. One way to achieve this stability factor is to use the alternative algorithm

$$\left[\!\!\left[\tilde{\mathbf{U}}, \tilde{\mathbf{V}}, \tilde{\mathbf{W}} \right]\!\!\right] = \left[\!\!\left[\mathbf{U}, \left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \mathbf{V}, \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right) \mathbf{W} \right]\!\!\right]$$

for nodes $[2,1]$, $[2,3]$, and $[2,4]$ of the recursion tree, while using the original algorithm at nodes $[1]$, $[2,2]$, $[2,5]$, $[2,6]$, and $[2,7]$. Similar improvements can be made based on the Strassen-Winograd algorithm, which has a slightly larger stability factor.

A more generic non-uniform approach is described in a patent by Castrapel and Gustafson [7]. They consider eight variants of the Strassen-Winograd algorithm, defined by

$$\left[\!\!\left[\left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^x \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^y \right) \mathbf{U}, \left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^z \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^x \right) \mathbf{V}, \left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^y \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^z \right) \mathbf{W} \right]\!\!\right]$$

with $x, y, z \in \{1, 2\}$. The correctness of these variants can be derived from [16, Equation (6)]. Castrapel and Gustafson suggest using random, round-robin, or matrix-dependent selections of algorithms to more evenly distribute the error, but they do not prove that any particular techniques will reduce the stability factor.

Example 7. We can improve the two-level stability factor for the $\langle 3, 2, 3 \rangle$ case in a similar manner. The smallest stability factor we have discovered for this case is $E = 20$, given by the $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ in Appendix B, which has stability vector $e = \begin{bmatrix} 20 & 20 & 2 & 12 & 4 & 20 & 4 & 12 & 20 \end{bmatrix}$. Compared to a uniform two-level stability factor of $20^2 = 400$, we can achieve a stability factor of 352 using 3 variants of the algorithm. We use the original algorithm at nodes $[1]$, $[2,2]$, $[2,6]$, $[2,8]$, $[2,14]$, and $[2,15]$, the variant

$$\left[\!\!\left[\left(I_2 \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \right) \mathbf{U}, \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes I_2 \right) \mathbf{V}, \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \right) \mathbf{W} \right]\!\!\right]$$

at nodes $[2,1]$, $[2,3]$, $[2,10]$, and $[2,11]$, and the variant

$$\left[\!\!\left[\left(I_2 \otimes \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \right) \mathbf{U}, \left(\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \otimes I_2 \right) \mathbf{V}, \left(\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \right) \mathbf{W} \right]\!\!\right]$$

at nodes $[2, 4]$, $[2, 5]$, $[2, 7]$, $[2, 9]$, $[2, 12]$, and $[2, 13]$. We suspect that better two-level stability factors are achievable.

5. Algorithm selection. Theorem 3 immediately provides several options for improving the numerical stability of fast matrix multiplication:

1. We can look for algorithms with a smaller Q and E. Since prior work on finding fast algorithms focuses on performance, this provides a new dimension for algorithm design. This is the focus of Subsection 5.1.
2. We can reduce the number of recursive levels before using standard matrix multiplication at the base case. The tradeoff is that fewer recursive levels means an asymptotically slower algorithm. We examine this in Subsection 5.2.
3. We can reduce $\|A\|$ and $\|B\|$ by pre-processing and post-processing the data. We provide several such strategies in Section 6.

5.1. Searching for better algorithms. Typically, the only quantity of interest for finding fast matrix multiplication algorithms is the *rank* of the solution. However, we can also search for algorithms to minimize the Q and E quantities while maintaining the same rank. This will improve the numerical stability of the algorithm without sacrificing (asymptotic) performance. We will also consider the number of non-zeros (nnz) in the solution, *i.e.*, the sum of the number of non-zero entries in $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$, as this affects the constant in the asymptotic complexity and has noticeable impact on empirical performance [3]. Thus, the parameters of interest for these algorithms is a performance-stability 3-tuple (nnz, Q, E). In general, the number of non-zeros is positively correlated with Q and E, since these quantities directly depend on the non-zero patterns of $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ (see (6) and (7)).

We first looked at the base case $\langle 4, 2, 3\rangle$, which has out-performed Strassen's algorithm in practice [3]. We found 479 algorithms with rank $R = 20$ using numerical low-rank tensor decomposition search techniques [3]. Of these, there were 208 performance-stability tuples. The smallest nnz, Q, and E quantities over all algorithms were 130, 12, and 32, and the corresponding algorithms had performance-stability tuples (130, 14, 34), (138, 12, 34), and (134, 13, 32) (no algorithm has parameters that achieved more than one of these minima). Subsequently, there is a theoretical trade-off between performance and stability. We note that although this list of algorithms is not exhaustive, they are the only publicly available $\langle 4, 2, 3\rangle$ algorithms.[1]

We tested the stability of these algorithms by computing the product of samples of random matrices $\mathbf{A} \in \mathbb{R}^{4096\times 2048}$ and $\mathbf{B} \in \mathbb{R}^{2048\times 3645}$. The distributions were a_{ij}, $b_{ij} \sim$ Uniform(0, 1) and a_{ij}, $b_{ij} \sim$ Uniform(-1, 1). In addition to the three "good" algorithms described above, we also compared against an algorithm with a much worse performance-stability tuple of (156, 26, 132). For each pair of matrices, we ran the four algorithms with number of recursive levels $L = 1, 2, \ldots, 6$. To estimate $\|\hat{\mathbf{C}} - \mathbf{C}\|$, we computed $\mathbf{C}$ using the classical algorithm in quadruple precision arithmetic. All other computations used double precision arithmetic. Figure 1 summarizes the results and includes the upper bound on the error from Theorem 3. We see the following results:

1. The error bounds are still pessimistic, even with the improved analysis from Theorem 3. Furthermore, the error bounds for the three good $\langle 4, 2, 3\rangle$ algorithms are quite similar.

[1] All of our algorithms, as well as the software for finding them, is publicly available at `https://github.com/arbenson/fast-matmul`.

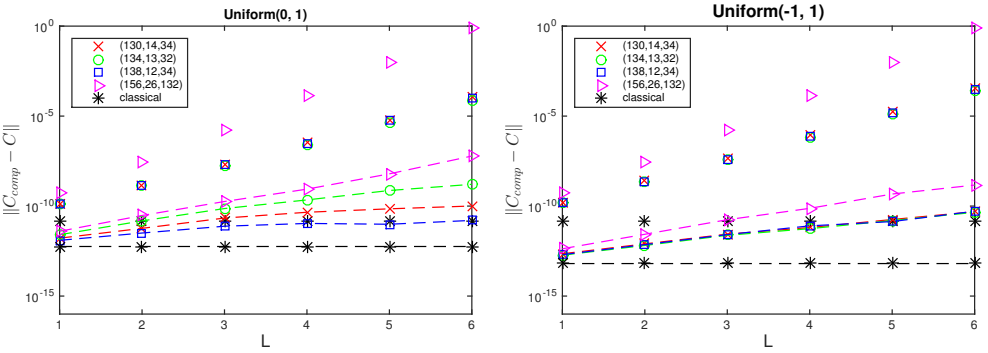

Fig. 1: Error for four $\langle 4, 2, 3\rangle$ fast matrix multiplication algorithms with different stability parameters and the classical algorithm as a function of the number of recursive levels, L. Three algorithms are "good" in the sense that they minimize the number of non-zeros, Q, or E. The dashed curves are the experimental error, and the corresponding markers are the upper bounds from Theorem 3. The experimental error increases with L, as modeled by Theorem 3. The good algorithms with minimal nnz, Q, and E all have similar performance, but the fast algorithm with a worse performance-stability tuple is noticeably less stable.

2. The true error increases with the number of recursive levels, as predicted by Theorem 3 and modeled by the error bound.
3. The difference between the "good" algorithms depends on the matrices, but the other $\langle 4, 2, 3\rangle$ algorithm is noticeably worse in both cases.

We also considered the $\langle 2, 3, 2\rangle$ base case, which has optimal rank $R = 11$ [5]. One known algorithm that achieves the optimal rank uses Strassen's algorithm on a 2×2 sub-block and classical matrix multiplication on the remaining sub-blocks. The base case of the algorithm is small enough so that we could use a SAT solver [9] to find over 10,000 rank 11 $\langle 2, 3, 2\rangle$ algorithms (ignoring symmetries). We found that the combination of Strassen's algorithm with the classical algorithm had a strictly smaller performance-stability triple than all of the other rank 11 solutions. We conclude that this algorithm is likely optimal in a performance *and* stability sense for the class of $\langle 2, 3, 2\rangle$ algorithms where the scalar multiplications are ± 1.

5.2. Performance and stability trade-offs with a small number of recursive levels. We now consider the performance and stability of fast matrix multiplication algorithms across several base cases and several values of L. Table 1 summarizes the best known (to us) stability factors (E) for several practical base case dimensions. The columns of the table represent the relevant performance and stability parameters for each algorithm, all of which can be computed from the $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ representation.

The rank R and the number of nonzeros (nnz), along with the number of recursive levels used, determine the number of floating point operations performed by the uniform, stationary version of the algorithm. The rank can be compared to the product $M_0K_0N_0$, the rank of the classical algorithm for that base case. The quantities Q and E are computed using Definitions 1 and 2, respectively; for a given base case we report the algorithm with the best known E along with that algorithm's Q. We do not report both $\langle M_0, K_0, N_0\rangle$ and $\langle N_0, K_0, M_0\rangle$ because the best algorithms for each have identical nnz, E, and Q parameters, due to transformations corresponding to

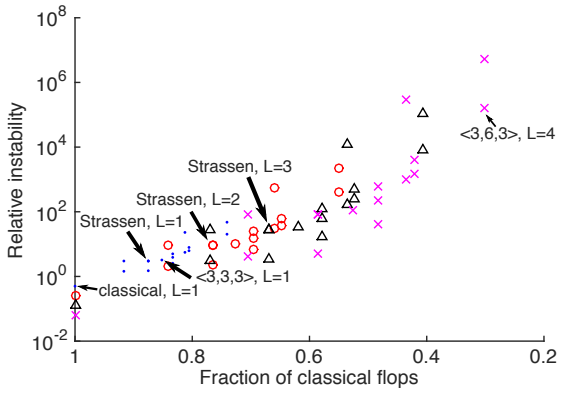

Fig. 2: Distribution of relative stability factor and percentage of classical flops for the algorithms in Table 1 with $L = 1$ (blue '.'), 2 (red 'o'), 3 (black triangle), and 4 (magenta 'x'). There is a general log-linear trade-off between stability and number of floating point operations.

transposition of the matrix multiplication.

Although we stress that these algorithms will be used with only a few levels of recursion, we also report the asymptotic stability exponent (stab. exp.) in order to compare algorithms across different base case dimensions. If an algorithm for a square base case $\langle N_0, N_0, N_0\rangle$ is used on square matrices of dimension N down to subproblems of constant dimension, the bound of Theorem 3 can be simplified to

$$\|\hat{\mathbf{C}} - \mathbf{C}\| \leq c \cdot N^{\log_{N_0} E} \log N \|\mathbf{A}\|\|\mathbf{B}\|\epsilon + O(\epsilon^2),$$

where c is a constant that depends in part on Q. In this case, the stability exponent is $\log_{N_0} E$. We note that the first two rows of Table 1 match the results of [4, Table 2]. The most stable rank-23 $\langle 3, 3, 3\rangle$ algorithm of which we are aware is a cyclic rotation of the one given in [23]. In the case of rectangular base cases $\langle M_0, K_0, N_0\rangle$, we assume a uniform, non-stationary algorithm based on cyclic use of algorithms for $\langle M_0, K_0, N_0\rangle$, $\langle N_0, M_0, K_0\rangle$, and $\langle K_0, N_0, M_0\rangle$, where the three recursive rules are transformations of each other, either by cyclic rotations or transposition (for more details, see Appendix B and Appendix C).

Figure 2 shows the distribution of relative stability and percentage of classical flops for the algorithms in Table 1, for $L = 1, 2, 3, 4$. We measure both terms asymptotically. Ignoring the quadratic cost of additions, the percentage of classical flops is $(R/(M_0K_0N_0))^L$. For large matrix dimension and L small, we can ignore Q by Theorem 3, and the relative stability factor is $(E/K_0^2)^L$ (the division by two is relative to the smallest possible K_0). In general, most algorithms follows a narrow log-linear trade-off between these two parameters. However, there is still room to select algorithms for a fixed number of recursion levels. For example, with $L = 1$, the $\langle 3, 3, 3\rangle$ algorithm has roughly the same stability and does fewer floating point operations than Strassen's algorithm.

6. Scaling. We now turn our attention to strategies for pre- and post-processing matrices in order to improve numerical stability. The error bounds from Section 4

Table 1: Principal quantities for a variety of fast matrix multiplication algorithms. The rank of the algorithm (R) drives the asymptotic complexity, and the total number of non-zeroes in the $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ (nnz) affects the constant in the complexity. Likewise, the E parameter drives the asymptotic behavior of the stability bound and the Q parameter affects the constant. The stability exponent (stab. exp.) denotes the asymptotic stability of the algorithm assuming square matrix multiplication.

$\langle M_0, K_0, N_0\rangle$	ref.	$M_0K_0N_0$	R	nnz	Q	E	stab. exp.
$\langle 2,2,2\rangle$	(classical)	8	8	24	4	2	1
$\langle 2,2,2\rangle$	[24]	8	7	36	8	12	3.58
$\langle 3,2,2\rangle$	[24]	12	11	48	8	12	3.03
$\langle 2,3,2\rangle$	[24]	12	11	48	9	13	3.03
$\langle 4,2,2\rangle$	[24]	16	14	72	8	12	2.94
$\langle 2,4,2\rangle$	[24]	16	14	72	12	24	2.94
$\langle 3,2,3\rangle$	Appendix B	18	15	94	10	20	3.21
$\langle 3,3,2\rangle$	Appendix B	18	15	94	11	23	3.21
$\langle 3,3,3\rangle$	[23]	27	23	139	15	29	3.07
$\langle 4,2,3\rangle$	[3]	24	20	130	14	34	3.38
$\langle 3,4,2\rangle$	[3]	24	20	130	14	30	3.38
$\langle 2,3,4\rangle$	[3]	24	20	130	14	35	3.38
$\langle 4,4,2\rangle$	Appendix C	32	26	257	22	89	3.90
$\langle 4,2,4\rangle$	Appendix C	32	26	257	23	92	3.93
$\langle 3,4,3\rangle$	[3]	36	29	234	23	100	3.66
$\langle 3,3,4\rangle$	[3]	36	29	234	18	71	3.66
$\langle 3,3,6\rangle$	[23]	54	40	960	39	428	4.69
$\langle 3,6,3\rangle$	[23]	54	40	960	48	728.5	4.69

can be summarized by the following relative error bound

$$\frac{|c_{ij} - \hat{c}_{ij}|}{|c_{ij}|} \leq f_{\text{alg}}(K)\frac{\|A\|\|B\|}{|c_{ij}|}\epsilon + O(\epsilon^2). \tag{13}$$

Recall that f_{alg} is the (at worst) polynomial function of the inner dimension that depends on the particular algorithm used. Unfortunately, these bounds can often be quite large when $|c_{ij}|$ is small relative to $\|A\|\|B\|$. For the remainder of this section, we will ignore f_{alg} and consider it a fixed quantity, so that $f_{\text{alg}}(K)\epsilon = O(\epsilon)$.

Example 8. Consider the matrices

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \ \mathbf{B} = \begin{bmatrix} z & 1 \\ z & 1 \end{bmatrix}, \ \mathbf{C} = \mathbf{A}\cdot\mathbf{B} = \begin{bmatrix} 2z & 2 \\ 2z & 2 \end{bmatrix}, \tag{14}$$

for small $z > 0$. By (13), we have the following relative error bound

$$\frac{|c_{11} - \hat{c}_{11}|}{|c_{11}|} \leq O\left(\frac{\|\mathbf{A}\|\|\mathbf{B}\|\epsilon}{|c_{11}|}\right) = O\left(\epsilon/z\right), \tag{15}$$

which can be quite large for small z. Furthermore, this bound is actually achieved with Strassen's algorithm (see Appendix A for the definition of Strassen's algorithm).

Specifically, Strassen's algorithm computes

$$m_1 = (a_{11} + a_{22})(b_{11} + b_{22}) = (1 + 1) \cdot (z + 1) \tag{16a}$$
$$m_4 = a_{22}(b_{21} - b_{11}) = 1 \cdot (z - 1) \tag{16b}$$
$$m_5 = (a_{11} + a_{12})b_{22} = (1 + 1) \cdot 1 \tag{16c}$$
$$m_7 = (a_{12} - a_{22})(b_{21} + b_{22}) = (1 - 1) \cdot (z + 1) \tag{16d}$$
$$c_{11} = m_1 + m_4 - m_5 + m_7. \tag{16e}$$

There are terms of size $O(1)$ in computing m_1, m_4, m_5, and m_7, so the absolute error $|c_{11} - \hat{c}_{11}|$ is $O(\epsilon)$. Since $c_{11} = z$, the relative error is $O(\epsilon/z)$.

We now demonstrate several methods for improving numerical stability issues by pre-processing $\mathbf{A}$ and $\mathbf{B}$ and post-processing $\mathbf{C}$. The idea underlying these methods is the following straightforward observation

$$\mathbf{C} = \mathbf{D}_A \mathbf{D}_A^{-1} \mathbf{A} \mathbf{D} \mathbf{D}^{-1} \mathbf{B} \mathbf{D}_B^{-1} \mathbf{D}_B, \tag{17}$$

for any nonsingular scaling matrices $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}$. By taking advantage of the associativity of matrix multiplication (in exact arithmetic) and scaling matrices $\mathbf{D}_A$, $\mathbf{D}_B$, and $\mathbf{D}$ that are easy to apply, we can improve the norm-wise bound in Theorem 3 without significantly affecting the performance of the algorithm.

For the algorithms and analysis in this section, we will consider *diagonal* scaling matrices with positive diagonal entries. In order to simplify the analysis, we will assume that there is no numerical error in applying the scaling matrices. This could be achieved, for example, by rounding the scaling matrix entries to the nearest power of two. Regardless, the error introduced by the fast matrix multiplication algorithm has the larger impact on the stability, and the scaling matrices can curb numerical inaccuracies.

6.1. Outside scaling. In light of (17), Dumistrescu proposed the following "outside scaling" matrices [12]:

$$\mathbf{D}_A = \operatorname{diag}\left(\max_j |a_{ij}|\right), \quad \mathbf{D}_B = \operatorname{diag}\left(\max_i |b_{ij}|\right). \tag{18}$$

The resulting procedure is Algorithm 1.

Algorithm 1 Outside scaling for fast matrix multiplication

Require: matrices $\mathbf{A}$ and $\mathbf{B}$
Ensure: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
1: $\mathbf{D}_A \leftarrow \operatorname{diag}(\max_j |a_{ij}|)$
2: $\mathbf{A}' \leftarrow \mathbf{D}_A^{-1}\mathbf{A}$
3: $\mathbf{D}_B \leftarrow \operatorname{diag}(\max_i |b_{ij}|)$
4: $\mathbf{B}' \leftarrow \mathbf{B}\mathbf{D}_B^{-1}$
5: $\mathbf{C}' \leftarrow \mathbf{A} \cdot \mathbf{B}$ with fast matrix multiplication.
6: $\mathbf{C} \leftarrow \mathbf{D}_A \mathbf{C}' \mathbf{D}_B$

Clearly, the algorithm correctly computes $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ in exact arithmetic, provided there are no all-zero rows in $\mathbf{A}$ or all-zero columns in $\mathbf{B}$. Importantly, the norm-wise bound in Theorem 3 applies to the *scaled* matrices $\mathbf{A}'$ and $\mathbf{B}'$. In particular, we get the following improved bound [12].

PROPOSITION 9. *Using Algorithm 1,*

$$|c_{ij} - \hat{c}_{ij}| \leq O(\epsilon)\|a_{i,:}\|\|b_{:,j}\|. \tag{19}$$

Proof. Scaling ensures that $\|\mathbf{A}'\|, \|\mathbf{B}'\| \leq 1$, so by (13), $\|\mathbf{C}' - \hat{\mathbf{C}}'\| \leq O(\epsilon)$. Since $\mathbf{C}' - \hat{\mathbf{C}}' = \mathbf{D}_A(\hat{\mathbf{C}} - \mathbf{C})\mathbf{D}_B$, the result follows from the fact that the ith diagonal entry of $\mathbf{D}_A$ is $\|a_{i,:}\|$ and jth diagonal entry of $\mathbf{D}_B$ is $\|b_{:,j}\|$. ∎

For the matrices in Example 8, the bound from Proposition 9 improves upon (15):

$$\frac{|c_{11} - \hat{c}_{11}|}{|c_{11}|} \leq O\left(\frac{\|a_{1,:}\|\|b_{:,1}\|\epsilon}{|c_{11}|}\right) = O(\epsilon) \tag{20}$$

This indeed improves the numerical stability of Strassen's algorithm. For the matrices in (14), the outside scaling is

$$\mathbf{A}' \leftarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \ \mathbf{B}' \leftarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \ \mathbf{C}' = \mathbf{A}' \cdot \mathbf{B}' = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}.$$

And when computing $\mathbf{C}'$ with Strassen's algorithm,

$$m_1 = (a'_{11} + a'_{22})(b'_{11} + b'_{22}) = (1+1)\cdot(1+1) \tag{21a}$$
$$m_4 = a'_{22}(b'_{21} - b'_{11}) = 1\cdot(1-1) \tag{21b}$$
$$m_5 = (a'_{11} + a'_{12})b'_{22} = (1+1)\cdot 1 \tag{21c}$$
$$m_7 = (a'_{12} - a'_{22})(b'_{21} + b'_{22}) = (1-1)\cdot(1+1) \tag{21d}$$
$$c'_{11} = m_1 + m_4 - m_5 + m_7. \tag{21e}$$

Now, all sub-terms are on the order of unity, so the relative error in computing c'_{11} is $O(\epsilon)$.

6.2. Inside scaling. There are several pairs of matrices where outside scaling is not sufficient for numerical stability.

EXAMPLE 10. Consider the matrices

$$\mathbf{A} = \begin{bmatrix} 1 & z \\ 1 & z \end{bmatrix}, \ \mathbf{B} = \begin{bmatrix} z & z \\ 1 & 1 \end{bmatrix}, \ \mathbf{C} = \mathbf{A}\cdot\mathbf{B} = \begin{bmatrix} 2z & 2z \\ 2z & 2z \end{bmatrix}. \tag{22}$$

Using outside scaling on these matrices does nothing since $\mathbf{D}_A = \mathbf{D}_B = \mathbf{I}$. However, using a fast algorithm can still have severe numerical stability issues. Computing c_{12} with Strassen's algorithm uses the following computations:

$$m_3 = a_{11}(b_{12} - b_{22}) = 1\cdot(z-1) \tag{23a}$$
$$m_5 = (a_{11} + a_{12})b_{22} = (1+z)\cdot 1 \tag{23b}$$
$$c_{12} = m_3 + m_5. \tag{23c}$$

The computation of m_3 and m_5 has terms of unit size, so $|c_{12} - \hat{c}_{12}|$ is $O(\epsilon)$ and the relative error is $O(\epsilon/z)$. This is reflected in the bound from (13):

$$\|\mathbf{A}\|\|\mathbf{B}\|/|c_{12}| = 1/(2z). \tag{24}$$

We now propose a new scaling technique called *inside scaling* to augment the outside scaling:

$$\mathbf{D} = \operatorname{diag}\left(\sqrt{\frac{\max_j |b_{kj}|}{\max_i |a_{ik}|}}\right) \tag{25}$$

The resulting procedure is in Algorithm 2. The idea is to scale the columns of $\mathbf{A}$ and the corresponding rows of $\mathbf{B}$ to have the same norm. In general, we get an improved error bound, as detailed in Proposition 11.

Algorithm 2 Inside scaling for fast matrix multiplication

Require: matrices $\mathbf{A}$ and $\mathbf{B}$
Ensure: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
1: $\mathbf{D} \leftarrow \operatorname{diag}\left(\sqrt{\frac{\max_j |b_{kj}|}{\max_i |a_{ik}|}}\right)$
2: $\mathbf{A}' \leftarrow \mathbf{AD}$
3: $\mathbf{B}' \leftarrow \mathbf{D}^{-1}\mathbf{B}$
4: $\mathbf{C} \leftarrow \mathbf{A}' \cdot \mathbf{B}'$ with fast matrix multiplication.

PROPOSITION 11. *Using Algorithm 2,*

$$\|\hat{\mathbf{C}} - \mathbf{C}\| \leq O(\epsilon) \max_{i,k,j} |a_{ik}||b_{kj}|. \tag{26}$$

Proof. By (13),

$$\|\hat{\mathbf{C}} - \mathbf{C}\| \leq O(\epsilon)\|\mathbf{AD}\|\|\mathbf{D}^{-1}\mathbf{B}\| = O(\epsilon)\left(\max_k \|a_{:,k}\| d_{kk}\right)\left(\max_l d_{ll}^{-1}\|b_{l,:}\|\right).$$

By the definition of $\mathbf{D}$,

$$\|a_{:,k}\| d_{kk} = d_{kk}^{-1}\|b_{k,:}\| = \sqrt{\|a_{:,k}\|\|b_{k,:}\|},$$

so the two maxima are attained at the same index. The result then follows from the fact that $\|a_{:,k}\|\|b_{k,:}\| = \max_{i,k,j} |a_{ik}||b_{kj}|$. ∎

For the $\mathbf{A}$ and $\mathbf{B}$ in Example 10, $\max_{i,k,j} |a_{ik}||b_{kj}| = z$, and we get an $O(\epsilon)$ relative error bound for computing each entry in $\mathbf{C}$. The inside scaling updates to the matrices in (22) are

$$\mathbf{D} = \begin{bmatrix} \sqrt{z} & 0 \\ 0 & 1/\sqrt{z} \end{bmatrix}, \ \mathbf{A}' \leftarrow \begin{bmatrix} \sqrt{z} & \sqrt{z} \\ \sqrt{z} & \sqrt{z} \end{bmatrix}, \ \mathbf{B}' \leftarrow \begin{bmatrix} \sqrt{z} & \sqrt{z} \\ \sqrt{z} & \sqrt{z} \end{bmatrix}. \tag{27}$$

Strassen's algorithm now computes

$$m_3 = a'_{11}(b'_{12} - b'_{22}) = \sqrt{z} \cdot (\sqrt{z} - \sqrt{z}) \tag{28a}$$
$$m_5 = (a'_{11} + a'_{12})b'_{22} = (\sqrt{z} + \sqrt{z}) \cdot \sqrt{z} \tag{28b}$$
$$c'_{12} = m_3 + m_5. \tag{28c}$$

This time, the computation of m_3 and m_5 involves terms on the order of z instead of on the order of unity, and we get $O(\epsilon)$ relative error in the computation.

Algorithm 3 Outside-inside scaling for fast matrix multiplication

$\mathbf{D}_A \leftarrow \operatorname{diag}(\max_j |a_{ij}|)$
$\mathbf{A}' \leftarrow \mathbf{D}_A^{-1}\mathbf{A}$
$\mathbf{D}_B \leftarrow \operatorname{diag}(\max_i |b_{ij}|)$
$\mathbf{B}' \leftarrow \mathbf{B}\mathbf{D}_B^{-1}$
$\mathbf{D} \leftarrow \operatorname{diag}\left(\sqrt{\frac{\max_j |b'_{kj}|}{\max_i |a'_{ik}|}}\right)$
$\mathbf{A}'' \leftarrow \mathbf{A}'\mathbf{D}$
$\mathbf{B}'' \leftarrow \mathbf{D}^{-1}\mathbf{B}'$
$\mathbf{C}'' \leftarrow \mathbf{A}'' \cdot \mathbf{B}''$ with fast matrix multiplication
$\mathbf{C} \leftarrow \mathbf{D}_A\mathbf{C}''\mathbf{D}_B$

6.3. Outside-inside and inside-outside scaling. We can naturally combine outside and inside scaling by performing one algorithm after the other. Interestingly, the results and improved error bounds depend on the application order of the scaling. We call Algorithm 3, which does outside scaling first, "outside-inside" scaling.

PROPOSITION 12. *Using Algorithm 3,*

$$|c_{ij} - \hat{c}_{ij}| \leq O(\epsilon)\|a_{i,:}\|\|b_{:,j}\| \max_{i,k,j} |a'_{ik}||b'_{kj}|. \tag{29}$$

Proof. By Proposition 11, $|c''_{ij} - \hat{c}''_{ij}| \leq O(\epsilon) \max_{i,k,j} |a'_{ik}||b'_{kj}|$. Since $\mathbf{C}' - \hat{\mathbf{C}}' = \mathbf{D}_A(\hat{\mathbf{C}} - \mathbf{C})\mathbf{D}_B$, the result follows from the fact that the ith diagonal entry of $\mathbf{D}_A$ is $\|a_{i,:}\|$ and jth diagonal entry of $\mathbf{D}_B$ is $\|b_{:,j}\|$. ∎

COROLLARY 13. *The outside-inside scaling error bound from Proposition 12 is stronger than the outside scaling bound from Proposition 9.*

Proof. Outside scaling ensures the rows of $\mathbf{A}'$ and the columns of $\mathbf{B}'$ have unit norm, so $\max_{i,k,j} |a'_{ik}||b'_{kj}| \leq 1$. ∎

We can also do the inside scaling first, resulting in the "inside-outside" scaling described in Algorithm 4.

Algorithm 4 Inside-outside scaling for fast matrix multiplication

$\mathbf{D} \leftarrow \operatorname{diag}\left(\sqrt{\frac{\max_j |b_{kj}|}{\max_i |a_{ik}|}}\right)$
$\mathbf{A}' \leftarrow \mathbf{A}\mathbf{D}$
$\mathbf{B}' \leftarrow \mathbf{D}^{-1}\mathbf{B}$
$\mathbf{D}_A \leftarrow \operatorname{diag}\left(\max_j |a'_{ij}|\right)$
$\mathbf{A}'' \leftarrow \mathbf{D}_A^{-1}\mathbf{A}'$
$\mathbf{D}_B \leftarrow \operatorname{diag}\left(\max_i |b'_{ij}|\right)$
$\mathbf{B}'' \leftarrow \mathbf{B}'\mathbf{D}_B^{-1}$
$\mathbf{C}'' \leftarrow \mathbf{A}'' \cdot \mathbf{B}''$ with fast matrix multiplication
$\mathbf{C} \leftarrow \mathbf{D}_A\mathbf{C}''\mathbf{D}_B$

PROPOSITION 14. *Using Algorithm 4,*

$$|c_{ij} - \hat{c}_{ij}| \leq O(\epsilon)\|a'_{i,:}\|\|b'_{:,j}\|. \tag{30}$$

Proof. We apply outside scaling to the matrices $\mathbf{A}'$ and $\mathbf{B}'$, so the result follows from Proposition 9. ∎

COROLLARY 15. *The inside-outside scaling error bound from Proposition 14 is stronger than the inside scaling bound from Proposition 11.*

Proof.

$$\begin{aligned}
\|a'_{i,:}\|\|b'_{:,j}\| &\le \max_{ij} \|a'_{i,:}\|\|b'_{:,j}\| \\
&= \max_{ik} |a_{i,k}| d_{kk} \max_{lj} |b_{l,j}| d_{ll}^{-1} \\
&= \max_{k} \|a_{:,k}\| \left(\frac{\|b_{k,:}\|}{\|a_{:,k}\|} \right)^{1/2} \max_{l} \|b_{l,:}\| \left(\frac{\|a_{:,l}\|}{\|b_{l,:}\|} \right)^{1/2} \\
&= \max_{k} \|a_{:,k}\|\|b_{k,:}\| = \max_{ikj} |a_{ik}||b_{kj}|.
\end{aligned}$$

∎

We now provide an example that shows that these two methods can achieve vastly different numerical results.

EXAMPLE 16. Consider the matrices

$$\mathbf{A} = \begin{bmatrix} 1 & z^{-1} \\ 1 & 1 \end{bmatrix}, \ \mathbf{B} = \begin{bmatrix} z & 1 \\ z & 1 \end{bmatrix}, \ \mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} 1+z & 1+z^{-1} \\ 2z & 2 \end{bmatrix}. \tag{31}$$

For this matrix, outside-inside scaling computes

$$\mathbf{A}' \leftarrow \begin{bmatrix} z & 1 \\ 1 & 1 \end{bmatrix}, \ \mathbf{B}' \leftarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \ \mathbf{A}'' \leftarrow \begin{bmatrix} z & 1 \\ 1 & 1 \end{bmatrix}, \ \mathbf{B}'' \leftarrow \begin{bmatrix} z^{-1/2} & z^{-1/2} \\ 1 & 1 \end{bmatrix}, \tag{32}$$

and inside-outside scaling computes

$$\begin{aligned}
\mathbf{A}' &\leftarrow \begin{bmatrix} 1 & z^{-1/2} \\ 1 & z^{1/2} \end{bmatrix}, & \mathbf{B}' &\leftarrow \begin{bmatrix} z & 1 \\ z^{1/2} & z^{-1/2} \end{bmatrix}, \\
\mathbf{A}'' &\leftarrow \begin{bmatrix} z^{1/2} & 1 \\ 1 & z^{1/2} \end{bmatrix}, & \mathbf{B}'' &\leftarrow \begin{bmatrix} z & z \\ 1 & 1 \end{bmatrix}.
\end{aligned} \tag{33}$$

Consider the computation of entry $c_{21} = 2z$. By Proposition 12, outside-inside scaling guarantees a relative error of $O(\epsilon)$. However, by Proposition 14, inside-outside scaling only guarantees a relative error of $O(z^{1/2})$. Indeed, if we use inside-outside scaling with Strassen's algorithm to compute c_{21},

$$m_2 = (a''_{21} + a''_{22}) b''_{11} = (1 + z^{1/2}) \cdot z \tag{34a}$$

$$m_4 = a''_{22}(b''_{21} - b''_{11}) = z^{1/2} \cdot (1 - z) \tag{34b}$$

$$c''_{21} = m_2 + m_4, \tag{34c}$$

and we have a sub-term on the order of $z^{1/2}$.

6.4. Repeated inside-outside scaling. Next, we consider repeatedly applying inside and outside scaling in alternating order, as shown in Algorithm 5. We start by analyzing the effect of the algorithm on the accuracy of the computed product. Theorem 18 shows that the elementwise error bound of the computed product is monotonically nonincreasing, meaning that accuracy is increased with each step. We

then analyze how rapidly the error bound converges to its limit. In Theorem 21 we show that convergence is initially quadratic and ultimately linear. Lemma 22 demonstrates an example in which convergence is exactly as guaranteed by Theorem 21, showing that our convergence analysis is sharp in the worst case. Finally, in Theorem 23 we provide a stopping criterion for the iteration. We allow the user to specify a relative-tolerance parameter τ, so that whenever our stopping criterion is satisfied, the error bound is within a relative distance τ from its limit. In particular, choosing $\tau = 1$ implies quadratic convergence throughout the iteration, and it achieves an error bound within a factor of 2 of the optimal bound that alternating scaling can achieve. Testing the stopping criterion in each iteration contributes only a lower-order term to the work and communication costs, and it is easy to implement.

We start with our accuracy analysis. In our analysis, we use $\mathbf{A}^{(t)}$ and $\mathbf{B}^{(t)}$ to denote the values of $\mathbf{A}'$ and $\mathbf{B}'$, respectively, after t steps of Algorithm 5. We also use $r_i^{(t)}$ and $s_j^{(t)}$ to denote the diagonal elements of $\mathbf{D}_A$ and $\mathbf{D}_B$, respectively, after t steps. The initial values of these variables correspond to $t = 0$ in our notation.

Algorithm 5 Alternating scaling

$\mathbf{A}' \leftarrow \mathbf{A}$, $\mathbf{B}' \leftarrow \mathbf{B}$, $\mathbf{D}_A \leftarrow \mathbf{I}$, $\mathbf{D}_B \leftarrow \mathbf{I}$
alternate
  step O
    $\mathbf{D}'_A \leftarrow \mathrm{diag}(\max_k |a'_{ik}|)$
    $\mathbf{D}_A \leftarrow \mathbf{D}_A \mathbf{D}'_A$
    $\mathbf{A}' \leftarrow (\mathbf{D}'_A)^{-1} \mathbf{A}'$
    $\mathbf{D}'_B \leftarrow \mathrm{diag}(\max_k |b'_{kj}|)$
    $\mathbf{D}_B \leftarrow \mathbf{D}'_B \mathbf{D}_B$
    $\mathbf{B}' \leftarrow \mathbf{B}' (\mathbf{D}'_B)^{-1}$
  end
  step I
    $\mathbf{D} \leftarrow \mathrm{diag}\left(\sqrt{\frac{\max_j |b'_{kj}|}{\max_i |a'_{ik}|}}\right)$
    $\mathbf{A}' \leftarrow \mathbf{A}' \mathbf{D}$
    $\mathbf{B}' \leftarrow \mathbf{D}^{-1} \mathbf{B}'$
  end
until converged
$\mathbf{C}' \leftarrow \mathbf{A}' \cdot \mathbf{B}'$ with fast matrix multiplication
$\mathbf{C} \leftarrow \mathbf{D}_A \mathbf{C}' \mathbf{D}_B$

Proposition 17. *Let t be the number of steps of Algorithm 5 that we complete. The computed product satisfies*

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

Proof. The analysis is the same as in the proofs of Propositions 12 and 14. ∎

Theorem 18. *The sequence*

$$r_i^{(t)} s_j^{(t)} \|\mathbf{A}^{(t)}\| \|\mathbf{B}^{(t)}\| \qquad \text{for } t = 0, 1, \ldots$$

is monotonically nonincreasing.

Proof. If step t is an O step, then

$$\|\mathbf{A}^{(t)}\| = \|\mathbf{B}^{(t)}\| = 1\,, \qquad r_i^{(t)} = r_i^{(t-1)}\|a_{i,:}^{(t-1)}\|\,, \qquad s_j^{(t)} = s_j^{(t-1)}\|b_{:,j}^{(t-1)}\|\,,$$

and therefore

$$\begin{aligned} r_i^{(t)} s_j^{(t)} \|\mathbf{A}^{(t)}\|\|\mathbf{B}^{(t)}\| &= r_i^{(t-1)} s_j^{(t-1)} \|a_{i,:}^{(t-1)}\|\|b_{:,j}^{(t-1)}\| \\ &\le r_i^{(t-1)} s_j^{(t-1)} \|\mathbf{A}^{(t-1)}\|\|\mathbf{B}^{(t-1)}\|\,. \end{aligned}$$

Next, assume that step t is an I step. Column k of $\mathbf{A}'$ is transformed so that

$$a_{ik}^{(t)} = \left(\frac{\|b_{k,:}^{(t-1)}\|}{\|a_{:,k}^{(t-1)}\|}\right)^{\frac{1}{2}} a_{ik}^{(t-1)}\,,$$

and therefore

$$\|a_{:,k}^{(t)}\| = \left(\frac{\|b_{k,:}^{(t-1)}\|}{\|a_{:,k}^{(t-1)}\|}\right)^{\frac{1}{2}} \|a_{:,k}^{(t-1)}\| = \sqrt{\|a_{:,k}^{(t-1)}\|\|b_{k,:}^{(t-1)}\|}\,, \tag{35}$$

and similarly

$$\|b_{k,:}^{(t)}\| = \sqrt{\|a_{:,k}^{(t-1)}\|\|b_{k,:}^{(t-1)}\|}\,. \tag{36}$$

Hence,

$$\|\mathbf{A}^{(t)}\| = \|\mathbf{B}^{(t)}\| = \max_k \sqrt{\|a_{:,k}^{(t-1)}\|\|b_{k,:}^{(t-1)}\|}\,, \qquad r_i^{(t)} = r_i^{(t-1)}\,, \qquad s_j^{(t)} = s_j^{(t-1)}\,,$$

and therefore

$$\begin{aligned} r_i^{(t)} s_j^{(t)} \|\mathbf{A}^{(t)}\|\|\mathbf{B}^{(t)}\| &= r_i^{(t-1)} s_j^{(t-1)} \Big(\max_k \sqrt{\|a_{:,k}^{(t-1)}\|\|b_{k,:}^{(t-1)}\|}\Big)^2 \\ &= r_i^{(t-1)} s_j^{(t-1)} \max_k\big(\|a_{:,k}^{(t-1)}\|\|b_{k,:}^{(t-1)}\|\big) \\ &\le r_i^{(t-1)} s_j^{(t-1)} \|\mathbf{A}^{(t-1)}\|\|\mathbf{B}^{(t-1)}\|\,. \end{aligned}$$

∎

Next, we show that the algorithm converges. We use the following notation. The index of the first O step of the iteration is denoted by t_0. Whenever step t is an O step, we use $r_i^{\prime(t)}$ and $s_j^{\prime(t)}$ to denote the diagonal elements of the matrices $\mathbf{D}'_A$ and $\mathbf{D}'_B$, respectively, that we compute in step t. Similarly, if step t is an I step, $p_k^{\prime(t)}$ denotes the diagonal elements of $\mathbf{D}$.

LEMMA 19. *The sequences $r_i^{(t)}$, $s_j^{(t)}$, $\|\mathbf{A}^{(t)}\|$ and $\|\mathbf{B}^{(t)}\|$ for $t = 0, 1, \ldots$ converge.*

Proof. As we show in the proof of Theorem 18,

$$\begin{aligned} \|\mathbf{A}^{(t)}\| = \|\mathbf{B}^{(t)}\| &= 1\,, \\ \|\mathbf{A}^{(t+1)}\| = \|\mathbf{B}^{(t+1)}\| &= \max_k \sqrt{\|a_{:,k}^{(t)}\|\|b_{k,:}^{(t)}\|} \qquad \text{for } t = t_0, t_0+2, \ldots\,. \end{aligned}$$

Therefore

$$\|\mathbf{A}^{(t+1)}\| = \max_k \sqrt{\|a_{:,k}^{(t)}\|\|b_{k,:}^{(t)}\|} \le \sqrt{\|\mathbf{A}^{(t)}\|\|\mathbf{B}^{(t)}\|} = 1\,,$$

and hence

$$r_i'^{(t+2)} = \|a_{i,:}^{(t+1)}\| \le \|\mathbf{A}^{(t+1)}\| \le 1\,. \tag{37}$$

The sequence $r_i^{(t)}$ satisfies

$$\begin{aligned} r_i^{(t_0)} &= r_i^{(t_0+1)} = r_i'^{(t_0)} \\ r_i^{(t_0+2)} &= r_i^{(t_0+3)} = r_i'^{(t_0)} r_i'^{(t_0+2)} \\ &\vdots \qquad\qquad \vdots \end{aligned}$$

It is nonnegative because $r_i'^{(t)} \ge 0$, it is monotonically nonincreasing by (37), and hence it must converge. The same is true for $s_j^{(t)}$.

Consider the effect of the first t steps of the iteration on $\mathbf{A}'$ and $\mathbf{B}'$. The cumulative effect of the O steps is to divide the rows of $\mathbf{A}'$ by $r_i^{(t)}$ and the columns of $\mathbf{B}'$ by $s_j^{(t)}$, and that of the I steps is to make sure that every column of $\mathbf{A}'$ is equal in norm to the corresponding row of $\mathbf{B}'$. Therefore

$$a_{ik}^{(t)} = a_{ik}\,\frac{1}{r_i^{(t)}}\left(\frac{\max_j \left|b_{kj}/s_j^{(t-1)}\right|}{\max_i \left|a_{ik}/r_i^{(t-1)}\right|}\right)^{\frac{1}{2}} \qquad \text{for } t = t_0+1, t_0+2, \dots\,,$$

which shows that the convergence of $r_i^{(t)}$ and $s_j^{(t)}$ guarantees the convergence of $a_{ik}^{(t)}$, and hence also the convergence of $\|\mathbf{A}^{(t)}\|$. The same is true for $\|\mathbf{B}^{(t)}\|$. ∎

The following lemma shows that the intermediate scaling factors that we compute in each step rapidly converge to 1. We use this lemma in our subsequent analysis. We use the notation

$$\begin{aligned} w^{(t)} &= \max\Big(\max_i \left|\log r_i'^{(t)}\right|, \max_j \left|\log s_j'^{(t)}\right|\Big)\,, \\ w^{(t+1)} &= \max_k \left|\log p_k'^{(t+1)}\right| \qquad\qquad \text{for } t = t_0, t_0+2, \dots\,. \end{aligned}$$

Lemma 20. *The following bounds hold:*

$$\begin{aligned} w^{(t)} &\le w^{(t-1)}\,, \\ w^{(t+1)} &\le 0.5 w^{(t)} \qquad\qquad \textit{for } t = t_0+2, t_0+4, \dots\,. \end{aligned}$$

Proof. Assume that $t = t_0+2, t_0+4, \dots$. Because step $t-2$ is an O step, there is a column g so that $\left|a_{ig}^{(t-2)}\right| = 1$, and therefore

$$r_i'^{(t)} = \max_k \left|a_{ik}^{(t-1)}\right| = \max_k \left|a_{ik}^{(t-2)} p_k'^{(t-1)}\right| \ge \left|a_{ig}^{(t-2)} p_g'^{(t-1)}\right| = p_g'^{(t-1)}\,.$$

Taking logarithms yields

$$\log r_i'^{(t)} \ge \log p_g'^{(t-1)}\,.$$

Both sides of this inequality are nonpositive because $r_i'^{(t)} \le 1$ by (37), and so

$$\left|\log r_i'^{(t)}\right| = -\log r_i'^{(t)} \le -\log p_g'^{(t-1)} = \left|\log p_g'^{(t-1)}\right|\,.$$

A similar analysis shows that

$$\left|\log s_j'^{(t)}\right| \leq \left|\log p_f'^{(t-1)}\right|,$$

for a suitably defined row f, and these two inequalities imply the first bound in the statement of the lemma.

Next, let us prove the secound bound. We have that

$$\left(p_k'^{(t+1)}\right)^2 = \frac{\max_j\left|b_{kj}^{(t)}\right|}{\max_i\left|a_{ik}^{(t)}\right|} = \frac{\max_j\left|b_{kj}^{(t-1)}/s_j'^{(t)}\right|}{\max_i\left|a_{ik}^{(t-1)}/r_i'^{(t)}\right|} \leq \frac{\max_j\left|b_{kj}^{(t-1)}\right|\ \max_j\left(1/s_j'^{(t)}\right)}{\max_i\left|a_{ik}^{(t-1)}/r_i'^{(t)}\right|}.$$

Inequality (37) states that $r_i'^{(t)} \leq 1$, and therefore

$$\max_i\left|a_{ik}^{(t-1)}/r_i'^{(t)}\right| \geq \max_i\left|a_{ik}^{(t-1)}\right|,$$

which we substitute into the previous inequality, obtaining

$$\left(p_k'^{(t+1)}\right)^2 \leq \frac{\max_j\left|b_{kj}^{(t-1)}\right|\ \max_j\left(1/s_j'^{(t)}\right)}{\max_i\left|a_{ik}^{(t-1)}\right|}.$$

By (35) and (36)

$$\max_i\left|a_{ik}^{(t-1)}\right| = \|a_{:,k}^{(t-1)}\| = \|b_{k,:}^{(t-1)}\| = \max_j\left|b_{kj}^{(t-1)}\right|,$$

which implies

$$\left(p_k'^{(t+1)}\right)^2 \leq \max_j\left(1/s_j'^{(t)}\right).$$

A similar analysis shows that

$$\left(p_k'^{(t+1)}\right)^2 \geq \frac{1}{\max_i\left(1/r_i'^{(t)}\right)}.$$

Taking the logarithm of these two bounds and interchanging the positions of the logarithms with those of the max operators yields

$$-\max_i\left(\log\left(1/r_i'^{(t)}\right)\right) \leq 2\log p_k'^{(t+1)} \leq \max_j\left(\log\left(1/s_j'^{(t)}\right)\right).$$

Because $r_i'^{(t)} \leq 1$ we have that $\log\left(1/r_i'^{(t)}\right) = \left|\log r_i'^{(t)}\right|$, and similarly for $s_j'^{(t)}$. Applying this to the previous inequality yields

$$-\max_i\left|\log r_i'^{(t)}\right| \leq 2\log p_k'^{(t+1)} \leq \max_j\left|\log s_j'^{(t)}\right|,$$

and therefore

$$2\left|\log p_k'^{(t+1)}\right| \leq \max\left(\max_i\left|\log r_i'^{(t)}\right|, \max_j\left|\log s_j'^{(t)}\right|\right),$$

which implies the second bound in the statement of the lemma. ∎

The following theorem is our characterization of the algorithm's convergence. We provide an interpretation of the result after we prove it. Our notation is as follows. Lemma 19 states that the sequences $r_i^{(t)}$, $s_j^{(t)}$, $\|\mathbf{A}^{(t)}\|$ and $\|\mathbf{B}^{(t)}\|$ converge. We use a superscript $\star$ to denote their limits, so that

$$r_i^{(t)} \to r_i^{(\star)}\,, \qquad s_j^{(t)} \to s_j^{(\star)}\,, \qquad \|\mathbf{A}^{(t)}\| \to \|\mathbf{A}^{(\star)}\|\,, \qquad \|\mathbf{B}^{(t)}\| \to \|\mathbf{B}^{(\star)}\|\,,$$

and we let

$$\mu_{ij}^{(t)} = \frac{\left|r_i^{(t)} s_j^{(t)} \|\mathbf{A}^{(t)}\|\|\mathbf{B}^{(t)}\| - r_i^{(\star)} s_j^{(\star)} \|\mathbf{A}^{(\star)}\|\|\mathbf{B}^{(\star)}\|\right|}{\left|r_i^{(\star)} s_j^{(\star)} \|\mathbf{A}^{(\star)}\|\|\mathbf{B}^{(\star)}\|\right|} = \frac{r_i^{(t)} s_j^{(t)} \|\mathbf{A}^{(t)}\|\|\mathbf{B}^{(t)}\|}{r_i^{(\star)} s_j^{(\star)} \|\mathbf{A}^{(\star)}\|\|\mathbf{B}^{(\star)}\|} - 1$$

be the relative distance of $r_i^{(t)} s_j^{(t)} \|\mathbf{A}^{(t)}\|\|\mathbf{B}^{(t)}\|$ from the limit.

THEOREM 21. *There is a sequence $\nu^{(t)}$ so that*

$$\mu_{ij}^{(t)} \le \nu^{(t)}\,, \qquad \mu_{ij}^{(t+1)} \le \nu^{(t+1)}\,,$$

and

$$\nu^{(t+1)} = \nu^{(t)}\,, \qquad \nu^{(t+2)} \le \frac{1}{2+\nu^{(t+2)}}\,\nu^{(t)} \qquad \text{for } t = t_0, t_0+2, \ldots\,.$$

Proof. Assume that $t = t_0, t_0+2, \ldots$. We have that

$$r_i^{(t)} = r_i'^{(t_0)} r_i'^{(t_0+2)} \cdots r_i'^{(t)}\,, \quad s_j^{(t)} = s_j'^{(t_0)} s_j'^{(t_0+2)} \cdots s_j'^{(t)}\,, \quad \|\mathbf{A}^{(t)}\| = \|\mathbf{B}^{(t)}\| = 1\,,$$

and therefore

$$r_i^{(\star)} = r_i'^{(t_0)} r_i'^{(t_0+2)} \cdots\,, \qquad s_j^{(\star)} = s_j'^{(t_0)} s_j'^{(t_0+2)} \cdots\,, \qquad \|\mathbf{A}^{(\star)}\| = \|\mathbf{B}^{(\star)}\| = 1\,.$$

Substituting this into the above yields

$$\begin{aligned}
\mu_{ij}^{(t)} &= \left(r_i^{(t)} s_j^{(t)} \|\mathbf{A}^{(t)}\|\|\mathbf{B}^{(t)}\|\right)\left(r_i^{(\star)} s_j^{(\star)} \|\mathbf{A}^{(\star)}\|\|\mathbf{B}^{(\star)}\|\right)^{-1} - 1 \\
&= \left(r_i'^{(t_0)} s_j'^{(t_0)} r_i'^{(t_0+2)} s_j'^{(t_0+2)} \cdots r_i'^{(t)} s_j'^{(t)}\right)\left(r_i'^{(t_0)} s_j'^{(t_0)} r_i'^{(t_0+2)} s_j'^{(t_0+2)} \cdots\right)^{-1} - 1 \\
&= \left(r_i'^{(t+2)} s_j'^{(t+2)} r_i'^{(t+4)} s_j'^{(t+4)} \cdots\right)^{-1} - 1\,.
\end{aligned}$$

Applying the definition of $w^{(t)}$ to this, we obtain

$$\begin{aligned}
\mu_{ij}^{(t)} &= \left(r_i'^{(t+2)} s_j'^{(t+2)} r_i'^{(t+4)} s_j'^{(t+4)} \cdots\right)^{-1} - 1 \\
&= \exp\left(-\log r_i'^{(t+2)} - \log s_j'^{(t+2)} - \log r_i'^{(t+4)} - \log s_j'^{(t+4)} - \cdots\right) - 1 \\
&= \exp\left(\left|\log r_i'^{(t+2)}\right| + \left|\log s_j'^{(t+2)}\right| + \left|\log r_i'^{(t+4)}\right| + \left|\log s_j'^{(t+4)}\right| + \cdots\right) - 1 \\
&\le \exp\left(2w^{(t+2)} + 2w^{(t+4)} + \cdots\right) - 1\,.
\end{aligned}$$

We define

$$\nu^{(t)} = \exp\left(2w^{(t+2)} + 2w^{(t+4)} + \cdots\right) - 1\,, \qquad \nu^{(t+1)} = \nu^{(t)}\,,$$

thereby guaranteeing that $\mu_{ij}^{(t)} \le \nu^{(t)}$, as the theorem states. Because $\gamma_{ij}^{(t)}$ is monotonically nonincreasing, so is its relative distance to its limit, meaning that

$$\mu_{ij}^{(t+1)} \le \mu_{ij}^{(t)}$$

and hence

$$\mu_{ij}^{(t+1)} \le \mu_{ij}^{(t)} \le \nu^{(t)} = \nu^{(t+1)} .$$

This proves another condition in the statement of the theorem, leaving us only with the condition $\nu^{(t+2)} \le \left(1/\left(2+\nu^{(t+2)}\right)\right)\nu^{(t)}$ to prove.

By Lemma 20,

$$\begin{aligned} w^{(t+3)} &\le 0.5w^{(t+2)} & w^{(t+4)} &\le w^{(t+3)} \le 0.5w^{(t+2)} \\ w^{(t+5)} &\le 0.5w^{(t+4)} \le 0.25w^{(t+2)} & w^{(t+6)} &\le w^{(t+5)} \le 0.25w^{(t+2)} \\ &\vdots \qquad\quad \vdots & &\vdots \qquad\quad \vdots \end{aligned}$$

Therefore

$$\begin{aligned} w^{(t+2)} &= (0.5 + 0.25 + \cdots)w^{(t+2)} \\ &= 0.5w^{(t+2)} + 0.25w^{(t+2)} + \cdots \\ &\ge w^{(t+4)} + w^{(t+6)} + \cdots , \end{aligned}$$

and thus

$$\exp\left(2w^{(t+2)}\right) \ge \exp\left(2w^{(t+4)} + 2w^{(t+6)} + \cdots\right) .$$

Applying this bound to the definition of $\nu^{(t)}$, we obtain

$$\begin{aligned} \nu^{(t)} &= \exp\left(2w^{(t+2)} + 2w^{(t+4)} + \cdots\right) - 1 \\ &= \exp\left(2w^{(t+2)}\right)\exp\left(2w^{(t+4)} + 2w^{(t+6)} + \cdots\right) - 1 \\ &\ge \left(\exp\left(2w^{(t+4)} + 2w^{(t+6)} + \cdots\right)\right)^2 - 1 . \end{aligned}$$

We define $x = \exp\left(2w^{(t+4)} + 2w^{(t+6)} + \cdots\right)$, so that the above expression has the form $x^2 - 1$ and $\nu^{(t+2)} = x - 1$. The rest of the proof is:

$$\nu^{(t)} \ge x^2 - 1 = (2 + x - 1)(x - 1) = \left(2 + \nu^{(t+2)}\right)\nu^{(t+2)} ,$$

which implies that $\nu^{(t+2)} \le \left(1/\left(2+\nu^{(t+2)}\right)\right)\nu^{(t)}$ as the theorem states. ∎

Let us explain the significance of the above theorem. It shows that the relative distance of the error from its limit is bounded by a sequence $\nu^{(t)}$ that satisfies the condition

$$\nu^{(t+2)} \le \frac{1}{2+\nu^{(t+2)}}\nu^{(t)} .$$

We have that $\nu^{(t+2)} \ge 0$, and so

$$\nu^{(t+2)} \le \frac{1}{2+\nu^{(t+2)}}\nu^{(t)} \le \frac{1}{2}\nu^{(t)} ,$$

and therefore we are guaranteed linear convergence. We are making progress at least at a convergence rate 0.5 with every two steps of the algorithm. Furthermore, multiplying the bound by $\nu^{(t+2)}$ on both sides yields

$$\left(\nu^{(t+2)}\right)^2 \le \frac{\nu^{(t+2)}}{2+\nu^{(t+2)}}\nu^{(t)} .$$

This tells us that convergence is quadratic so long as our relative distance from the limit is greater than 1. The prefactor that corresponds to quadratic convergence is $\nu^{(t+2)}/\big(2+\nu^{(t+2)}\big)$, which is at most 1, and it monotonically decreases towards 1/3 as $\nu^{(t+2)} \to 1$.

The following lemma shows that our convergence analysis is sharp.

LEMMA 22. *There are matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *and indices* i *and* j *so that*

$$\mu_{ij}^{(t+1)} = \mu_{ij}^{(t)}\,, \qquad \mu_{ij}^{(t+2)} = \frac{1}{2+\mu_{ij}^{(t+2)}}\;\mu_{ij}^{(t)} \qquad \text{for } t = t_0, t_0+2, \dots\,.$$

Proof. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 1 & 2^{-2^v} \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

for some integer v, let us start the iteration with an O step, and assume that $t = t_0, t_0+2, \dots$. A straightforward calculation, which we omit for brevity, shows that

$$\begin{aligned}
\mathbf{A}^{(t)} &= \begin{bmatrix} 1 & 0 \\ 1 & 2^{-2^{v-(t-t_0)/2}} \end{bmatrix}, & \begin{bmatrix} r_1^{(t)} \\ r_2^{(t)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \\
\mathbf{B}^{(t)} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & \begin{bmatrix} s_1^{(t)} \\ s_2^{(t)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2^{-2^v(1-2^{-(t-t_0)/2})} \end{bmatrix}, \\
\mathbf{A}^{(t+1)} &= \begin{bmatrix} 1 & 0 \\ 1 & 2^{-2^{v-(t-t_0)/2-1}} \end{bmatrix}, & \begin{bmatrix} r_1^{(t+1)} \\ r_2^{(t+1)} \end{bmatrix} &= \begin{bmatrix} r_1^{(t)} \\ r_2^{(t)} \end{bmatrix}, \\
\mathbf{B}^{(t+1)} &= \begin{bmatrix} 1 & 0 \\ 0 & 2^{-2^{v-(t-t_0)/2-1}} \end{bmatrix}, & \begin{bmatrix} s_1^{(t+1)} \\ s_2^{(t+1)} \end{bmatrix} &= \begin{bmatrix} s_1^{(t)} \\ s_2^{(t)} \end{bmatrix},
\end{aligned}$$

and therefore

$$\mathbf{A}^{(\star)} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad \mathbf{B}^{(\star)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \begin{bmatrix} r_1^{(\star)} \\ r_2^{(\star)} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \qquad \begin{bmatrix} s_1^{(\star)} \\ s_2^{(\star)} \end{bmatrix} = \begin{bmatrix} 1 \\ 2^{-2^v} \end{bmatrix}$$

and

$$\|\mathbf{A}^{(t)}\| = \|\mathbf{B}^{(t)}\| = \|\mathbf{A}^{(t+1)}\| = \|\mathbf{B}^{(t+1)}\| = \|\mathbf{A}^{(\star)}\| = \|\mathbf{B}^{(\star)}\| = 1\,.$$

Substituting the above into the definition of $\mu_{ij}^{(t)}$ yields

$$\begin{aligned}
\mu_{22}^{(t)} = \mu_{22}^{(t+1)} &= \frac{\big|r_2^{(t)} s_2^{(t)} \|\mathbf{A}^{(t)}\|\|\mathbf{B}^{(t)}\| - r_2^{(\star)} s_2^{(\star)} \|\mathbf{A}^{(\star)}\|\|\mathbf{B}^{(\star)}\|\big|}{\big|r_2^{(\star)} s_2^{(\star)} \|\mathbf{A}^{(\star)}\|\|\mathbf{B}^{(\star)}\|\big|} \\
&= \frac{\big|2^{-2^v(1-2^{-(t-t_0)/2})} - 2^{-2^v}\big|}{\big|2^{-2^v}\big|} \\
&= 2^{2^{v-(t-t_0)/2}} - 1\,.
\end{aligned}$$

Letting $x = 2^{2^{v-(t-t_0)/2-1}}$, so that the above expression has the form $x^2 - 1$ and $\mu_{22}^{(t+2)} = x - 1$, we have that

$$\mu_{22}^{(t)} = x^2 - 1 = (2 + x - 1)(x - 1) = \big(2 + \mu_{22}^{(t+2)}\big)\,\mu_{22}^{(t+2)}\,,$$

which proves the lemma. ∎

Next, we derive a stopping criterion for the iteration. We explain how it works after the proof.

THEOREM 23. *Let $\tau > 0$ be a user-specified tolerance parameter. We have that for $t = t_0, t_0 + 2, \dots,$*

$$\max_{i,j} \mu_{ij}^{(t+1)} \le \tau \quad \text{if} \quad \min_k p_k'^{(t+1)} \ge (1+\tau)^{-\frac{1}{4}} \quad \text{and} \quad \max_k p_k'^{(t+1)} \le (1+\tau)^{\frac{1}{4}},$$

and

$$\max_{i,j} \mu_{ij}^{(t+2)} \le \tau \quad \text{if} \quad \min_i r_i'^{(t+2)} \ge (1+\tau)^{-\frac{1}{2}} \quad \text{and} \quad \min_j s_j'^{(t+2)} \ge (1+\tau)^{-\frac{1}{2}}.$$

Proof. In the proof of Theorem 21, we show that for all i, j and $t = t_0, t_0 + 2, \dots,$

$$\begin{aligned} \mu_{ij}^{(t+1)} &\le \mu_{ij}^{(t)}, \\ \mu_{ij}^{(t)} &\le \exp\bigl(2w^{(t+2)} + 2w^{(t+4)} + \cdots\bigr) - 1, \\ \exp\bigl(2w^{(t+4)} + 2w^{(t+6)} + \cdots\bigr) &\le \exp\bigl(2w^{(t+2)}\bigr). \end{aligned}$$

Putting these three statements together yields

$$\begin{aligned} \mu_{ij}^{(t+1)} &\le \mu_{ij}^{(t)} \le \exp\bigl(2w^{(t+2)} + 2w^{(t+4)} + \cdots\bigr) - 1 \\ &= \exp\bigl(2w^{(t+2)}\bigr) \exp\bigl(2w^{(t+4)} + 2w^{(t+6)} + \cdots\bigr) - 1 \\ &\le \exp\bigl(2w^{(t+2)}\bigr) \exp\bigl(2w^{(t+2)}\bigr) - 1 \\ &= \exp\bigl(4w^{(t+2)}\bigr) - 1. \end{aligned}$$

Lemma 20 guarantees that $w^{(t+2)} \le w^{(t+1)}$, and substituting this into the above yields

$$\mu_{ij}^{(t+1)} \le \exp\bigl(4w^{(t+1)}\bigr) - 1. \tag{38}$$

Similarly,

$$\mu_{ij}^{(t+2)} \le \exp\bigl(2w^{(t+4)} + 2w^{(t+6)} + \cdots\bigr) - 1 \le \exp\bigl(2w^{(t+2)}\bigr) - 1. \tag{39}$$

Next, let us prove the first statement of the theorem. Assume that

$$(1+\tau)^{-\frac{1}{4}} \le p_k'^{(t+1)} \le (1+\tau)^{\frac{1}{4}}$$

for all k. Taking logarithms yields

$$-0.25 \log(1+\tau) \le \log p_k'^{(t+1)} \le 0.25 \log(1+\tau),$$

or equivalently

$$\bigl|\log p_k'^{(t+1)}\bigr| \le 0.25 \log(1+\tau),$$

and therefore

$$w^{(t+1)} = \max_k \bigl|\log p_k'^{(t+1)}\bigr| \le 0.25 \log(1+\tau).$$

Substituting this into (38), we find that

$$\mu_{ij}^{(t+1)} \le \exp\bigl(4w^{(t+1)}\bigr) - 1 \le \exp\bigl(4 \cdot 0.25 \log(1+\tau)\bigr) - 1 = \tau\,,$$

for all i, j, which proves the first statement of the theorem.

Let us prove the second statement of the theorem. Assume that

$$(1+\tau)^{-\frac{1}{2}} \le r_i'^{(t+2)}\,, \qquad (1+\tau)^{-\frac{1}{2}} \le s_j'^{(t+2)}$$

for all i and j. Taking logarithms yields

$$-0.5\log(1+\tau) \le \log r_i'^{(t+2)}\,, \qquad -0.5\log(1+\tau) \le \log s_j'^{(t+2)}\,.$$

We show in the proof of Lemma 19 that $r_i'^{(t+2)} \le 1$, and therefore $\log r_i'^{(t+2)} \le 0$ and similarly for $s_j'^{(t+2)}$. Hence

$$\bigl|\log r_i'^{(t+2)}\bigr| \le 0.5\log(1+\tau)\,, \qquad \bigl|\log s_j'^{(t+2)}\bigr| \le 0.5\log(1+\tau)\,,$$

and hence

$$w^{(t+2)} = \max\Bigl(\max_i \bigl|\log r_i'^{(t+2)}\bigr|, \max_j \bigl|\log s_j'^{(t+2)}\bigr|\Bigr) \le 0.5\log(1+\tau)\,.$$

Substituting this into (39) yields

$$\mu_{ij}^{(t+2)} \le \exp\bigl(2w^{(t+2)}\bigr) - 1 \le \exp\bigl(2 \cdot 0.5\log(1+\tau)\bigr) - 1 = \tau\,,$$

for all i, j, which proves the second statement of the theorem. ∎

The stopping criterion works as follows. We start testing the intermediate scaling factors $p_k'^{(t)}$, $r_i'^{(t)}$ and $s_j'^{(t)}$ in each iteration starting with iteration $t_1 = t_0 + 1$, the iteration that immediately follows the first O step. In the I steps, we test whether all of the $p_k'^{(t)}$ fall within the interval $\bigl[(1+\tau)^{-\frac{1}{4}}, (1+\tau)^{\frac{1}{4}}\bigr]$, and in the O steps, we test whether all of the $r_i'^{(t)}$ and $s_j'^{(t)}$ are greater than the threshold $(1+\tau)^{-1/2}$. Whenever one of these conditions is true, Theorem 23 states that we are within a relative distance τ from the limit, and so we stop iterating.

6.5. Scaling is not always enough. We conclude this section with an example where Strassen's algorithm computes a result with large relative error, using any of the scaling algorithms presented in this section.

Example 24. Consider the matrices

$$\mathbf{A} = \begin{bmatrix} 1 & z \\ z & 1 \end{bmatrix},\ \mathbf{B} = \begin{bmatrix} 1 & z \\ z & 1 \end{bmatrix},\ \mathbf{C} = \mathbf{A}\cdot\mathbf{B} = \begin{bmatrix} 1+z & 2z \\ 2z & 1+z \end{bmatrix}. \tag{40}$$

In this case, both outside and inside scaling leave the matrix unchanged. When computing c_{12},

$$m_3 = a_{11}(b_{12} - b_{22}) = 1(z-1) \tag{41a}$$

$$m_5 = (a_{11} + a_{12})b_{22} = (1+z)1 \tag{41b}$$

$$c_{12} = m_3 + m_5, \tag{41c}$$

There are subterms on the order of unity, so the relative error is $O(1/z)$.

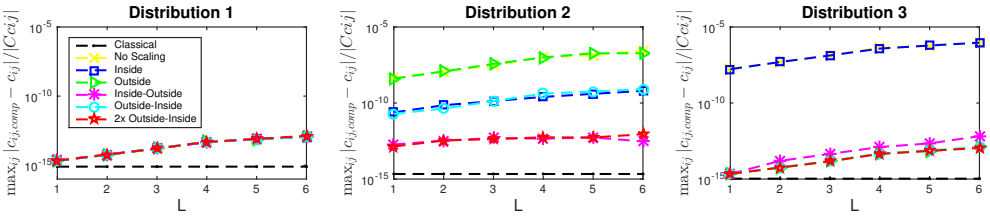

Fig. 3: Relative error of Strassen's algorithm as a function of the number of recursive steps, L, for several scaling techniques. The results in each plot are for matrices $\mathbf{A}$ and $\mathbf{B}$ sampled from different probability distributions. (Top) Stability is well-behaved, and no scaling is necessary for small relative errors. (Middle) The matrices are adversarial and inside-outside or 2-times repeated outside-inside scaling have the smallest relative errors. (Right) The matrices are adversarial and outside, outside-inside, or 2-times repeated outside-inside scaling have the smallest relative errors.

6.6. Numerical experiments. We tested the scaling algorithms on samples of random matrices whose entries were not as contrived as those in the prior sections. We used a sample of $\mathbf{A} \in \mathbb{R}^{N\times N}$ and $\mathbf{B}^{N\times N}$ from the following distributions:

1. $a_{ij}, b_{ij} \sim \text{Uniform}(0,\, 1)$
2. $a_{ij} \sim \text{Uniform}(0, 1/N^2)$ if $j > N/2$, otherwise, $a_{ij} \sim \text{Uniform}(0, 1)$; $b_{ij} \sim \text{Uniform}(0, 1/N^2)$ if $i < N/2$, otherwise $b_{ij} \sim \text{Uniform}(0, 1)$
3. $a_{ij} \sim \text{Uniform}(0, N^2)$ if $i < N/2$ and $j > N/2$, otherwise, $a_{ij} \sim \text{Uniform}(0, 1)$; $b_{ij} \sim \text{Uniform}(0, 1/N^2)$ if $j < N/2$, otherwise $b_{ij} \sim \text{Uniform}(0, 1)$

Samples from the first distribution are well-behaved for fast matrix multiplication algorithms. On the other hand, samples from the second and third distributions are "adversarial" and model the matrices in Examples 10 and 16, respectively.

We sampled one pair of matrices ($N = 3500$) from each distribution and computed the error with Strassen's algorithm for $L = 1, 2, \ldots, 6$. Figure 3 summarizes the results, showing the largest relative error made by the fast algorithm with different scaling techniques. We estimated the largest relative error $|\hat{c}_{ij} - c_{ij}|/|c_{ij}|$, where $\mathbf{C}$ is actually a computed product with quadruple precision. For the first probability distribution, the relative errors are all roughly the same. With the second distribution, only inside-outside scaling or 2-times repeated outside-inside scaling compute relatively accurate solutions. In this case, inside and outside-inside scaling are moderately more accurate than no scaling or outside scaling, but they still produce relative errors several orders of magnitude larger than the best case. Finally, for the third distribution, inside scaling and no scaling result in much larger relative errors, and inside-outside scaling is slightly worse than outside, outside-inside, or 10-times repeated outside-inside scaling. These experiments demonstrate that with no prior knowledge of the distribution, repeated outside-inside scaling is the safe choice for fast matrix multiplication. Corollaries 13 and 15 provide the theoretical underpinning for this empirical observation.

Each iteration of outside or inside scaling is $O(N^2)$ flops, so scaling does not affect the asymptotic performance. However, quadratic costs do affect the practical implementation of fast matrix multiplication [3]. Subsequently we tested the performance impact of scaling. We use *effective gflops* [3, 21] to measure the performance

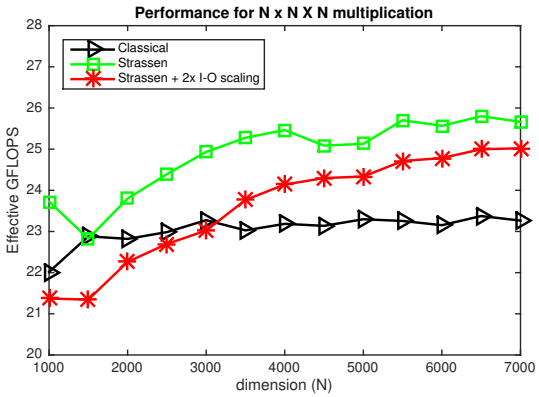

Fig. 4: Performance of Strassen's algorithm ($L = 1$), with and without two iterations of inside-outside scaling, and the classical algorithm. Each data point is the median of five trials.

of multiplying an $M \times K$ matrix by a $K \times N$ matrix:

$$\frac{2 \cdot MKN - MN}{\text{time in seconds}} \cdot \text{1e-9}. \tag{42}$$

This lets us compare fast matrix multiplication algorithms to the classical algorithm on a familiar inverse-time scale. All experiments were conducted on a single compute node on NERSC's Edison machine. Each node has two 12-core Intel 2.4 GHz Ivy Bridge processors and 64 GB of memory. Our experiments were single-threaded.

Figure 4 summarizes the performance results for Strassen's algorithm ($L = 1$), with and without 2x repeated outside-inside scaling, for multiplying square matrices of dimension N. There is a noticeable impact on performance. Strassen's without scaling out-performs the classical algorithm for $N \geq 2500$ while scaling pushes this threshold to $N \geq 3500$. As N grows, the performance impact of scaling gets smaller. This follows from the asymptotic analysis—as N grows, the impact from quadratic terms shrinks.

7. Discussion. One of the central components of our error analysis is that two data-independent quantities drive the error bounds. First, Q captures the *accumulation error* from adding matrices. Second, E measures the *growth* in the number of terms. Our results in Section 5 show that having a small E is important, but this does not fully characterize stability in practice. The same result has been observed when comparing Strassen's algorithm and the Winograd variant [15]. A positive result from our experiments is that the number of non-zeroes in the $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ matrices, which determines the constant for the asymptotic complexity, is positively correlated with E. In other words, by minimizing the number of matrix additions, we also improve stability. Another lesson from our analysis is that we should not think of using fast algorithms asymptotically but rather as having a fixed number of recursive levels. This leads to better performance in practice [3] and also to the improved error bounds and numerical stability presented in Sections 4 and 5. Finally, because the principal quantities for understanding algorithmic error (E and Q) are independent of the asymptotic complexity, we have new metrics over which to optimize when searching for fast matrix multiplication algorithms.

For performance reasons, the best choice of fast algorithm depends on the shape of the matrices being multiplied [3]. In general, a choice of algorithm is made at each recursive level. Subsequently, uniform non-stationary algorithms are the right choice in practice for achieving the best performance. Theorem 5 provides the appropriate error bounds for this case.

The analysis in Subsection 4.5 formalizes the error analysis for existing techniques to improve stability of Strassen's algorithm and the Winograd variant [8, 7] and also generalizes the approach for all fast matrix multiplication algorithms. The analysis provides the formula over which to optimize when considering non-uniform, non-stationary algorithms. However, finding the best algorithm is a combinatorial optimization problem that grows exponentially in the number of recursive levels. Algorithm design in this space is an interesting avenue for future research.

Using the above techniques improves the normwise accuracy of the computed product. However, because the errors are normwise, small elements of the product can be computed less accurately than warranted by their condition numbers. By pre- and post-processing the data, we can improve componentwise accuracy as well. Specifically, we analyzed a hierarchy of diagonal scaling techniques that reduce the number of cases where fast matrix multiplication yields inaccurate small components of the product. Nevertheless, there are cases that cannot be solved by our diagonal scaling algorithms (*e.g.*, Example 24). When scaling helps, a couple of iterations are sufficient, and this is backed up by Theorem 21. In our experiments, we found that scaling incurs a performance penalty in practice. Our implementation did not take advantage of overlapping inside and outside scaling, which would reduce the overhead. The asymptotic cost of scaling is $O(n^2)$ operations. This is a low-order term compared to the cost of current fast multipication algorithms, and therefore the overhead of scaling becomes less significant as the matrices that we multiply become larger.

REFERENCES

[1] Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz, *Communication-optimal parallel algorithm for Strassen's matrix multiplication*, in Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2012, pp. 193–204.

[2] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, *Graph expansion and communication costs of fast matrix multiplication*, in Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '11, ACM, 2011, pp. 1–12, http://doi.acm.org/10.1145/1989493.1989495.

[3] Austin R. Benson and Grey Ballard, *A framework for practical parallel fast matrix multiplication*, in Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2015, ACM, 2015, pp. 42–53, doi:10.1145/2688500.2688513, http://doi.acm.org/10.1145/2688500.2688513.

[4] Dario Bini and Grazia Lotti, *Stability of fast algorithms for matrix multiplication*, Numerische Mathematik, 36 (1980), pp. 63–72.

[5] Markus Bläser, *On the complexity of the multiplication of matrices of small formats*, Journal of Complexity, 19 (2003), pp. 43–60.

[6] Richard P. Brent, *Algorithms for matrix multiplication*, tech. report, Stanford University, Stanford, CA, USA, 1970.

[7] R.R. Castrapel and J.L. Gustafson, *Precision improvement method for the Strassen/Winograd matrix multiplication method*, Apr. 24 2007, http://www.google.com/patents/US7209939. US Patent 7,209,939.

[8] Paolo D'Alberto, *The better accuracy of Strassen-Winograd algorithms (FastMMW)*, Advances in Linear Algebra & Matrix Theory, 4 (2014), p. 9.

[9] Leonardo De Moura and Nikolaj Bjørner, *Z3: An efficient smt solver*, in Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.

[10] James Demmel, Ioana Dumitriu, and Olga Holtz, *Fast linear algebra is stable*, Numerische Mathematik, 108 (2007), pp. 59–91.
[11] James Demmel, Ioana Dumitriu, Olga Holtz, and Robert Kleinberg, *Fast matrix multiplication is stable*, Numerische Mathematik, (2006).
[12] Bogdan Dumitrescu, *Improving and estimating the accuracy of Strassen's algorithm*, Numerische Mathematik, 79 (1998), pp. 485–499.
[13] François Le Gall, *Powers of tensors and fast matrix multiplication*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, 2014, pp. 296–303.
[14] Harold V. Henderson and S. R. Searle, *The vec-permutation matrix, the vec operator and kronecker products: a review*, Linear and Multilinear Algebra, 9 (1981), pp. 271–288, doi:10.1080/03081088108817379, http://dx.doi.org/10.1080/03081088108817379.
[15] Nicholas J Higham, *Accuracy and stability of numerical algorithms*, SIAM, 2002.
[16] R. W. Johnson and A. M. McLoughlin, *Noncommutative bilinear algorithms for 3 x 3 matrix multiplication*, SIAM Journal on Computing, 15 (1986), pp. 595–603.
[17] Igor Kaporin, *A practical algorithm for faster matrix multiplication*, Numerical Linear Algebra with Applications, 6 (1999), pp. 687–700, doi:10.1002/(SICI)1099-1506(199912)6:8¡687::AID-NLA177¿3.0.CO;2-I, http://dx.doi.org/10.1002/(SICI)1099-1506(199912)6:8<687::AID-NLA177>3.0.CO;2-I.
[18] Igor Kaporin, *The aggregation and cancellation techniques as a practical tool for faster matrix multiplication*, Theoretical Computer Science, 315 (2004), pp. 469–510.
[19] J. Laderman, V. Pan, and X.-H. Sha, *On practical algorithms for accelerated matrix multiplication*, Linear Algebra and its Applications, 162 (1992), pp. 557–588.
[20] J. Landsberg, *New lower bounds for the rank of matrix multiplication*, SIAM Journal on Computing, 43 (2014), pp. 144–149, doi:10.1137/120880276, http://dx.doi.org/10.1137/120880276, arXiv:http://dx.doi.org/10.1137/120880276.
[21] Benjamin Lipshitz, Grey Ballard, James Demmel, and Oded Schwartz, *Communication-avoiding parallel Strassen: Implementation and performance*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, 2012, p. 101.
[22] Webb Miller, *Computational complexity and numerical stability*, SIAM Journal on Computing, 4 (1975), pp. 97–107.
[23] AV Smirnov, *The bilinear complexity and practical algorithms for matrix multiplication*, Computational Mathematics and Mathematical Physics, 53 (2013), pp. 1781–1795.
[24] Volker Strassen, *Gaussian elimination is not optimal*, Numerische Mathematik, 13 (1969), pp. 354–356.

Appendix A. Strassen's Algorithm. Strassen's algorithm [24] is a $\langle 2, 2, 2\rangle$ algorithm specified by the following $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ matrices:

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Note that the rows of $\mathbf{U}$ and $\mathbf{V}$ correspond to a column-major ordering of the entries of the input matrices and the rows of $\mathbf{W}$ correspond to a row-major ordering of the output matrix, following the convention of previous work [6, 16]. We point out that this algorithm is cyclic-invariant, so that $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!] = [\![\mathbf{W}, \mathbf{U}, \mathbf{V}]\!] = [\![\mathbf{V}, \mathbf{W}, \mathbf{U}]\!]$ (up to permutations on the columns of the matrices), which implies that all three rotations have the same Q and E values.

Appendix B. $\langle 3,2,3\rangle$ fast matrix multiplication algorithm. The following algorithm for base case $\langle 3,2,3\rangle$ has 94 nonzeros with $E = 20$ and $Q = 10$.

$$\mathbf{U} = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & 0 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}.$$

Note that the rows of $\mathbf{U}$ and $\mathbf{V}$ correspond to a column-major ordering of the entries of the input matrices and the rows of $\mathbf{W}$ correspond to a row-major ordering of the output matrix, which implies that $[\![\mathbf{W}, \mathbf{U}, \mathbf{V}]\!]$ is an algorithm for $\langle 3,3,2\rangle$ and $[\![\mathbf{V}, \mathbf{W}, \mathbf{U}]\!]$ is an algorithm for $\langle 2,3,3\rangle$.

Appendix C. $\langle 4,4,2\rangle$ fast matrix multiplication algorithm. The following algorithm specifies a rank 26 fast matrix multiplication algorithm with base case $\langle 4,4,2\rangle$.

$$\mathbf{U} = \frac{1}{2}\begin{bmatrix}
2 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & -2 & 0 & 2 & -2 & 0 \\
0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 2 & 0 & -1 & -1 & 0 & -1 & -2 & 2 & 0 & -2 & -2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & -2 & 2 & 0 & 2 & 0 \\
0 & 2 & 0 & 0 & -2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -2 & 2 & 0 & 2 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \\
0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \\
0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$\mathbf{V} = \frac{1}{2}\begin{bmatrix}
2 & 0 & -2 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 2 & 2 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & -2 & 2 & 2 & 0 & 0 & -2 & 1 & 0 \\
-2 & 0 & -2 & -2 & 0 & -2 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\
-2 & 0 & 2 & -2 & 0 & 0 & -2 & 0 & -2 & 2 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 & 0 & 2 & 2 & 0 & -2 & -2 & 0 & 0 \\
2 & 1 & -2 & 2 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2 & 2 & 0 & -2 & 0 & 0 & 2 & -1 & 2 \\
2 & 1 & -2 & 2 & 0 & 2 & 0 & 2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 & 2 & 0 & 2 & -2 & 0 & -2 & 0 & 0 & 2 & -1 & 0 \\
-2 & -2 & -2 & -2 & 0 & 0 & -2 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 2 & 2 & 2 & 0 & -2 \\
-2 & 0 & -2 & -2 & -2 & 2 & -2 & 0 & -2 & -2 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -2 & 2 & -2 & 0
\end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix}
1 & -1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & -1 \\
0 & -1 & 0 & 1 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & -1 & 1 & -1 & -1 & -1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & -1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & -1 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}.$$

Note that the rows of $\mathbf{U}$ and $\mathbf{V}$ correspond to a column-major ordering of the entries of the input matrices and the rows of $\mathbf{W}$ correspond to a row-major ordering of the output matrix, which implies that $[\![\mathbf{W}, \mathbf{U}, \mathbf{V}]\!]$ is an algorithm for $\langle 4, 2, 4\rangle$ and $[\![\mathbf{V}, \mathbf{W}, \mathbf{U}]\!]$ is an algorithm for $\langle 2, 4, 4\rangle$. However, $[\![\mathbf{V}, \mathbf{W}, \mathbf{U}]\!]$ yields an $E = 102$, which is greater than $[\![\mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$'s 89. The E value can be maintained for base case $\langle 2, 4, 4\rangle$ by using a different transformation that corresponds to transposing the matrix multiplication: $[\![\boldsymbol{\mathcal{P}}_{4,2}\mathbf{V}, \boldsymbol{\mathcal{P}}_{4,4}\mathbf{U}, \boldsymbol{\mathcal{P}}_{2,4}\mathbf{W}]\!]$, where $\boldsymbol{\mathcal{P}}_{m,n}$ is the *vec permutation* matrix [14], exchanging column-ordering for row-ordering for a vectorized $m \times n$ matrix.