INTEGER MULTIPLICATION IS AT LEAST AS HARD AS MATRIX TRANSPOSITION

DAVID HARVEY AND JORIS VAN DER HOEVEN

Abstract. Working in the multitape Turing model, we show how to reduce the problem of matrix transposition to the problem of integer multiplication. If transposing an $n \times n$ binary matrix requires $\Omega(n^2 \log n)$ steps on a Turing machine, then our reduction implies that multiplying n-bit integers requires $\Omega(n \log n)$ steps. In other words, if matrix transposition is as hard as expected, then integer multiplication is also as hard as expected.

1. Introduction

We work throughout in the *multitape Turing model*. In this model, an "algorithm" is a Turing machine with a fixed, finite number of one-dimensional tapes, and the time complexity of an algorithm refers to the number of steps executed by the machine. For detailed definitions see for instance [AHU75, §1.6] or [Pap94, Ch. 2].

For any integer $n \geqslant 1$ we define

$$\lg n \coloneqq \max(\lceil \log_2 n \rceil, 1).$$

For an integer $\ell \geqslant 0$ we write $\lg^{\circ \ell}$ for the ℓ-fold composition of lg. Since $1 \leqslant \lg n < n$ for all $n \geqslant 2$, it makes sense to define the *iterated logarithm* function,

$$\lg^* n \coloneqq \min\{\ell \geqslant 0 : \lg^{\circ \ell} n = 1\}, \qquad n \geqslant 1.$$

For functions $f(x)$ and $g(x)$ defined on some domain $D \subseteq \mathbb{Z}^n$, the statement $f(x) = O(g(x))$ (respectively $f(x) = \Omega(g(x))$) means that there exists a constant $C > 0$ such that $f(x) \leqslant Cg(x)$ (respectively $f(x) \geqslant Cg(x)$) for all $x \in D$. For functions $f(n)$ and $g(n)$ defined on $\mathbb{Z}^+$, we write $f(n) = \omega(g(n))$ to mean that $\lim_{n \to \infty} f(n)/g(n) = \infty$.

1.1. Multiplication and transposition.

In this paper we study the relationship between the complexity of two fundamental problems in the Turing model:

- *Integer multiplication.* The input consists of an integer $m \geqslant 1$ and two non-negative m-bit integers x and y. The output is the product xy. All integers are assumed to be stored in the standard binary representation.

  A machine M computing this function will be called a *multiplication machine*. We write $\mathsf{M}_M(m)$ for the worst-case running time of M on m-bit inputs. The subscript M may be omitted if it is clear from context.

1991 *Mathematics Subject Classification.* Primary 68Q17, Secondary 68W30.

David Harvey was supported by ARC Future Fellowship grant FT160100219.

Joris van der Hoeven was supported by an ERC-2023-ADG grant for the ODELIX project (number 101142171).

- *Matrix transposition.* The input consists of integers $n_1, n_2, b, m \geqslant 1$ such that $n_1 n_2 b \leqslant m$, and an $n_1 \times n_2$ matrix with b-bit entries. The matrix is encoded on the tape in row-major order, i.e., as an array of b-bit strings of length $n_1 n_2$, with the matrix entry A_{i_1,i_2} stored at index $n_2 i_1 + i_2$ in the array, for $0 \leqslant i_1 < n_1$ and $0 \leqslant i_2 < n_2$. The output consists of the transpose of this matrix, i.e., the matrix entries must be rearranged into an $n_2 \times n_1$ array, with $A_{i_1,i_2} = (A^T)_{i_2,i_1}$ now appearing at position $n_1 i_2 + i_1$. Alternatively, one may think of this operation as switching from row-major to column-major representation.

  (The parameter m is included for technical reasons: our matrix transposition algorithms will often need to "know" what size integer multiplication problem is being targeted. One may think of m as an upper bound for the total bit size of the matrix, which is $n_1 n_2 b$.)

  A machine T computing this function will be called a *transposition machine*. We write $\mathsf{T}_T(m; n_1, n_2, b)$ for the worst-case running time of T for the given parameters.

The case of *square binary matrices*, where $n_1 = n_2$ and $b = 1$, is of particular interest. A machine handling this case will be called a *binary transposition machine* (for brevity we omit "square"). It is convenient to describe the input size by the single parameter m, so we assume that such a machine takes as input an integer $m \geqslant 1$ and an $n \times n$ binary matrix, where $n := \lfloor m^{1/2} \rfloor$. We denote its worst-case running time by $\mathsf{T}_T(m)$.

The following upper bounds are known for the complexity of these problems. For integer multiplication, the present authors recently described a (somewhat complicated) algorithm M^* [HvdH21] that achieves

$$\mathsf{M}_{M^*}(m) = O(m \lg m). \tag{1.1}$$

For matrix transposition, there is a simple folklore algorithm T^* achieving

$$\mathsf{T}_{T^*}(m; n_1, n_2, b) = O(n_1 n_2 b \lg \min(n_1, n_2)) = O(m \lg \min(n_1, n_2)). \tag{1.2}$$

In particular, for the binary case this algorithm achieves

$$\mathsf{T}(m) = O(m \lg m).$$

Briefly, the algorithm operates as follows. If $n_1 \leqslant n_2$, it first recursively transposes the "top" and "bottom" halves of the matrix, and then interleaves the results together appropriately. The $n_1 > n_2$ case may be handled by running the same algorithm in reverse. (We do not know the precise history of this algorithm. The basic idea goes back at least to [Sto71, Flo72, Pau73], although these papers work in somewhat different complexity models. For a more modern presentation, see for example [BGS07, Lem. 18].)

What is believed to be the true complexity of integer multiplication and matrix transposition? For both problems, non-trivial lower bounds are known only in certain very restricted models of computation (see Section 1.4). If we allow the full power of the multitape Turing machine, then unfortunately no lower bounds are known beyond the trivial linear bounds $\mathsf{M}(m) = \Omega(m)$ and $\mathsf{T}(m; n_1, n_2, b) = \Omega(n_1 n_2 b)$. (These lower bounds must hold because the machine needs enough time to write its output.) It seems reasonable to guess that for both problems, the upper bounds mentioned above are sharp, i.e., we suspect that for any multiplication

machine M,

$$\mathsf{M}_M(m) = \Omega(m \lg m), \tag{1.3}$$

and for any transposition machine T,

$$\mathsf{T}_T(m; n_1, n_2, b) = \Omega(n_1 n_2 b \lg \min(n_1, n_2)).$$

In particular, for any binary transposition machine T, we expect that

$$\mathsf{T}_T(m) = \Omega(m \lg m).$$

The lower bound (1.3) was already suggested by Schönhage and Strassen in their famous paper [SS71]. We are not aware of any officially conjectured lower bound for transposition in the literature, although the absence of non-trivial lower bounds in the Turing model for transposition and other basic operations has attracted some attention [Reg95].

Remark 1.1. In a statement of the form (1.3), the implied big-Ω constant depends on the machine M. This is unavoidable, because the *linear speedup theorem* [Pap94, Thm. 2.2] states that for any Turing machine A running in time $f(m)$ on input of size m, and for any constant $C > 1$, we can construct another Turing machine A' that solves the same problem as A in time at most $f(m)/C + O(m)$. (The new machine simulates the original machine, executing several steps of the original machine in each step, by using a packed encoding of the original alphabet.)

1.2. **Summary of main results.** The aim of this paper is to present various reductions from matrix transposition to integer multiplication. We state here some sample consequences of these reductions, concentrating on the binary case for simplicity. The bottom line is that *lower bounds for matrix transposition imply lower bounds for integer multiplication.* The proofs of all results in this section (and some more general results) will be given in Section 6.

First, we have the following striking relationship between the lower bound conjectures for multiplication and transposition mentioned earlier.

Theorem 1.2. *If the conjectured lower bound*

$$\mathsf{T}_T(m) = \Omega(m \lg m)$$

holds for every binary transposition machine T, then the conjectured lower bound

$$\mathsf{M}_M(m) = \Omega(m \lg m)$$

holds for every multiplication machine M.

In fact, we can prove something slightly stronger than this: provided that $\mathsf{T}(m) = \omega(m \lg \lg m)$, then roughly speaking $\mathsf{M}(m) = \Omega(\mathsf{T}(m))$, i.e., multiplying integers is at least as hard as transposing matrices of the same bit size. For a precise statement, see Corollary 6.3.

If even the lower bound $\mathsf{T}(m) = \omega(m \lg \lg m)$ is out of reach, then we can still say something, but our results are somewhat weaker. For example, if $\mathsf{T}(m) = \omega(m \lg \lg \lg m)$, then we would like to be able to prove that $\mathsf{M}(m) = \omega(m \lg \lg \lg m)$, but we cannot quite manage this. Instead we have the following weaker statement.

Theorem 1.3. *Fix $\ell \geqslant 2$. If no binary transposition machine achieves $\mathsf{T}(m) = O(m \lg^{\circ \ell} m)$, then no multiplication machine achieves $\mathsf{M}(m) = O(m \lg^{\circ \ell} m)$.*

Finally, if we cannot even rule out the possibility that $\mathsf{T}(m) = O(m \lg^{\circ \ell} m)$ is feasible for every fixed ℓ, then we make our last stand with the following result.

Theorem 1.4. *Let $f(m)$ be a non-decreasing function. If no binary transposition machine achieves $\mathsf{T}(m) = O(mf(m) \lg^* m)$, then no multiplication machine achieves $\mathsf{M}(m) = O(mf(m))$.*

Theorem 1.4 implies for example that if no transposition machine achieves $\mathsf{T}(m) = O(m \lg^* m)$, then multiplication cannot be performed in linear time. We would prefer to be able to prove the stronger statement that if transposition cannot be carried out in linear time, then neither can multiplication, but our methods do not appear to be strong enough to prove this. (See however Theorem 6.11, which gives a result of similar strength for matrices with larger coefficients.)

Theorems 1.2–1.4 suggest that instead of trying to prove lower bounds for multiplication directly, it might be preferable to attack the transposition problem. After all, transposition merely moves data around; it does not seem to involve any actual *computation*. Unfortunately, as mentioned earlier, we still have no idea how to prove non-trivial lower bounds for transposition in the Turing model. We hope that the results of this paper will inspire further research on this question.

1.3. **Overview of methods.** The core of our reduction runs as follows. Suppose that we wish to transpose an $n_1 \times n_2$ matrix with b-bit entries. We interpret the matrix as a vector in $\mathbb{C}^{n_1 n_2}$, regarding its entries as b-bit fixed-point approximations to complex numbers. The idea is to compute the DFT (discrete Fourier transform) of this vector using one algorithm, and then compute the inverse DFT of the result using a different algorithm, in such a way that the original data comes back in transposed order.

For the DFT in the forward direction, we use Bluestein's method [Blu70] to reduce the DFT to a convolution problem, and then Kronecker substitution [GG13, Cor. 8.27] to reduce the convolution to a large integer multiplication problem. The details of the Bluestein–Kronecker combination are worked out in Section 3.

For the inverse DFT, we first use the Cooley–Tukey method [CT65] to decompose the transform into DFTs along the rows and columns of the matrix, plus a collection of multiplications by "twiddle factors". We then handle the row and column transforms using the same Bluestein–Kronecker combination mentioned above. This strategy is explained in Section 4. The reason that it succeeds in transposing the data ultimately comes down to algebraic properties of the Cooley–Tukey decomposition. (This is not unrelated to the familiar fact that textbook "decimation-in-time" and "decimation-in-frequency" FFT algorithms naturally produce their output in bit-reversed order.)

In all of the above steps, it is crucial to carefully manage the ordering of data on the tape. For instance, we rely heavily on the fact that Bluestein's trick computes the DFT in the natural ordering; it would not work to substitute Rader's algorithm [Rad68], which involves complicated permutations of the data. Another example is that in the Cooley–Tukey decomposition, we cannot afford to transpose the data so that the column transforms operate on contiguous data; instead we must show how to carry out the Kronecker substitution for all columns in parallel, without reordering the data.

Throughout the computation, we must ensure that all numerical computations are performed with sufficient accuracy to recover the correct (integer) results at the

end of the transposition. This entails some straightforward but tedious numerical analysis. To facilitate this, in Section 2 we introduce a framework for analysing numerical error in fixed-point computations, along the lines of [HvdH21, §2].

The approach described so far has a serious limitation: it only works when the coefficient size b is sufficiently large compared to the matrix dimensions n_1 and n_2. This occurs for several reasons, including coefficient growth in the Kronecker substitution step, and the need to be able to represent complex $(n_1 n_2)$-th roots of unity with sufficient accuracy. As a partial workaround for this problem, in Section 5 we explain how to decompose a given transposition problem into a collection of transposition problems of exponentially smaller dimension. In Section 6 we show how to use this strategy to handle transposition problems involving smaller coefficient sizes, all the way down to single-bit coefficients, and thereby finish the proofs of the results stated in Section 1.2.

Remark 1.5. We take this opportunity to correct a missing attribution in one of our earlier papers. The Bluestein–Kronecker combination was announced in [HvdHL16], but in fact, as pointed out to us by Dan Bernstein (personal communication, 2022), it goes back at least to [Sch82, §3].

1.4. **Known lower bounds.** In the standard Turing model, virtually no non-trivial (i.e. superlinear) lower bounds are known for basic computational tasks such as transposition, integer multiplication, FFTs, finding duplicate elements in a list, sorting, and so on [Reg95]. Interesting lower bounds have been proved for the Turing model under additional restrictions and for various other complexity models.

Perhaps the most impressive lower bound result for multiplication is the $\Omega(n \log n)$ proved for "online" (or relaxed) integer multiplication [PFM74, CA69], where we restrict the bits of the input and output to be given and computed "one by one". An almost matching upper bound $O(n \log n\, e^{O(\sqrt{\log \log n})})$ is also known for this problem [Hoe14]. In the Boolean circuit model, a $\Omega(n \log n)$ lower bound was recently obtained as well, conditional on an open conjecture on network coding [AFKL19]. For the related problem of FFT computations, Morgenstern proved an $\Omega(n \log n)$ lower bound for evaluating a complex DFT of length n in a suitably restricted algebraic complexity model [Mor73].

Concerning the binary $n \times n$ matrix transposition problem, a natural restriction is to consider Turing machines that can only move data, but not perform any actual computations. In such a model, Stoß proved the expected $\Omega(n^2 \log n)$ lower bound [Sto73]. Non-trivial lower bounds are also known for a few less natural complexity models, such as a two-level fast-slow memory model [Flo72] or Turing machines with a single tape [Kir88, DM93]. Finally, conditional on the same network coding conjecture as above, an $\Omega(n^2 \log n)$ lower bound has been proved for Turing machines with two tapes [AHJ+06].

2. Fixed-point arithmetic

Let $\mathcal{D} \coloneqq \{u \in \mathbb{C} : |u| \leqslant 1\}$ be the complex unit disc. Let $p \geqslant 1$ be a precision parameter, and define

$$\tilde{\mathcal{D}} \coloneqq (2^{-p}\,\mathbb{Z}[\mathrm{i}]) \cap \mathcal{D} = \{2^{-p}(x + \mathrm{i}y) : x, y \in \mathbb{Z} \text{ and } x^2 + y^2 \leqslant 2^{2p}\}.$$

We will regard elements of $\tilde{\mathcal{D}}$ as finite-precision approximations for elements of $\mathcal{D}$. An element of $\tilde{\mathcal{D}}$ will be represented on the tape by the pair of integers (x, y), and occupies $O(p)$ space.

If $z \in \mathcal{D}$, we systematically write $\tilde{z} \in \tilde{\mathcal{D}}$ for a fixed-point approximation for z that has been computed by some algorithm. We write

$$\varepsilon(\tilde{z}) \coloneqq 2^p\,|\tilde{z} - z|$$

for the associated error, measured as a multiple of 2^{-p} (the "unit in the last place").

We define a round-towards-zero function $\rho\colon \mathbb{C} \to \mathbb{C}$ as follows. First, define $\rho_0\colon \mathbb{R} \to \mathbb{Z}$ by $\rho(x) \coloneqq \lfloor x \rfloor$ for $x \geqslant 0$ and $\rho(x) \coloneqq \lceil x \rceil$ for $x < 0$. Then define $\rho_0\colon \mathbb{C} \to \mathbb{Z}[\mathrm{i}]$ by setting $\rho_0(x + \mathrm{i}y) \coloneqq \rho_0(x) + \mathrm{i}\rho_0(y)$. Finally, set $\rho(z) \coloneqq 2^{-p}\rho_0(2^p z)$ for $z \in \mathbb{C}$. In other words, $\rho(z)$ rounds the real and imaginary parts of z to the nearest multiple of 2^{-p} in the direction of the origin. Clearly $|\rho(z)| \leqslant |z|$ for all $z \in \mathbb{C}$, so ρ maps $\mathcal{D}$ to $\tilde{\mathcal{D}}$, and also

$$|\rho(z) - z| \leqslant \sqrt{2} \cdot 2^{-p}, \qquad z \in \mathbb{C}. \tag{2.1}$$

The following result is almost identical to [HvdH21, Cor. 2.10].

Lemma 2.1 (Fixed point multiplication)**.** *Given any multiplication machine M, there exists a Turing machine with the following properties. Its input consists of the precision parameter p and approximations $\tilde{u}, \tilde{v} \in \tilde{\mathcal{D}}$ for $u, v \in \mathcal{D}$. Its output is an approximation $\tilde{w} \in \tilde{\mathcal{D}}$ for $w \coloneqq uv \in \mathcal{D}$ such that $\varepsilon(\tilde{w}) < \varepsilon(\tilde{u}) + \varepsilon(\tilde{v}) + 2$. Its running time is $O(\mathsf{M}_M(p))$.*

Proof. We first compute $\tilde{u}\tilde{v} \in \mathcal{D}$ by multiplying out the real and imaginary parts of $\tilde{u}$ and $\tilde{v}$ and summing appropriately. We then define $\tilde{w} \coloneqq \rho(\tilde{u}\tilde{v}) \in \tilde{\mathcal{D}}$. Applying ρ amounts to a simple rounding operation, so the complexity of computing $\tilde{w}$ is $4\,\mathsf{M}_M(p) + O(p) = O(\mathsf{M}_M(p))$. As for the error bound, observe that

$$\varepsilon(\tilde{w}) = 2^p\,|\tilde{w} - w| \leqslant 2^p\,|\rho(\tilde{u}\tilde{v}) - \tilde{u}\tilde{v}| + 2^p\,|\tilde{u}\tilde{v} - uv|.$$

We have $2^p\,|\rho(\tilde{u}\tilde{v}) - \tilde{u}\tilde{v}| \leqslant \sqrt{2}$ by (2.1). For the second term,

$$\begin{aligned} 2^p\,|\tilde{u}\tilde{v} - uv| \leqslant 2^p\,|\tilde{u}\tilde{v} - u\tilde{v}| + 2^p\,|u\tilde{v} - uv| &= 2^p\,|\tilde{u} - u| \cdot |\tilde{v}| + |u| \cdot 2^p\,|\tilde{v} - v| \\ &\leqslant \varepsilon(u) \cdot 1 + 1 \cdot \varepsilon(v). \end{aligned}$$

Thus $\varepsilon(\tilde{w}) \leqslant \sqrt{2} + \varepsilon(u) + \varepsilon(v) < \varepsilon(u) + \varepsilon(v) + 2$. □

We next consider the computation of roots of unity. For $n \geqslant 1$, define

$$\zeta_n \coloneqq e^{2\pi\mathrm{i}/n} \in \mathcal{D}.$$

Lemma 2.2 (Roots of unity)**.** *For any constant $C > 0$, there exists a Turing machine with the following properties. Its input consists of the precision parameter p and a positive integer $n < 2^{Cp}$. Its output is an approximation $\tilde{\zeta}_n \in \tilde{\mathcal{D}}$ for ζ_n with $\varepsilon(\tilde{\zeta}_n) < 2$. Its running time is $O(p \lg^2 p)$.*

(In the above lemma, and in similar situations throughout the paper, the big-O constant depends on C. The symbol C will be reused many times; it does not refer to the same constant each time.)

Proof. Let $q \coloneqq p + C'$ for a suitable constant $C' > 0$. Using standard methods (see for example [BB87, Chs. 6–7] or [BZ11, Ch. 4]) together with the multiplication machine M^* (whose complexity is given by (1.1)), we may compute a q-bit approximation to π in time $O(q \lg^2 q)$, a q-bit approximation to $1/n$ in time $O(q \lg q)$, and then q-bit approximations to $\cos(2\pi/n)$ and $\sin(2\pi/n)$ in time $O(q \lg^2 q)$. Rounding towards the origin at precision p, we obtain the desired error bound. For further details, see [HvdH21, §2.8]. □

Remark 2.3. The reader may wonder why we used M^* in Lemma 2.2, i.e., why could we not write the complexity bound as $O(\mathsf{M}_M(p)\log p)$ for a given multiplication machine M, along similar lines to Lemma 2.1? The reason is that the standard textbook results rely on properties of $\mathsf{M}_M(m)$ that may not be satisfied for an arbitrary black-box multiplication machine M. Consider for example the reciprocal step in the above proof, i.e., computing $1/n$. In the typical Newton iteration scheme for the reciprocal, the precision doubles at each step, so the complexity is really $O(\mathsf{M}(q) + \mathsf{M}(q/2) + \mathsf{M}(q/4) + \cdots)$. This can only be simplified to $O(\mathsf{M}(q))$ under additional assumptions, such as $\mathsf{M}(q)/q$ being non-decreasing. This holds for M^* but we cannot guarantee that it holds for M.

3. DFTs and the Bluestein–Kronecker trick

3.1. **Conventions for vectors.** If $U \in \mathbb{C}^n$, we write $U_0, \ldots, U_{n-1}$ for the components of U. We write $\mathcal{D}^n$ to denote the subset of $\mathbb{C}^n$ whose entries lie in $\mathcal{D}$, and similarly for $\tilde{\mathcal{D}}^n$. Elements of $\tilde{\mathcal{D}}^n$ are always stored on the tape in the standard ordering $U_0, \ldots, U_{n-1}$. If $U = (U_0, \ldots, U_{n-1}) \in \mathcal{D}^n$ is a vector, and $\tilde{U} = (\tilde{U}_0, \ldots, \tilde{U}_{n-1}) \in \tilde{\mathcal{D}}^n$ is an approximation, we write $\varepsilon(\tilde{U})$ for $\max_i \varepsilon(\tilde{U}_i)$.

3.2. **Discrete Fourier transforms.** Let us recall the definition of the DFT of length $n \geqslant 1$ over $\mathbb{C}$. Given a vector $X \in \mathbb{C}^n$, its DFT is the vector $Y \in \mathbb{C}^n$ defined by

$$Y_t \coloneqq \frac{1}{n} \sum_{s=0}^{n-1} \zeta_n^{-st} X_s, \qquad 0 \leqslant t < n, \tag{3.1}$$

where we recall that $\zeta_n \coloneqq e^{2\pi \mathrm{i}/n} \in \mathcal{D}$. Note the slightly nonstandard scaling factor $1/n$, which we have included to ensure that the DFT maps $\mathcal{D}^n$ into $\mathcal{D}^n$.

It is straightforward to check that the inverse transform is given by

$$X_s = \sum_{t=0}^{n-1} \zeta_n^{st} Y_t, \qquad 0 \leqslant s < n. \tag{3.2}$$

This formula is the same as the formula for the forward DFT, except for a sign change and the absence of the scaling factor.

3.3. **Bluestein's trick.** We now recall Bluestein's reduction from DFTs to convolutions [Blu70]. Let $n \geqslant 1$ and put $\zeta \coloneqq \zeta_n$. Let $X \in \mathbb{C}^n$, and let $Y \in \mathbb{C}^n$ be the DFT of X according to (3.1). For any $k \in \mathbb{Z}$, define

$$\omega_k \coloneqq \zeta^{\binom{k}{2}} = \zeta^{k(k-1)/2}. \tag{3.3}$$

Then the identity $st = \binom{t}{2} + \binom{-s}{2} - \binom{t-s}{2}$ implies that

$$Y_t = \frac{1}{n} \sum_{s=0}^{n-1} \zeta^{-st} X_s = \frac{1}{n}\, \bar{\omega}_t \sum_{s=0}^{n-1} (\bar{\omega}_{-s} X_s) \cdot \omega_{t-s}, \qquad 0 \leqslant t < n.$$

(The bar $\bar{\cdot}$ denotes complex conjugation.) The latter sum may be recognised as a portion of an acyclic convolution of a sequence of length n with a sequence of

length $2n-1$. More explicitly, if we define vectors $X', Y' \in \mathbb{C}^n$ and $B \in \mathbb{C}^{2n-1}$ by

$$X'_s \coloneqq \bar{\omega}_{-s} X_s, \qquad 0 \leqslant s < n, \tag{3.4}$$

$$Y'_t \coloneqq \omega_t Y_t, \qquad 0 \leqslant t < n, \tag{3.5}$$

$$B_r \coloneqq \omega_{r-n+1}, \qquad 0 \leqslant r < 2n-1, \tag{3.6}$$

then the relation becomes

$$Y'_t = \frac{1}{n} \sum_{s=0}^{n-1} X'_s B_{(t+n-1)-s}, \qquad 0 \leqslant t < n. \tag{3.7}$$

Therefore, the desired DFT may be computed by first using (3.4) to compute the vector X' from X, then computing the convolution in (3.7) to obtain Y', and finally deducing Y from Y' via (3.5).

3.4. **Bluestein plus Kronecker substitution.** The following result shows how to use integer multiplication to evaluate a sum of the type (3.7). The idea is to perform a Kronecker substitution [GG13, Cor. 8.27], i.e., pack the coefficients of the sequences into large integers, multiply the resulting integers, and then extract the convolution from the resulting integer product. Recall that p denotes the fixed-point precision as in Section 2.

Proposition 3.1 (One-dimensional convolution)**.** *Let $C, C' > 0$ be constants, and let M, M' be multiplication machines. Then there exists a Turing machine with the following properties. Its input consists of the precision parameter p, positive integers n and m such that*

$$\lg n < Cp, \qquad \lg p < C'n, \qquad np \leqslant m,$$

and approximations $\tilde{F} \in \tilde{\mathcal{D}}^n$ and $\tilde{G} \in \tilde{\mathcal{D}}^{2n-1}$ for vectors $F \in \mathcal{D}^n$ and $G \in \mathcal{D}^{2n-1}$. Define $H \in \mathcal{D}^n$ by the formula

$$H_t \coloneqq \frac{1}{n} \sum_{s=0}^{n-1} F_s G_{(t+n-1)-s}, \qquad 0 \leqslant t < n.$$

The output of the machine is an approximation $\tilde{H} \in \tilde{\mathcal{D}}^n$ such that $\varepsilon(\tilde{H}) < \varepsilon(\tilde{F}) + \varepsilon(\tilde{G}) + 2$. Its running time is

$$O(\mathsf{M}_M(m) + n\,\mathsf{M}_{M'}(p)).$$

Before giving the proof, we mention a simple fact that will be used frequently:

Lemma 3.2. *Let $C > 1$ be a constant and let M be a multiplication machine. Then there exists a Turing machine that takes as input positive integers m and $m' \leqslant Cm$, and two m'-bit integers x and y, and returns the product xy in time $O(\mathsf{M}_M(m))$.*

Proof. Simply cut up each multiplicand of size m' into $\lceil m'/m \rceil = O(1)$ chunks of size m, multiply out the chunks, and sum appropriately. □

Loosely speaking, Lemma 3.2 says that if $m' = O(m)$ then $\mathsf{M}(m') = O(\mathsf{M}(m))$. However, the latter statement is not quite correct, because if the multiplication machine M is given inputs of size m', it does not "know" that it should reduce to problems of size m. The point of Lemma 3.2 is to embed M in a machine that explicitly requires both m and m' as input.

Proof of Proposition 3.1. Recall that $2^p\tilde{F}_s, 2^p\tilde{G}_j \in \mathbb{Z}[\mathrm{i}]$. Consider the polynomials

$$f(x) \coloneqq \sum_{s=0}^{n-1} (2^p\tilde{F}_s)x^s \in \mathbb{Z}[\mathrm{i}][x], \qquad g(x) \coloneqq \sum_{j=0}^{2n-2} (2^p\tilde{G}_j)x^j \in \mathbb{Z}[\mathrm{i}][x],$$

and let $w \coloneqq fg \in \mathbb{Z}[\mathrm{i}][x]$ be their product, say $w(x) = \sum_{k=0}^{3n-3} W_k x^k$. Since $|2^p\tilde{F}_s|, |2^p\tilde{G}_j| \leqslant 2^p$, we have the bound $|W_k| \leqslant n(2^p)^2$ for all k. Let

$$\beta \coloneqq 2p + \lg n + 2,$$

so that $|W_k| \leqslant 2^{\beta-2}$. Note that the hypothesis $\lg n < Cp$ implies that $\beta = O(p)$.

We first compute $f(2^\beta), g(2^\beta) \in \mathbb{Z}[\mathrm{i}]$, by concatenating the input coefficients with appropriate zero-padding (or one-padding in the case of negative coefficients), handling the real and imaginary parts separately. This requires time $O(n\beta) = O(np)$, including the cost of various carry/borrow handling, the details of which are omitted.

We next compute the product $w(2^\beta) = f(2^\beta)g(2^\beta)$ by using M to multiply out the real and imaginary parts of $f(2^\beta)$ and $g(2^\beta)$, working in chunks of size m, and then adding/subtracting appropriately. Each multiplicand has $n\beta = O(np) = O(m)$ bits, so by Lemma 3.2 the cost is $O(\mathsf{M}_M(m))$.

Since the real and imaginary parts of W_k are bounded in absolute value by $2^{\beta-2}$, we may extract all the W_k unambiguously from $w(2^\beta)$ in time $O(np)$, i.e., we first examine $w(2^\beta) \bmod 2^\beta$ to recover W_0, then we remove W_0 from the sum and continue onto W_1, and so on.

At this stage we have recovered the coefficients of interest, namely

$$W_{t+n-1} = \sum_{s=0}^{n-1} (2^p\tilde{F}_s)(2^p\tilde{G}_{(t+n-1)-s}) \in \mathbb{Z}[\mathrm{i}], \qquad 0 \leqslant t < n,$$

and we define the output approximations for H_t to be

$$\tilde{H}_t \coloneqq \rho(W_{t+n-1}/2^{2p}n) \in \tilde{\mathfrak{D}}, \qquad 0 \leqslant t < n.$$

This is well defined, as we showed earlier that $|W_{t+n-1}/2^{2p}n| \leqslant 1$. Computing each $\tilde{H}_t$ from W_{t+n-1} requires a division by n followed by appropriate rounding. The cost of the rounding is $O(p)$. For the division by n, we first use the multiplication machine M^* to precompute a p-bit approximation for $1/n$ in time $O(p \lg p)$ (see Remark 2.3). This bound simplifies to $O(np)$ thanks to the hypothesis $\lg p < C'n$. The division by n then reduces to a p-bit multiplication (plus additional linear-time work); using M' to compute this product, the cost is $O(\mathsf{M}_{M'}(p))$ for each t, and hence $O(n\,\mathsf{M}_{M'}(p))$ altogether.

Finally, observe that

$$\begin{aligned}
\left|\frac{W_{t+n-1}}{2^p n} - 2^p H_t\right| &= \frac{2^p}{n}\left|\sum_{s=0}^{n-1}\left(\tilde{F}_s\tilde{G}_{(t+n-1)-s} - F_s G_{(t+n-1)-s}\right)\right| \\
&\leqslant \frac{2^p}{n}\sum_{s=0}^{n-1}\left(|\tilde{F}_s - F_s||\tilde{G}_{(t+n-1)-s}| + |F_s||\tilde{G}_{(t+n-1)-s} - G_{(t+n-1)-s}|\right) \\
&\leqslant \frac{1}{n}\sum_{s=0}^{n-1}\left(\varepsilon(\tilde{F}_s)\cdot 1 + 1\cdot\varepsilon(\tilde{G}_{(t+n-1)-s})\right) \leqslant \varepsilon(\tilde{F}) + \varepsilon(\tilde{G}),
\end{aligned}$$

and hence, by (2.1),

$$\begin{aligned}\varepsilon(\tilde{H}_t) &= 2^p|\tilde{H}_t - H_t| \\ &\leqslant 2^p \left|\rho\left(\frac{W_{t+n-1}}{2^{2p}n}\right) - \frac{W_{t+n-1}}{2^{2p}n}\right| + 2^p \left|\frac{W_{t+n-1}}{2^{2p}n} - H_t\right| \\ &\leqslant \sqrt{2} + \varepsilon(\tilde{F}) + \varepsilon(\tilde{G})\end{aligned}$$

for all t. □

Proposition 3.3 (One-dimensional DFT)**.** *Let $C, C' > 0$ be constants, and let M, M' be multiplication machines. Then there exists a Turing machine with the following properties. Its input consists of the precision parameter p, positive integers n and m such that*

$$\lg n < Cp, \qquad \lg^2 p < C'n, \qquad np \leqslant m,$$

and an approximation $\tilde{X} \in \tilde{\mathcal{D}}^n$ for a vector $X \in \mathcal{D}^n$. Let $Y \in \mathcal{D}^n$ be the DFT of X according to (3.1)*. The output of the machine is an approximation $\tilde{Y} \in \tilde{\mathcal{D}}^n$ such that $\varepsilon(\tilde{Y}) < \varepsilon(\tilde{X}) + 12n^2$. Its running time is*

$$O(\mathsf{M}_M(m) + n\,\mathsf{M}_{M'}(p)).$$

Remark 3.4. Let us comment briefly on the significance of the hypotheses in Proposition 3.3. The condition $\lg n < Cp$ was already alluded to in Section 1.3, and seems to be unavoidable; the reduction simply does not work if p is too small relative to n. By contrast, the hypothesis $\lg^2 p < C'n$ is merely a technical annoyance. It is included to control the cost of computing roots of unity, but we will see in Section 6 that it does not cause any serious problems, because it is always possible to cut up large coefficients into smaller chunks without any detrimental effect.

Proof of Proposition 3.3. The case $n = 1$ is trivial, so we assume that $n \geqslant 2$. We will use the strategy described in Section 3.3.

Step 1: compute ζ. Invoking Lemma 2.2, we compute an approximation $\tilde{\zeta} \in \tilde{\mathcal{D}}$ for $\zeta \coloneqq \zeta_n = e^{2\pi\mathrm{i}/n}$ with $\varepsilon(\tilde{\zeta}) < 2$ in time $O(p \lg^2 p) = O(\mathsf{M}_{M'}(p) \lg^2 p)$. Using the hypothesis $\lg^2 p < C'n$, this simplifies to $O(n\,\mathsf{M}_{M'}(p))$.

Step 2: compute powers of ζ. Define $\gamma_j \coloneqq \zeta^j \in \mathcal{D}$. We compute approximations $\tilde{\gamma}_j \in \tilde{\mathcal{D}}$ for $j = 0, 1, \ldots, n-1$ by taking $\tilde{\gamma}_0 \coloneqq 1$, $\tilde{\gamma}_1 \coloneqq \tilde{\zeta}$, and then repeatedly using Lemma 2.1 together with the identity $\gamma_j = \gamma_{j-1} \cdot \zeta$. This requires time $O(n\,\mathsf{M}_{M'}(p))$ (using the machine M'), and by induction we obtain $\varepsilon(\tilde{\gamma}_j) < 4j - 2$ for all $j = 1, \ldots, n-1$. We conclude that $\varepsilon(\tilde{\gamma}_j) < 4(n-1) - 2 = 4n - 6$ for all $j = 0, \ldots, n-1$.

Step 3: compute ω_k. Similarly, we compute approximations $\tilde{\omega}_k \in \tilde{\mathcal{D}}$ for $\omega_k \in \mathcal{D}$ (defined in (3.3)), for $k = 0, \ldots, n$, by taking $\tilde{\omega}_0 \coloneqq 1$ and then repeatedly applying the identity $\omega_k = \omega_{k-1} \cdot \gamma_{k-1}$. Again this requires time $O(n\,\mathsf{M}_{M'}(p))$, and we obtain by induction $\varepsilon(\tilde{\omega}_k) < 4k(n-1)$ for $k = 1, \ldots, n$. Therefore $\varepsilon(\tilde{\omega}_k) < 4n(n-1)$ for all $k = 0, \ldots, n$.

Step 4: compute B_r. We compute an approximation $\tilde{B} \in \tilde{\mathcal{D}}^{2n-1}$ for the vector $B \in \mathcal{D}^{2n-1}$ defined in (3.6). The entries $B_r = \omega_{r-n+1}$ are obtained directly from the output of Step 3, using the fact that $\omega_{-s} = \omega_{s+1}$ (since $\binom{-s}{2} = \binom{s+1}{2}$). This requires time $O(np) = O(n\,\mathsf{M}_{M'}(p))$, and we obtain $\varepsilon(\tilde{B}) < 4n(n-1)$.

Step 5: compute X'_s. We use Lemma 2.1 to compute an approximation $\tilde{X}' \in \tilde{\mathcal{D}}^n$ for $X' \in \mathcal{D}^n$ defined by $X'_s = \bar{\omega}_{-s} X_s$ (see (3.4)). This requires time $O(n\,\mathsf{M}_{M'}(p))$, and we obtain $\varepsilon(\tilde{X}') < \varepsilon(\tilde{X}) + 4n(n-1) + 2$.

Step 6: perform convolution. We apply Proposition 3.1 with $F \coloneqq X'$, $G \coloneqq B$, $H \coloneqq Y'$, to obtain an approximation $\tilde{Y}' \in \tilde{\mathcal{D}}^n$ for $Y' \in \mathcal{D}^n$, where the entries Y'_t are given by (3.7). This requires time $O(\mathsf{M}_M(m) + n\,\mathsf{M}_{M'}(p))$, and we obtain $\varepsilon(\tilde{Y}') < \varepsilon(\tilde{X}') + \varepsilon(\tilde{B}) + 2 < \varepsilon(\tilde{X}) + 8n(n-1) + 4$.

Step 7: compute Y_t. We finally approximate $Y \in \mathcal{D}^n$ via $Y_t = \bar{\omega}_t Y'_t$ (see (3.5)). Using Lemma 2.1 this requires time $O(n\,\mathsf{M}_{M'}(p))$, and we obtain $\varepsilon(\tilde{Y}) < \varepsilon(\tilde{Y}') + 4n(n-1) + 2 < \varepsilon(\tilde{X}) + 12n(n-1) + 6 < \varepsilon(\tilde{X}) + 12n^2$. □

Remark 3.5. The same algorithm works if we want to evaluate the transform $\frac{1}{n}\sum_{s=0}^{n-1} \zeta_n^{st} X_s$, i.e., replacing ζ_n^{-1} by ζ_n in (3.1). We use this fact several times below without further comment.

Remark 3.6. In the above proof, instead of computing the ω_k via multiplication, one could extract them from the list of ζ^j by means of a sorting algorithm. (As a side-effect, the numerical error would be reduced from $O(n^2)$ to $O(n)$.) However, the best complexity bound we know for this approach is $O(pn \lg n)$ via a merge sort [Knu98], introducing an unwanted $\lg n$ factor into the final result.

In the main reduction in Section 4, we will need to apply the Bluestein–Kronecker method to compute transforms along various one-dimensional "slices" of higher-dimensional arrays. In the remainder of this section we explain how to generalise Propositions 3.1 and 3.3 to handle this case.

3.5. **Conventions for arrays.** We will write $\mathbb{C}^{n_1,\dots,n_d}$ for the tensor product $\mathbb{C}^{n_1} \otimes \cdots \otimes \mathbb{C}^{n_d}$, i.e., the set of d-dimensional arrays of size $n_1 \times \cdots \times n_d$ with entries in $\mathbb{C}$. We write similarly $\mathcal{D}^{n_1,\dots,n_d}$ and $\tilde{\mathcal{D}}^{n_1,\dots,n_d}$ for the subsets of $\mathbb{C}^{n_1,\dots,n_d}$ with coefficients lying in $\mathcal{D}$ or $\tilde{\mathcal{D}}$. For an array $U \in \mathbb{C}^{n_1,\dots,n_d}$, we write $U_{s_1,\dots,s_d}$ for the entry at position $(s_1,\dots,s_d)$, where $0 \leqslant s_i < n_i$ for $i = 1,\dots,d$. The entries of an array $U \in \tilde{\mathcal{D}}^{n_1,\dots,n_d}$ are always stored on the tape in row-major order, i.e., as a list of length $n_1 \cdots n_d$, with $U_{s_1,\dots,s_d}$ stored at index $s_1(n_2 \cdots n_d) + s_2(n_3 \cdots n_d) + \cdots + s_{d-1}n_d + s_d$.

3.6. **Transforms along slices.** Let $l_1, l_2, n \geqslant 1$. Suppose that we are given an $l_1 \times n \times l_2$ array $X \in \mathcal{D}^{l_1,n,l_2}$, and we wish to compute the transform $Y \in \mathcal{D}^{l_1,n,l_2}$ given by

$$Y_{i_1,t,i_2} \coloneqq \frac{1}{n}\sum_{s=0}^{n-1} \zeta_n^{-st} X_{i_1,s,i_2}, \qquad 0 \leqslant i_1 < l_1,\ \ 0 \leqslant t < n,\ \ 0 \leqslant i_2 < l_2. \tag{3.8}$$

In other words, for each (i_1, i_2), we want to compute a transform of length n along the slice indexed by $(i_1, *, i_2)$. We cannot simply apply Proposition 3.3 to each slice separately, because that would require first transposing the data, which is precisely what we are trying to avoid. Instead, we will show how to encode all of the DFTs into a single large integer multiplication problem.

We begin by generalising Proposition 3.1 to handle this situation.

Proposition 3.7 (Convolutions along slices)**.** *Let $C, C' > 0$ be constants, and let M, M' be multiplication machines. Then there exists a Turing machine with*

the following properties. Its input consists of the precision parameter p, positive integers l_1, l_2, n, m such that

$$\lg n < Cp, \qquad \lg p < C' l_1 l_2 n, \qquad l_1 l_2 n p \leqslant m,$$

and approximations $\tilde{F} \in \tilde{\mathcal{D}}^{l_1,n,l_2}$ and $\tilde{G} \in \tilde{\mathcal{D}}^{2n-1}$ for an array $F \in \mathcal{D}^{l_1,n,l_2}$ and a vector $G \in \mathcal{D}^{2n-1}$. Define $H \in \mathcal{D}^{l_1,n,l_2}$ by the formula

$$H_{i_1,t,i_2} \coloneqq \frac{1}{n} \sum_{s=0}^{n-1} F_{i_1,s,i_2} G_{(t+n-1)-s}, \qquad 0 \leqslant i_1 < l_1, \;\; 0 \leqslant t < n, \;\; 0 \leqslant i_2 < l_2.$$

The output of the machine is an approximation $\tilde{H} \in \tilde{\mathcal{D}}^{l_1,n,l_2}$ such that $\varepsilon(\tilde{H}) < \varepsilon(\tilde{F}) + \varepsilon(\tilde{G}) + 2$. Its running time is

$$O(\mathsf{M}_M(m) + l_1 l_2 n \, \mathsf{M}_{M'}(p)).$$

Proof. The proof is similar to the proof of Proposition 3.1, working instead with the polynomials

$$f(x) \coloneqq \sum_{i_1=0}^{l_1-1} \sum_{s=0}^{n-1} \sum_{i_2=0}^{l_2-1} (2^p \tilde{F}_{i_1,s,i_2}) x^{3nl_2 i_1 + l_2 s + i_2} \in \mathbb{Z}[\mathrm{i}][x],$$
$$g(x) \coloneqq \sum_{j=0}^{2n-2} (2^p \tilde{G}_j) x^{l_2 j} \in \mathbb{Z}[\mathrm{i}][x].$$

We claim that the coefficients of the product $f(x)g(x)$ include the desired output coefficients, in the correct order. Indeed, we may rewrite $f(x)g(x)$ as

$$f(x)g(x) = \sum_{i_1=0}^{l_1-1} \sum_{i_2=0}^{l_2-1} \left(\sum_{s=0}^{n-1} \sum_{j=0}^{2n-2} (2^p \tilde{F}_{i_1,s,i_2})(2^p \tilde{G}_j) x^{l_2(s+j)} \right) x^{3nl_2 i_1 + i_2}.$$

The expression within the large parentheses is a polynomial of degree at most $3n-3$ in x^{l_2}. For each (i_1, i_2), it corresponds to the convolution of $F_{i_1,*,i_2}$ with G_*. The $x^{3nl_2 i_1 + i_2}$ term at the end ensures that these polynomials do not overlap in $f(x)g(x)$. (Actually, the algorithm would still work if 3 were replaced by 2, since we are only interested in the "middle third" of each block.)

Briefly, the algorithm runs as follows. We first evaluate at $x = 2^\beta$, taking $\beta \coloneqq 2p + \lg n + 2 = O(p)$. Evaluating $f(2^\beta)$ and $g(2^\beta)$ requires time $O(l_1 l_2 n p)$; the key point is that during evaluation of $f(2^\beta)$, the coefficients in the input array $\tilde{F}$ are already in the correct order for the desired concatenation. Computing the product $f(2^\beta)g(2^\beta)$ reduces to multiplying integers of bit size $O(l_1 l_2 n p) = O(m)$. Finally, during the decoding phase, the desired output approximations $\tilde{H}_{i_1,t,i_2}$ are already in the correct order. We omit the details of the rounding and error analysis, which are essentially identical to Proposition 3.1. □

Proposition 3.8 (DFTs along slices)**.** *Let $C, C' > 0$ be constants, and let M, M' be multiplication machines. Then there exists a Turing machine with the following properties. Its input consists of the precision parameter p, positive integers l_1, l_2, n, m such that*

$$\lg n < Cp, \qquad \lg^2 p < C' l_1 l_2 n, \qquad l_1 l_2 n p \leqslant m,$$

and an approximation $\tilde{X} \in \tilde{\mathcal{D}}^{l_1,n,l_2}$ *for an array* $X \in \mathcal{D}^{l_1,n,l_2}$. *Let* $Y \in \mathcal{D}^{l_1,n,l_2}$ *be the DFT of* X *along slices given by* (3.8). *The output of the machine is an approximation* $\tilde{Y} \in \tilde{\mathcal{D}}^{l_1,n,l_2}$ *such that* $\varepsilon(\tilde{Y}) < \varepsilon(\tilde{X}) + 12n^2$. *Its running time is*

$$O(\mathsf{M}_M(m) + l_1 l_2 n\, \mathsf{M}_{M'}(p)).$$

Proof. The algorithm is essentially the same as in the proof of Proposition 3.3. We first execute steps 1–4 (computing various roots of unity). Next, we run step 5 (computing X'_s from X_s) on each slice independently; this can be achieved by a single pass over the array. For step 6, we can perform the required convolutions on all slices simultaneously by replacing the use of Proposition 3.1 with Proposition 3.7. Finally, step 7 (computing Y_t from Y'_t) can again be run independently on all slices in a single pass. The error analysis is identical to the proof of Proposition 3.3. □

Remark 3.9. Proposition 3.8 may also be used to handle transforms along slices of higher-dimensional arrays. For example, if we have a 4-dimensional array of size $q_1 \times q_2 \times n \times q_3$, and we wish to transform with respect to the third coordinate, then we may reinterpret the data as a 3-dimensional array of size $(q_1 q_2) \times n \times q_3$, and invoke Proposition 3.8 with $l_1 \coloneqq q_1 q_2$ and $l_2 \coloneqq q_3$. The complexity in this case would be $O(\mathsf{M}_M(m) + q_1 q_2 q_3 n\, \mathsf{M}_{M'}(p))$.

4. The main reduction

In this section we present the main reduction from matrix transposition to integer multiplication, in the case that the matrix entries are neither too small nor too large relative to the matrix dimensions. For reasons that will become clear in Section 5, we actually consider the following more general transposition problem. Let $l_1, l_2, n_1, n_2, b \geqslant 1$, and consider a 4-dimensional array A of size

$$l_1 \times n_1 \times n_2 \times l_2,$$

with b-bit entries. We wish to transpose the n_1 and n_2 components, i.e., we want to compute the array A' of size

$$l_1 \times n_2 \times n_1 \times l_2$$

whose entries are given by

$$A'_{i_1,j_2,j_1,i_2} = A_{i_1,j_1,j_2,i_2} \qquad \text{for all } i_1, i_2, j_1, j_2.$$

Equivalently, one may think of the input as a list of l_1 arrays of size $n_1 \times n_2$ with $l_2 b$-bit entries, and the goal is to transpose each of these $n_1 \times n_2$ arrays. The case $l_1 = l_2 = 1$ corresponds to an ordinary $n_1 \times n_2$ matrix transposition with b-bit entries.

A machine T performing this type of transposition will be called a *generalised transposition machine*. It takes as input the parameters l_1, l_2, n_1, n_2, b, a positive integer m such that $l_1 l_2 n_1 n_2 b \leqslant m$, and the array to transpose. We denote its worst case running time by $\mathsf{T}_T(m; l_1, n_1, n_2, l_2, b)$.

Theorem 4.1. *Let* $C, C' > 0$ *be constants, and let* M, M' *be multiplication machines. Then there exists a generalised transposition machine* T *such that for any input parameters* l_1, l_2, n_1, n_2, b, m *satisfying*

$$\lg(n_1 n_2) < Cb, \qquad \lg^2 b < C' l_1 l_2 n_1 n_2, \qquad l_1 l_2 n_1 n_2 b \leqslant m, \tag{4.1}$$

we have

$$\mathsf{T}_T(m; l_1, n_1, n_2, l_2, b) = O(\mathsf{M}_M(m) + l_1 l_2 n_1 n_2\, \mathsf{M}_{M'}(b)).$$

Proof. We are given as input an array A of size $l_1 \times n_1 \times n_2 \times l_2$, whose entries are b-bit strings. Let us write $n := n_1 n_2$ and $l := l_1 l_2$. If either $n_1 = 1$ or $n_2 = 1$, then there is nothing to do, so we may assume that $n_1, n_2 \geqslant 2$.

In the following discussion, we frequently deal with array entries whose first coordinate is $i_1 \in \{0, \ldots, l_1 - 1\}$ and last coordinate is $i_2 \in \{0, \ldots, l_2 - 1\}$. For improved legibility we will place these variables in superscripts instead of subscripts. For example, we will write $A^{i_1,i_2}_{j_1,j_2}$ to mean A_{i_1,j_1,j_2,i_2}. The reader should keep in mind that the arrangement of data on the tape is still of course given by the row-major ordering with respect to (i_1, j_1, j_2, i_2).

Step 1: encode input data. We interpret the entries of A as integers in the interval $0 \leqslant A^{i_1,i_2}_{j_1,j_2} < 2^b$. Introducing the fixed-point framework of Section 2, with precision

$$p := b + \lg(80n^3) = O(b),$$

we encode these integers into a 3-dimensional array $R \in \tilde{\mathcal{D}}^{l_1,n,l_2}$ given by

$$R^{i_1,i_2}_{j_1 n_2 + j_2} := 2^{-b} A^{i_1,i_2}_{j_1,j_2} \in \tilde{\mathcal{D}}, \qquad 0 \leqslant j_1 < n_1, \;\; 0 \leqslant j_2 < n_2.$$

This encoding amounts to zero-padding, and can be performed in time $O(lnp) = O(m)$.

Step 2: forward DFTs. For each (i_1, i_2), consider the scaled DFT

$$S^{i_1,i_2}_k := \frac{1}{n} \sum_{j=0}^{n-1} \zeta_n^{-jk} R^{i_1,i_2}_j \in \mathcal{D}, \qquad 0 \leqslant k < n.$$

According to Proposition 3.8, we may compute an approximation $\tilde{S} \in \mathcal{D}^{l_1,n,l_2}$ with $\varepsilon(\tilde{S}) < 12n^2$ in time $O(\mathsf{M}_M(m') + ln\,\mathsf{M}_{M'}(p))$ where $m' := l_1 l_2 n_1 n_2 p$. Moreover, since $p = O(b)$ and $m' = O(m)$, we may use Lemma 3.2 to modify the construction in Proposition 3.8 slightly, replacing all calls to M' and M with size parameters respectively p and m' by several invocations with size parameters b and m. The complexity of computing $\tilde{S}$ this way then becomes $O(\mathsf{M}_M(m) + ln\,\mathsf{M}_{M'}(b))$ as desired. In the rest of this proof, we will replace p and m' by b and m in this way without further comment.

The aim of the remaining steps is to compute the inverse DFTs of the slices of S, i.e., to recover the coefficients of R. Recall (see (3.2)) that these are given by

$$R^{i_1,i_2}_j = \sum_{k=0}^{n-1} \zeta_n^{jk} S^{i_1,i_2}_k, \qquad 0 \leqslant j < n. \tag{4.2}$$

However, we will use a different algorithm to compute the inverse DFT, so that the coefficients of R are produced in a different order, corresponding to the desired transpose of the original input matrix A.

We begin by writing $j = j_1 n_2 + j_2$ and $k = k_2 n_1 + k_1$ with $0 \leqslant j_1, k_1 < n_1$ and $0 \leqslant j_2, k_2 < n_2$, to obtain the well-known Cooley–Tukey decomposition of (4.2):

$$R^{i_1,i_2}_{j_1 n_2 + j_2} = \sum_{k_1=0}^{n_1-1} \zeta_{n_1}^{j_1 k_1} \cdot \zeta_n^{j_2 k_1} \sum_{k_2=0}^{n_2-1} \zeta_{n_2}^{j_2 k_2} S^{i_1,i_2}_{k_2 n_1 + k_1}.$$

Let us define intermediate quantities $U, V, W \in \mathcal{D}^{l_1,n_2,n_1,l_2}$ by

$$U^{i_1,i_2}_{j_2,k_1} \coloneqq \frac{1}{n_2}\sum_{k_2=0}^{n_2-1} \zeta_{n_2}^{j_2k_2} S^{i_1,i_2}_{k_2n_1+k_1} \in \mathcal{D}, \qquad 0 \leqslant j_2 < n_2,\ \ 0 \leqslant k_1 < n_1,$$

$$V^{i_1,i_2}_{j_2,k_1} \coloneqq \zeta_n^{j_2k_1} U^{i_1,i_2}_{j_2,k_1} \in \mathcal{D}, \qquad 0 \leqslant j_2 < n_2,\ \ 0 \leqslant k_1 < n_1,$$

$$W^{i_1,i_2}_{j_2,j_1} \coloneqq \frac{1}{n_1}\sum_{k_1=0}^{n_1-1} \zeta_{n_1}^{j_1k_1} V^{i_1,i_2}_{j_2,k_1} \in \mathcal{D}, \qquad 0 \leqslant j_2 < n_2,\ \ 0 \leqslant j_1 < n_1,$$

so that

$$R^{i_1,i_2}_{j_1n_2+j_2} = nW^{i_1,i_2}_{j_2,j_1}.$$

In the next three steps, we compute approximations for U, V and W in turn.

Step 3: inverse DFTs in n_2 direction. Reinterpreting S as an array of size $l_1 \times n_2 \times n_1 \times l_2$, i.e., as an element of $\mathcal{D}^{l_1,n_2,n_1,l_2}$, the above formula for $U^{i_1,i_2}_{j_2,k_1}$ amounts to transforming slices of S in the n_2 direction. Therefore, by Proposition 3.8 and Lemma 3.2 (see also Remark 3.9), we may compute an approximation $\tilde{U} \in \tilde{\mathcal{D}}^{l_1,n_2,n_1,l_2}$ for U with $\varepsilon(\tilde{U}) < \varepsilon(\tilde{S}) + 12n_2^2 < 24n^2$ in time $O(\mathsf{M}_M(m) + ln\,\mathsf{M}_{M'}(b))$.

Step 4: apply twiddle factors. Let $\gamma_{j_2,k_1} \coloneqq \zeta_n^{j_2k_1}$. Following the method of the proof of Proposition 3.3, we first compute approximations for $\gamma_{1,k_1} = \zeta_n^{k_1}$ for $k_1 = 0, \ldots, n_1 - 1$ with $\varepsilon(\tilde{\gamma}_{1,k_1}) < 4n_1 - 6$. We then similarly compute approximations for $\gamma_{j_2,k_1} = (\gamma_{1,k_1})^{j_2}$, for all k_1 and j_2, with $\varepsilon(\tilde{\gamma}_{j_2,k_1}) < 4(n_1-1)(n_2-1) - 2$. Finally we use Lemma 2.1 and (Lemma 3.2) to compute approximations for $V^{i_1,i_2}_{j_2,k_1} = \gamma_{j_2,k_1} U^{i_1,i_2}_{j_2,k_1}$, with $\varepsilon(\tilde{V}) < 4(n_1-1)(n_2-1) - 2 + \varepsilon(\tilde{U}) + 2 < 28n^2$. Altogether this requires time $O(ln\,\mathsf{M}_{M'}(b))$.

Step 5: inverse DFTs in n_1 direction. The definition of $W^{i_1,i_2}_{j_2,j_1}$ amounts to transforming the V array in the n_1 direction. Again by Proposition 3.8 we may compute an approximation $\tilde{W} \in \tilde{\mathcal{D}}^{l_1,n_2,n_1,l_2}$ with $\varepsilon(\tilde{W}) < \varepsilon(\tilde{V}) + 12n_1^2 < 40n^2$ in time $O(\mathsf{M}_M(m) + ln\,\mathsf{M}_{M'}(b))$.

Step 6: scale and round. The above estimates show that

$$\begin{aligned}|2^b n\tilde{W}^{i_1,i_2}_{j_2,j_1} - A^{i_1,i_2}_{j_1,j_2}| &= 2^b|n\tilde{W}^{i_1,i_2}_{j_2,j_1} - R^{i_1,i_2}_{j_1n_2+j_2}| \\ &= 2^b n|\tilde{W}^{i_1,i_2}_{j_2,j_1} - W^{i_1,i_2}_{j_2,j_1}| \leqslant 2^{b-p}n \cdot \varepsilon(\tilde{W}) < 2^{b-p} \cdot 40n^3 \leqslant \tfrac{1}{2}.\end{aligned}$$

In other words, the nearest integer to $2^b n\tilde{W}^{i_1,i_2}_{j_2,j_1}$ is exactly $A^{i_1,i_2}_{j_1,j_2}$. We may therefore recover $A^{i_1,i_2}_{j_1,j_2}$ by first multiplying $2^p\tilde{W}^{i_1,i_2}_{j_2,j_1} \in \mathbb{Z}[\mathrm{i}]$ by n, and then dividing by 2^{p-b} and rounding to the nearest integer (the imaginary part may be discarded). The cost is $O(\mathsf{M}_{M'}(b))$ per coefficient, and we have finally succeeded in transposing the original A array. □

5. Decomposing transpositions into subproblems

In this section we describe a method for decomposing an $n_1 \times n_2$ transposition problem into smaller problems, by cutting n_1 and/or n_2 into chunks. We use the following terminology: a *problem of type* $(l_1, n_1, n_2, l_2; b)$ is a generalised transposition problem (see Section 4) of size $l_1 \times n_1 \times n_2 \times l_2$ with b-bit coefficients.

Lemma 5.1. *Let $n_1, n_2, n_1', n_2', b \geqslant 1$, and assume that $n_1' \mid n_1$ and $n_2' \mid n_2$. Then an $n_1 \times n_2$ transposition problem with b-bit entries may be decomposed into:*

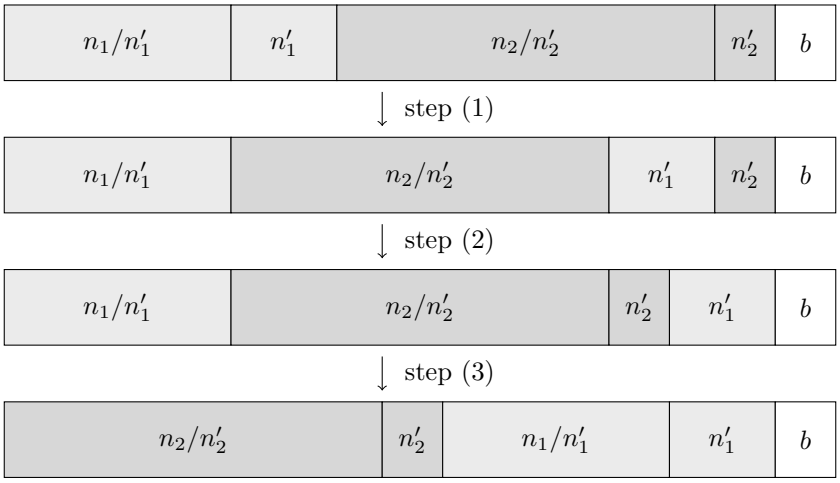

FIGURE 1. Decomposing a transposition (Lemma 5.1).

(1) *a problem of type* $\left(\frac{n_1}{n_1'}, n_1', \frac{n_2}{n_2'}, n_2'; b\right)$, *followed by*
(2) *a problem of type* $\left(\frac{n_1 n_2}{n_1' n_2'}, n_1', n_2', 1; b\right)$, *followed by*
(3) *a problem of type* $\left(1, \frac{n_1}{n_1'}, n_2, n_1'; b\right)$.

Proof. The decomposition is illustrated in Figure 1. We begin with an $n_1 \times n_2$ matrix A, where the entry A_{i_1,i_2} is stored at index $i_1 n_2 + i_2$. Let

$$\begin{aligned} i_1 &= j_1 n_1' + k_1, \qquad 0 \leqslant k_1 < n_1', \quad 0 \leqslant j_1 < n_1/n_1', \\ i_2 &= j_2 n_2' + k_2, \qquad 0 \leqslant k_2 < n_2', \quad 0 \leqslant j_2 < n_2/n_2'. \end{aligned}$$

Then the index $i_1 n_2 + i_2$ may be rewritten as

$$j_1(n_1' n_2) + k_1 n_2 + j_2 n_2' + k_2.$$

In step (1), we exchange the k_1 and j_2 coordinates. After this, the A_{i_1,i_2} entry is stored at index

$$j_1(n_1' n_2) + j_2(n_1' n_2') + k_1 n_2' + k_2.$$

In step (2), we exchange the k_1 and k_2 coordinates. After this, the A_{i_1,i_2} entry is stored at index

$$\begin{aligned} &j_1(n_1' n_2) + j_2(n_1' n_2') + k_2 n_1' + k_1 \\ &= j_1(n_1' n_2) + i_2 n_1' + k_1. \end{aligned}$$

In step (3), we exchange the j_1 and i_2 coordinates. After this, the A_{i_1,i_2} entry is stored at index

$$\begin{aligned} &i_2 n_1 + j_1 n_1' + k_1 \\ &= i_2 n_1 + i_1, \end{aligned}$$

and we have succeeded in transposing the array. □

The next result analyses a Turing machine implementation of the decomposition of Lemma 5.1. We focus on a special case in which n_1' and n_2' are taken to be exponentially smaller than $\max(n_1, n_2)$. The basic idea will be to reduce steps (1) and (3) (the "outer transpositions") to integer multiplication problems via Theorem 4.1, and to delegate step (2) (the "inner transposition") to some other transposition machine T', which will be specified later.

For technical reasons it will be convenient to work with power-of-two parameter sizes. A *dyadic transposition machine* is a machine having the same interface as an ordinary transposition machine, but with the additional restriction that the parameters n_1, n_2, b must lie in $2^{\mathbb{N}} \coloneqq \{1, 2, 4, \ldots\}$.

Proposition 5.2. *Let M, M' be multiplication machines, and let T' be a dyadic transposition machine. Then there exists a dyadic transposition machine T with the following properties.*

Let $n_1, n_2, b \in 2^{\mathbb{N}}$ and $m \geqslant 1$, and assume that $n_1 n_2 b \leqslant m$. Let

$$s \coloneqq 2^{\lg \lg \max(n_1, n_2)},$$

and define

$$n_1' \coloneqq \min(n_1, s), \qquad n_2' \coloneqq \min(n_2, s), \qquad m' \coloneqq n_1' n_2' b.$$

Then

$$\mathsf{T}_T(m; n_1, n_2, b) < \frac{n_1 n_2}{n_1' n_2'} \mathsf{T}_{T'}(m'; n_1', n_2', b) + O\left(\mathsf{M}_M(m) + \frac{m}{s} \mathsf{M}_{M'}(s)\right).$$

Proof. Clearly $n_1' \mid n_1$, since $n_1' \leqslant n_1$ and $n_1' \in 2^{\mathbb{N}}$. Similarly $n_2' \mid n_2$. Let us now consider in turn each step from Lemma 5.1.

Step (1). If $s \geqslant n_2$ then $n_2' = n_2$, so this step becomes trivial. Assume therefore that $s < n_2$, so that $n_2' = s$, and we are dealing with a problem of type

$$\left(\frac{n_1}{n_1'}, n_1', \frac{n_2}{s}, s; b\right).$$

This is equivalent to a problem of type

$$\left(\frac{n_1}{n_1'}, n_1', \frac{n_2}{s}, b; s\right),$$

i.e., by simply reinterpreting each length-s array of b-bit entries as a length-b array of s-bit entries. We will effect this transposition by invoking Theorem 4.1 with the parameters

$$(l_1, n_1, n_2, l_2, b) \coloneqq \left(\frac{n_1}{n_1'}, n_1', \frac{n_2}{s}, b, s\right)$$

(and $m \coloneqq m$, $M \coloneqq M$, $M' \coloneqq M'$). Let us verify the hypotheses (4.1) for these parameters. The first inequality states that $\lg(n_1'(n_2/s)) < Cs$ for a suitable constant $C > 0$. This holds as

$$\lg\left(n_1' \frac{n_2}{s}\right) \leqslant \lg(n_1 n_2) \leqslant 2 \lg \max(n_1, n_2) \leqslant 2s.$$

For the second inequality, we must show that

$$\lg^2 s < C' \frac{n_1}{n_1'} n_1' \frac{n_2}{s} b = C' n_1 n_2 b / s$$

for suitable C'. Writing $r \coloneqq \max(n_1, n_2)$, we have $s = 2^{\lg\lg r} \leqslant 2\lg r$ and $\lg s = \lg\lg r$, so

$$s\lg^2 s \leqslant (2\lg r)(\lg\lg r)^2 < C'r \leqslant C'n_1n_2b$$

for large enough C' (in fact $C' = 6$ will do). The third inequality is clear. According to Theorem 4.1, the cost of this transposition is

$$O\left(\mathsf{M}_M(m) + \frac{n_1}{n_1'}n_1'\frac{n_2}{s}b\,\mathsf{M}_{M'}(s)\right) = O\left(\mathsf{M}_M(m) + \frac{m}{s}\,\mathsf{M}_{M'}(s)\right).$$

Step (2). This problem is equivalent to $n_1n_2/n_1'n_2'$ separate two-dimensional $n_1' \times n_2'$ transposition problems with b-bit coefficients. Using T', the cost is

$$\frac{n_1n_2}{n_1'n_2'}\,\mathsf{T}_{T'}(m'; n_1', n_2', b).$$

Step (3). If $s \geqslant n_1$ then $n_1' = n_1$ and the problem is trivial, so assume that $s < n_1$. Then $n_1' = s$ and we face a problem of type

$$\left(1, \frac{n_1}{s}, n_2, s; b\right).$$

As in step (1), we may reinterpret this as a problem of type

$$\left(1, \frac{n_1}{s}, n_2, b; s\right).$$

Again we invoke Theorem 4.1, this time with parameters

$$(l_1, n_1, n_2, l_2, b) \coloneqq \left(1, \frac{n_1}{s}, n_2, b, s\right).$$

To see that the hypotheses (4.1) are satisfied, observe that

$$\lg\left(\frac{n_1}{s}n_2\right) \leqslant \lg(n_1n_2) \leqslant 2s,$$

and by the same argument as before,

$$\lg^2 s < C'n_1n_2b/s = C'\frac{n_1}{s}n_2b.$$

The cost of this transposition is therefore given by

$$O\left(\mathsf{M}_M(m) + \frac{n_1}{s}n_2b\,\mathsf{M}_{M'}(s)\right) = O\left(\mathsf{M}_M(m) + \frac{m}{s}\,\mathsf{M}_{M'}(s)\right). \qquad \square$$

6. Proofs of main results

In this section we prove the main results stated in Section 1.2.

6.1. **Proof of Theorem 1.2.** We first establish the following reduction from transposition to multiplication.

Proposition 6.1. *Let M be a multiplication machine. Then there exists a dyadic transposition machine T such that*

$$\mathsf{T}_T(m; n_1, n_2, b) = O(\mathsf{M}_M(m) + m\lg\lg\max(n_1, n_2)).$$

Proof. We apply Proposition 5.2 with $M' \coloneqq M^*$ and $T' \coloneqq T^*$. By (1.1) and (1.2), the resulting dyadic transposition machine T satisfies

$$\begin{aligned}\mathsf{T}_T(m; n_1, n_2, b) &< \frac{n_1 n_2}{n_1' n_2'} \mathsf{T}_{T^*}(m'; n_1', n_2', b) + O\left(\mathsf{M}_M(m) + \frac{m}{s}\mathsf{M}_{M^*}(s)\right) \\ &= O\left(n_1 n_2 b \lg \min(n_1', n_2') + \mathsf{M}_M(m) + m \lg s\right),\end{aligned}$$

where n_1', n_2', s, m' are defined as in Proposition 5.2. Since $n_i' \leqslant s$, $n_1 n_2 b \leqslant m$ and $\lg s = \lg\lg \max(n_1, n_2)$,

$$\mathsf{T}_T(m; n_1, n_2, b) = O(\mathsf{M}_M(m) + m \lg\lg \max(n_1, n_2)). \qquad \square$$

It is straightforward to remove the dyadic restriction:

Corollary 6.2. *Let M be a multiplication machine. Then there exists an (ordinary non-dyadic) transposition machine T such that*

$$\mathsf{T}_T(m; n_1, n_2, b) = O(\mathsf{M}_M(m) + m \lg\lg \max(n_1, n_2)).$$

Proof. Let $n_1, n_2, b, m \geqslant 1$ be arbitrary input parameters such that $n_1 n_2 b \leqslant m$. We round up the parameters to powers of two, setting

$$\tilde{n}_1 \coloneqq 2^{\lceil \log_2 n_1 \rceil}, \qquad \tilde{n}_2 \coloneqq 2^{\lceil \log_2 n_2 \rceil}, \qquad \tilde{b} \coloneqq 2^{\lceil \log_2 b \rceil}.$$

Let $\tilde{m} \coloneqq 8m$ so that $\tilde{n}_1 \tilde{n}_2 \tilde{b} \leqslant \tilde{m}$.

We first copy the $n_1 \times n_2$ input matrix with b-bit entries into an $\tilde{n}_1 \times \tilde{n}_2$ matrix with $\tilde{b}$-bit entries, by suitable zero-padding. Clearly this can be done in time $O(m)$. We then invoke the machine from Proposition 6.1 with parameters $\tilde{n}_1, \tilde{n}_2, \tilde{b}, \tilde{m}$ to transpose this matrix. This requires time

$$O(\mathsf{M}_M(\tilde{m}) + \tilde{m} \lg\lg \max(\tilde{n}_1, \tilde{n}_2)) = O(\mathsf{M}_M(8m) + m \lg\lg \max(n_1, n_2)).$$

Finally we copy everything back to the desired $n_2 \times n_1$ output matrix (with b-bit entries) in time $O(m)$.

To get the desired complexity bound, we make one more small change: using Lemma 3.2, we replace all calls to M with size parameter $8m$ by several calls with m instead. The resulting machine T satisfies

$$\mathsf{T}_T(m; n_1, n_2, b) = O(\mathsf{M}_M(m) + m \lg\lg \max(n_1, n_2)). \qquad \square$$

Specialising to the binary case, and restating in terms of lower bounds, we obtain the following.

Corollary 6.3. *Assume that*

$$\mathsf{T}_T(m) = \omega(m \lg\lg m)$$

for every binary transposition machine T. Then for any multiplication machine M, there exists a binary transposition machine T such that

$$\mathsf{M}_M(m) = \Omega(\mathsf{T}_T(m)).$$

Proof. Taking $n_1, n_2 \coloneqq \lfloor m^{1/2} \rfloor$ and $b \coloneqq 1$ in Corollary 6.2, we get a binary transposition machine satisfying

$$\mathsf{T}_T(m) = O(\mathsf{M}_M(m) + m \lg\lg m).$$

Therefore, there is an absolute constant $C > 0$ such that

$$\mathsf{T}_T(m) < C(\mathsf{M}_M(m) + m \lg\lg m)$$

for all $m \geqslant 1$, and hence

$$\mathsf{M}_M(m) > \frac{\mathsf{T}_T(m)}{C} - m \lg\lg m = \left(\frac{1}{C} - \frac{m \lg\lg m}{\mathsf{T}_T(m)}\right) \mathsf{T}_T(m)$$

for all $m \geqslant 1$. Since we assumed that $\mathsf{T}_T(m) = \omega(m \lg\lg m)$, we conclude that in fact $\mathsf{M}_M(m) = \Omega(\mathsf{T}_T(m))$. □

Theorem 1.2 now follows immediately from Corollary 6.3.

Remark 6.4. The $m \lg\lg m$ term above arises from the use of the folklore transposition algorithm to handle the "inner" transpositions of size exponentially smaller than m. In the following sections we will iterate the decomposition strategy from Section 5, in an attempt to make the innermost transpositions even smaller, and hence further reduce the size of the $m \lg\lg m$ term.

6.2. **Proof of Theorem 1.3.** For any multiplication machine M, define

$$\overline{\mathsf{M}}_M(m) \coloneqq m \cdot \max_{k \leqslant m} \frac{\mathsf{M}_M(k)}{k}, \qquad m \geqslant 1.$$

Observe that by construction, $\overline{\mathsf{M}}_M(m)/m$ is non-decreasing in m.

Remark 6.5. If $\mathsf{M}_M(m)/m$ is "non-decreasing up to a constant", i.e., there exists a constant $C \geqslant 1$ such that $\mathsf{M}_M(m')/m' \leqslant C\,\mathsf{M}_M(m)/m$ for all $m' \leqslant m$, which is the case for all "reasonable" multiplication algorithms known to the authors, then $\overline{\mathsf{M}}_M(m) = O(\mathsf{M}_M(m))$.

We now have the following variant of Proposition 6.1, which has a smaller error term, but at the cost of replacing $\mathsf{M}(m)$ by $\overline{\mathsf{M}}(m)$.

Proposition 6.6. *Fix an integer $\ell \geqslant 2$, and let M be a multiplication machine. Then there exists a dyadic transposition machine T such that*

$$\mathsf{T}_T(m; n_1, n_2, b) = O(\overline{\mathsf{M}}_M(m) + m \lg^{\circ \ell} \max(n_1, n_2)).$$

Proof. The idea is to chain together $\ell - 1$ invocations of Proposition 5.2. We start by defining $T_\ell \coloneqq T^*$. Then for $j = \ell - 1, \ldots, 1$ in turn, we apply Proposition 5.2 with $T' \coloneqq T_{j+1}$ and $M' \coloneqq M$ to obtain a new machine T_j. We claim that the final top-level machine T_1 satisfies the desired bound

$$\mathsf{T}_{T_1}(m; n_1, n_2, b) = O(\overline{\mathsf{M}}_M(m) + m \lg^{\circ \ell} \max(n_1, n_2)).$$

To prove this, let $n_1, n_2, b \in 2^{\mathbb{N}}$ and $m \geqslant 1$ be the input parameters for T_1, with $n_1 n_2 b \leqslant m$. For $j = 1, \ldots, \ell$, let $n_1^{(j)}$, $n_2^{(j)}$, $m^{(j)}$ be the parameters subsequently passed to T_j as we proceed down the call chain; in particular, $n_i^{(1)} = n_i$. For $j = 1, \ldots, \ell - 1$ let $s^{(j)}$ be the corresponding value of s as in the statement of Proposition 5.2.

Let us show that

$$\lg \max(n_1^{(j)}, n_2^{(j)}) = \lg^{\circ j} \max(n_1, n_2), \qquad j = 1, \ldots, \ell. \tag{6.1}$$

Suppose first that $n_1 \geqslant n_2$. It follows that $n_1^{(j)} \geqslant n_2^{(j)}$ for all $j = 1, \ldots, \ell$, since by induction we have $n_1^{(j+1)} = \min(n_1^{(j)}, s^{(j)}) \geqslant \min(n_2^{(j)}, s^{(j)}) = n_2^{(j+1)}$. Hence

$$s^{(j)} = 2^{\lg\lg \max(n_1^{(j)}, n_2^{(j)})} = 2^{\lg\lg n_1^{(j)}}$$

and

$$\lg n_1^{(j+1)} = \lg \min(n_1^{(j)}, s^{(j)}) = \min(\lg n_1^{(j)}, \lg\lg n_1^{(j)}) = \lg\lg n_1^{(j)}$$

for $j = 1, \dots, \ell - 1$. By induction $\lg n_1^{(j)} = \lg^{\circ j} n_1$ for $j = 1, \dots, \ell$, so (6.1) holds. If $n_2 \geqslant n_1$, a symmetrical argument applies.

Now we estimate the complexity. For each $j = 1, \dots, \ell - 1$ we have

$$\mathsf{T}_{T_j}\left(m^{(j)}; n_1^{(j)}, n_2^{(j)}, b\right) < \frac{n_1^{(j)} n_2^{(j)}}{n_1^{(j+1)} n_2^{(j+1)}} \mathsf{T}_{T_{j+1}}\left(m^{(j+1)}; n_1^{(j+1)}, n_2^{(j+1)}, b\right) + O\left(\mathsf{M}_M(m^{(j)}) + \frac{m^{(j)}}{s^{(j)}} \mathsf{M}_M(s^{(j)})\right).$$

Combining these inequalities, we obtain

$$\mathsf{T}_{T_1}(m; n_1, n_2, b) < \frac{n_1 n_2}{n_1^{(\ell)} n_2^{(\ell)}} \mathsf{T}_{T_\ell}\left(m^{(\ell)}; n_1^{(\ell)}, n_2^{(\ell)}, b\right) + \sum_{j=1}^{\ell-1} O\left(\frac{n_1 n_2}{n_1^{(j)} n_2^{(j)}}\left(\mathsf{M}_M(m^{(j)}) + \frac{m^{(j)}}{s^{(j)}} \mathsf{M}_M(s^{(j)})\right)\right).$$

Since $T_\ell = T^*$, by (6.1) the first term becomes

$$\frac{n_1 n_2}{n_1^{(\ell)} n_2^{(\ell)}} \cdot O\left(n_1^{(\ell)} n_2^{(\ell)} b \lg \min(n_1^{(\ell)}, n_2^{(\ell)})\right) = O(m \lg \max(n_1^{(\ell)}, n_2^{(\ell)})) = O(m \lg^{\circ \ell} \max(n_1, n_2)).$$

For the second term, first observe that

$$\frac{n_1 n_2}{n_1^{(j)} n_2^{(j)}} \leqslant \frac{m}{m^{(j)}}, \qquad j = 1, \dots, \ell - 1.$$

Indeed, for $j = 1$ the inequality holds as both sides are equal to 1, and for $j > 1$ it holds as $m^{(j)} = n_1^{(j)} n_2^{(j)} b$ and $n_1 n_2 b \leqslant m$. Therefore the second term becomes

$$\sum_{j=1}^{\ell-1} O\left(m \frac{\mathsf{M}_M(m^{(j)})}{m^{(j)}} + m \frac{\mathsf{M}_M(s^{(j)})}{s^{(j)}}\right). \tag{6.2}$$

We now claim that $m^{(j)} \leqslant m$ and $s^{(j)} \leqslant m$ for all $j < \ell$. The first inequality is trivial for $j = 1$, and for $j > 1$ we have $m^{(j)} = n_1^{(j)} n_2^{(j)} b \leqslant n_1 n_2 b \leqslant m$ since clearly $n_i^{(j)} \leqslant n_i$. For the second inequality, since $2^{\lg\lg n} \leqslant n$ for all $n \geqslant 2$,

$$s^{(j)} = 2^{\lg\lg \max(n_1^{(j)}, n_2^{(j)})} \leqslant 2^{\lg\lg \max(n_1, n_2)} \leqslant 2^{\lg\lg(n_1 n_2)} \leqslant n_1 n_2 \leqslant m$$

for $j < \ell$. (Strictly speaking this argument only works for $n_1 n_2 \geqslant 2$; we ignore the trivial case $n_1 = n_2 = 1$.)

Therefore (6.2) becomes $\sum_{j=1}^{\ell-1} O(\overline{\mathsf{M}}_M(m))$, and since ℓ is fixed, this simplifies further to $O(\overline{\mathsf{M}}_M(m))$. We conclude that

$$\mathsf{T}_{T_1}(m; n_1, n_2, b) = O(\overline{\mathsf{M}}_M(m) + m \lg^{\circ \ell} \max(n_1, n_2)). \qquad \square$$

Again it is easy to drop the dyadic restriction:

Corollary 6.7. *Fix an integer $\ell \geqslant 2$, and let M be a multiplication machine. Then there exists an (ordinary, non-dyadic) transposition machine T such that*

$$\mathsf{T}_T(m; n_1, n_2, b) = O(\overline{\mathsf{M}}_M(m) + m \lg^{\circ \ell} \max(n_1, n_2)).$$

Proof. As in the proof of Corollary 6.2, we use a zero-padding strategy to reduce to the dyadic case (Proposition 6.6), and then we use Lemma 3.2 to replace every call to M with size parameter $\geqslant m$ by $O(1)$ calls with size m. (To be completely pedantic, this requires modifying the transposition machine interface slightly to enable the top-level parameter m to be passed down the call chain, at least for the top few levels. This does not affect the complexity.) □

Proof of Theorem 1.3. Suppose that we are given a multiplication machine M such that $\mathsf{M}_M(m) = O(m \lg^{\circ \ell} m)$. Then

$$\overline{\mathsf{M}}_M(m) = m \cdot \max_{k \leqslant m} \frac{\mathsf{M}_M(k)}{k} = m \cdot \max_{k \leqslant m} O(\lg^{\circ \ell} k) = O(m \lg^{\circ \ell} m). \tag{6.3}$$

Taking $n_1, n_2 \coloneqq \lfloor m^{1/2} \rfloor$ and $b \coloneqq 1$ in Corollary 6.7, we get a binary transposition machine T satisfying $\mathsf{T}_T(m) = O(m \lg^{\circ \ell} m)$. But this contradicts the hypothesis that no such machine exists. □

6.3. **Proof of Theorem 1.4.** The following reduction may be regarded as a more aggressive version of Proposition 6.6.

Proposition 6.8. *Let M be a multiplication machine. Then there exists a dyadic transposition machine T such that*

$$\mathsf{T}_T(m; n_1, n_2, b) = O(\ell_0 \, \overline{\mathsf{M}}(m)), \qquad \ell_0 \coloneqq \max(1, \lg^* \max(n_1, n_2) - \lg^* b).$$

Comparing with Proposition 6.6, we see that the error term has been completely eliminated, but we pay with a factor of ℓ_0 in the main term.

Proof. The argument is similar to the proof of Proposition 6.6, but instead of constructing a sequence of machines, we must construct a single machine that calls itself recursively, where the number of recursion levels depends on the relative sizes of $\max(n_1, n_2)$ and b.

The base case. We first describe a machine T' that handles the base case of the recursion. Instead of calling the folklore algorithm T^* as was done in the proof of Proposition 6.6, we will construct T' via Theorem 4.1. The machine T' has the same interface as a dyadic transposition machine, i.e., the input consists of integers $n_1, n_2, b \in 2^{\mathbb{N}}$ and $m \geqslant 1$ such that $n_1 n_2 b \leqslant m$, but we impose the additional constraint that

$$\lg \max(n_1, n_2) \leqslant b.$$

Let $s \coloneqq 2^{\lg \lg \max(n_1, n_2)}$. Then s is a power of two and $\lg s \leqslant \lg b$. Since b is also a power of two, this implies that $s \mid b$. We perform the transposition by invoking Theorem 4.1 with $M' \coloneqq M$ and with parameters

$$(l_1, n_1, n_2, l_2, b) \coloneqq \left(1, n_1, n_2, \frac{b}{s}, s\right),$$

i.e., we reinterpret the $n_1 \times n_2$ input matrix with b-bit entries as an array of size $1 \times n_1 \times n_2 \times (b/s)$ with s-bit entries. Let us verify the hypotheses in (4.1). The first inequality states that $\lg(n_1 n_2) < Cs$ for suitable $C > 0$; this holds as $\lg(n_1 n_2) \leqslant 2 \lg \max(n_1, n_2) \leqslant 2s$. For the second inequality we must prove that $\lg^2 s < C' n_1 n_2 b / s$; this follows by a similar argument to step (1) in the proof of Proposition 5.2. The cost of this transposition is thus

$$O\left(\mathsf{M}_M(m) + n_1 n_2 \frac{b}{s} \mathsf{M}_M(s)\right) = O\left(\mathsf{M}_M(m) + \frac{m}{s} \mathsf{M}_M(s)\right).$$

Since $s \leqslant b \leqslant m$, this bound simplifies to $O(\overline{\mathsf{M}}_M(m))$.

(We remark that it would not have worked to invoke Theorem 4.1 directly for the original problem of type $(1, n_1, n_2, 1; b)$, because the second inequality in (4.1) might fail if b is much larger than n_1 and n_2. This is the reason for introducing the auxiliary parameter s, and indirectly the reason for requiring b to be a power of two.)

The recursive case. We now describe the main dyadic transposition machine T, which takes as input any $n_1, n_2, b \in 2^{\mathbb{N}}$ and $m \geqslant 1$ such that $n_1 n_2 b \leqslant m$.

If $\lg \max(n_1, n_2) > b$, we use the same strategy as in Proposition 5.2 (taking $M' \coloneqq M$), i.e., we set

$$s \coloneqq 2^{\lg \lg \max(n_1, n_2)}, \qquad n_i' \coloneqq \min(n_i, s), \qquad m' \coloneqq n_1' n_2' b,$$

and decompose into $n_1 n_2 / n_1' n_2'$ transpositions of size $n_1' \times n_2'$, plus additional work of cost

$$O\left(\mathsf{M}_M(m) + \frac{m}{s}\,\mathsf{M}_M(s)\right).$$

We then call T recursively for each small $n_1' \times n_2'$ subproblem. The recursion continues until we encounter $\lg \max(n_1, n_2) \leqslant b$, at which point we finally call the base case machine T'.

Let us estimate the number of recursive calls, and in particular show that this quantity is finite. For $j = 1, 2, \ldots$, let $n_1^{(j)}$, $n_2^{(j)}$ and $m^{(j)}$ denote the values of n_1, n_2 and m passed to the j-th recursive call, where $j = 1$ corresponds to the initial call. As shown in the proof of Proposition 6.6 (see (6.1)), we have

$$\lg \max(n_1^{(j)}, n_2^{(j)}) = \lg^{\circ j} \max(n_1, n_2)$$

for each j. The sequence $\lg^{\circ j} \max(n_1, n_2)$ is strictly decreasing, so we must eventually reach $\lg \max(n_1^{(j)}, n_2^{(j)}) \leqslant b$. Let ℓ be the smallest j for which this occurs, i.e., the number of recursive calls (including the initial call).

We claim that

$$\ell \leqslant \max(1, \lg^* \max(n_1, n_2) - \lg^* b + 1). \tag{6.4}$$

This clearly holds if $\ell = 1$. Suppose now that $\ell \geqslant 2$. Then

$$\lg^{\circ(\ell-1)} \max(n_1, n_2) > b.$$

If $b = 1$, this implies that $\lg^* \max(n_1, n_2) > \ell - 1$, and $\lg^* b = 0$, so (6.4) holds. If $b \geqslant 2$, then $\lg^* b \geqslant 1$, so applying $\lg^{\circ((\lg^* b)-1)}$ to both sides,

$$\lg^{\circ(\ell+(\lg^* b)-2)} \max(n_1, n_2) \geqslant \lg^{\circ((\lg^* b)-1)} b > 1.$$

This implies that $\lg^* \max(n_1, n_2) > \ell + \lg^* b - 2$, i.e., $\ell \leqslant \lg^* \max(n_1, n_2) - \lg^* b + 1$, so again (6.4) holds.

The same complexity argument as in the proof of Proposition 6.6 now shows that

$$\mathsf{T}_T(m; n_1, n_2, b) < \frac{n_1 n_2}{n_1^{(\ell)} n_2^{(\ell)}}\,\mathsf{T}_{T'}(m^{(\ell)}; n_1^{(\ell)}, n_2^{(\ell)}, b) + \sum_{j=1}^{\ell-1} O(\overline{\mathsf{M}}_M(m)).$$

(Note here that the big-O constant is the same at each recursion level.) We showed above that the cost of each call to the base case is $O(\overline{\mathsf{M}}_M(m^{(\ell)}))$, so the first term

becomes

$$O\left(\frac{n_1 n_2}{n_1^{(\ell)} n_2^{(\ell)}}\,\overline{\mathsf{M}}_M(m^{(\ell)})\right) = O\left(\frac{m}{m^{(\ell)}}\,\overline{\mathsf{M}}_M(m^{(\ell)})\right) = O(\overline{\mathsf{M}}_M(m)).$$

Therefore we finally obtain

$$\mathsf{T}_T(m; n_1, n_2, b) = O(\overline{\mathsf{M}}_M(m)) + \sum_{j=1}^{\ell-1} O(\overline{\mathsf{M}}_M(m)) = O(\ell\,\overline{\mathsf{M}}_M(m)). \qquad \square$$

As usual, we have the following non-dyadic version.

Corollary 6.9. *Let M be a multiplication machine. Then there exists an (ordinary, non-dyadic) transposition machine T such that*

$$\mathsf{T}_T(m; n_1, n_2, b) = O(\ell_0\,\overline{\mathsf{M}}_M(m)), \qquad \ell_0 \coloneqq \max(1, \lg^* \max(n_1, n_2) - \lg^* b).$$

Proof. Applying the rounding and zero-padding strategy of Corollary 6.2, we obtain the complexity bound $O(\tilde{\ell}\,\overline{\mathsf{M}}_M(\tilde{m}))$, where by (6.4) we have

$$\tilde{\ell} \leqslant \max(1, \lg^* \max(\tilde{n}_1, \tilde{n}_2) - \lg^* \tilde{b} + 1).$$

Clearly $\lg^* \tilde{n}_i = \lg^* n_i$ and $\lg^* \tilde{b} = \lg^* b$, so $\tilde{\ell} = O(\ell_0)$ where ℓ_0 is defined as in the statement of Proposition 6.8. Finally, modifying the machine via Lemma 3.2 in the usual way, we may replace $\overline{\mathsf{M}}_M(\tilde{m})$ by $\overline{\mathsf{M}}_M(m)$, to obtain the desired bound $O(\ell_0\,\overline{\mathsf{M}}_M(m))$. $\square$

Proof of Theorem 1.4. Let $f(m)$ be a non-decreasing function, and suppose that there exists a multiplication machine M satisfying $\mathsf{M}_M(m) = O(mf(m))$. Then

$$\overline{\mathsf{M}}_M(m) = m \cdot \max_{k \leqslant m} \frac{\mathsf{M}_M(k)}{k} = m \cdot \max_{k \leqslant m} O(f(k)) = O(mf(m)).$$

Applying Corollary 6.9 with $b \coloneqq 1$ and $n_1, n_2 \coloneqq \lfloor m^{1/2} \rfloor$, we obtain a binary transposition machine T such that

$$\mathsf{T}_T(m) = O(\lg^*(\lfloor m^{1/2} \rfloor)\,\overline{\mathsf{M}}_M(m)) = O(mf(m) \lg^* m).$$

But this contradicts the hypothesis that no such machine exists. $\square$

Remark 6.10. Using the $\overline{\mathsf{M}}_M(m)$ notation, it is possible to express Theorem 1.4 directly in the form of a lower bound. Namely, the theorem is equivalent to the following statement: for any multiplication machine M, there exists a binary transposition machine T such that

$$\overline{\mathsf{M}}_M(m) = \Omega\left(\frac{\mathsf{T}_T(m)}{\lg^* m}\right).$$

6.4. **Transposition with larger coefficients.** We mentioned in Section 1.2 that our methods are not quite strong enough to prove the following statement: if the binary transposition problem cannot be solved in linear time, then multiplication cannot be carried out in linear time. To make progress towards this goal, with our current methods it appears to be necessary to abandon the binary case and work instead with larger coefficients. In this section we briefly outline a result in this direction.

Fix an integer $\ell \geqslant 1$, and for any $m \geqslant 1$ consider the transposition problem with parameters

$$b \coloneqq \lg^{\circ \ell} m, \qquad n_1, n_2 = n \coloneqq \lfloor (m/b)^{1/2} \rfloor. \tag{6.5}$$

A machine T performing this type of transposition will be called an *ℓ-logarithmic transposition machine*. We denote its worst-case running time by $\mathsf{T}^{\ell}_{T}(m)$.

Theorem 6.11. *Fix $\ell \geqslant 1$ and a non-decreasing function $f(m)$. If no ℓ-logarithmic transposition machine achieves $\mathsf{T}^{\ell}(m) = O(mf(m))$, then no multiplication machine achieves $\mathsf{M}(m) = O(mf(m))$.*

For example, if ℓ-logarithmic transposition cannot be performed in linear time — even for one fixed value of ℓ — then multiplication cannot be carried out in linear time. In this sense, Theorem 6.11 is slightly stronger than Theorem 1.4, although it does involve an arguably less natural transposition problem.

Proof. Suppose that M is a multiplication machine such that $\mathsf{M}_M(m) = O(mf(m))$. Apply Corollary 6.9 with n_1, n_2, b defined as in (6.5). This yields an ℓ-logarithmic transposition machine T such that $\mathsf{T}^{\ell}_{T}(m) = O(\ell_0 \, \overline{\mathsf{M}}_M(m))$ where

$$\ell_0 = \max(1, \lg^* \max(n_1, n_2) - \lg^* b) \leqslant \max(1, \lg^* m - \lg^* b).$$

But since $b = \lg^{\circ \ell} m$ we clearly have $\lg^* m \leqslant \lg^* b + \ell$. Thus $\ell_0 \leqslant \ell = O(1)$, since ℓ was assumed fixed. The same calculation as in the proof of Theorem 1.4 then shows that $\mathsf{T}^{\ell}_{T}(m) = O(\overline{\mathsf{M}}_M(m)) = O(mf(m))$, contradicting the hypothesis. □

7. Discussion and perspectives

7.1. **Alternative complexity models.** It is interesting to ask what impact our reductions have in complexity models other than the multitape Turing machine.

One model often discussed in the context of integer multiplication is the *Boolean circuit model* [Pap94, §4.3]. The $O(n \log n)$ multiplication algorithm of [HvdH21] works in this model, and it is reasonable to guess that this is optimal. Unfortunately, matrix transposition is trivial (zero cost!) in this model, so while our reductions still make sense here, they are completely useless.

If we want to stay closer to the Turing machine paradigm, one natural variant to consider is the *multitape Turing machine with d-dimensional tapes* (for fixed d) [Hen66, Reg95]. While transposition is not free in such a model, it can easily be carried out in linear time (provided that $d \geqslant 2$), suggesting that our reductions are not especially interesting here either.

7.2. **Other permutations beyond transposition.** Matrix transposition may be regarded as a specific type of permutation of the input. It would be interesting to know which other permutations can be reduced to integer multiplication. For instance, what about the permutation $(x_0, \ldots, x_{2^\ell - 1}) \mapsto (x_{\rho(0)}, \ldots, x_{\rho(2^\ell - 1)})$, where $\rho(i)$ stands for the bit-reversal of i as an ℓ-bit integer?

7.3. **Alternative coefficient rings.** In the main reduction (Theorem 4.1), we chose to work over the complex numbers. It may be possible to work instead with DFTs over a finite field $\mathbb{F}_q$. This approach has a long history in the context of integer multiplication [Pol71], and has the advantage of avoiding the analysis of numerical error. However, it introduces new technical difficulties, such as the construction of field extensions containing suitable roots of unity. We have not checked the details.

7.4. **Polynomial multiplication over finite fields.** Let $\mathsf{M}_q(n)$ denote the cost (in the Turing model) of multiplying polynomials in $\mathbb{F}_q[x]$ of degree n. By analogy with the integer case, it is widely suspected that $\mathsf{M}_q(n) = \Theta(b \log b)$ where $b \coloneqq n \log q$ is the total bit size. Currently, the best unconditional upper bound is $\mathsf{M}_q(n) = O(b \log b \cdot 4^{\lg^* b})$ [HvdH19], and there is even a conditional upper bound $\mathsf{M}_q(n) = O(b \log b)$ assuming an unproved but plausible number-theoretic hypothesis [HvdH22].

It is natural to ask whether the methods of this paper generalise to this situation, i.e., can we reduce matrix transposition to polynomial multiplication over $\mathbb{F}_q$? If so, then lower bounds on transposition may imply lower bounds for $\mathsf{M}_q(n)$.

A rough analogy for the Boolean circuit model in this context would be some variant of algebraic complexity, such as counting the number of arithmetic operations in $\mathbb{F}_q$, without regard for memory access or locality. Again, any analogue of our reduction would be useless in such a model, as transposition is presumably free of charge.

7.5. **Performing small products in parallel.** Consider the following problem: given integers $u_1, \dots, u_n$ and $v_1, \dots, v_n$ of bit size p, compute the products $u_1 v_1$, $\dots$, $u_n v_n$. Is there a way to reduce this problem to a single integer multiplication problem of size $O(np)$? (Or $O(1)$ such problems?) We do not know how to do this, but it is not implausible that such a reduction exists. If this reduction were possible, then many of the results of this paper could likely be improved, and the proofs would simplify considerably. Namely, in Theorem 4.1, the second term $l_1 l_2 n_1 n_2 \, \mathsf{M}(b)$ could probably be absorbed into the $\mathsf{M}(m)$ term, by performing all the multiplications by Bluestein factors and twiddle factors "in parallel". This would likely imply that many of the $\overline{\mathsf{M}}(m)$ terms in Section 6 could be replaced by simply $\mathsf{M}(m)$, which would in turn allow us to strengthen the statements of Theorem 1.3 and Theorem 1.4 into more direct lower bounds for $\mathsf{M}(m)$.

7.6. **Small versus large coefficients.** Our methods are weaker than ideal when applied to matrices with small coefficients, such as the $b = 1$ case (compare Theorem 1.4 with Theorem 6.11). On the other hand, one feels intuitively that transposing a matrix with b-bit entries should be b times more expensive than transposing a matrix of the same dimensions with 1-bit entries. In symbols, we might reasonably expect that

$$|\, \mathsf{T}(n_1, n_2, b) - b \, \mathsf{T}(n_1, n_2, 1)| = O(n_1 n_2 b).$$

Unfortunately, we have not yet succeeded in designing reductions between transpositions with b-bit and 1-bit entries, in either direction. If we could establish that $\mathsf{T}(n_1, n_2, b) = \Omega(b \, \mathsf{T}(n_1, n_2, 1))$, then we could immediately improve our results for small coefficients (such as Theorem 1.4) by leveraging the known results for larger coefficients.

If we restrict attention to *oblivious* Turing machines, i.e., machines for which the sequence of tape movements depends only on the size of the input, then there is a straightforward reduction in one direction: given a machine handling the 1-bit case, we can transpose a b-bit matrix by simulating b copies of the 1-bit machine in parallel. Hence, in this model $\mathsf{T}(n_1, n_2, b) = O(b \, \mathsf{T}(n_1, n_2, 1))$. Unfortunately, this reduction goes in the wrong direction to be useful in the way suggested in the previous paragraph.

References

[AFKL19] P. Afshani, C. B. Freksen, L. Kamma, and K. G. Larsen, *Lower bounds for multiplication via network coding*, 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019) (Dagstuhl, Germany) (Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, eds.), Leibniz International Proceedings in Informatics (LIPIcs), vol. 132, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, pp. 10:1–10:12.

[AHJ+06] M. Adler, N. J. A. Harvey, K. Jain, R. Kleinberg, and A. R. Lehman, *On the capacity of information networks*, Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (USA), SODA '06, Society for Industrial and Applied Mathematics, 2006, pp. 241–250.

[AHU75] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1975, Second printing. MR 413592

[BB87] J. M. Borwein and P. B. Borwein, *Pi and the AGM*, Canadian Mathematical Society Series of Monographs and Advanced Texts, John Wiley & Sons, Inc., New York, 1987, A study in analytic number theory and computational complexity, A Wiley-Interscience Publication. MR 877728

[BGS07] A. Bostan, P. Gaudry, and É. Schost, *Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator*, SIAM J. Comput. **36** (2007), no. 6, 1777–1806. MR 2299425 (2008a:11156)

[Blu70] L. I. Bluestein, *A linear filtering approach to the computation of discrete Fourier transform*, IEEE Transactions on Audio and Electroacoustics **18** (1970), no. 4, 451–455.

[BZ11] R. P. Brent and P. Zimmermann, *Modern Computer Arithmetic*, Cambridge Monographs on Applied and Computational Mathematics, vol. 18, Cambridge University Press, Cambridge, 2011. MR 2760886

[CA69] S. A. Cook and S. O. Aanderaa, *On the minimum computation time of functions*, Trans. Amer. Math. Soc. **142** (1969), 291–314. MR 0249212

[CT65] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp. **19** (1965), 297–301. MR 0178586

[DM93] M. Dietzfelbinger and W. Maass, *The complexity of matrix transposition on one-tape off-line Turing machines with output tape*, Theoretical Computer Science **108** (1993), no. 2, 271–290.

[Flo72] R. W. Floyd, *Permuting information in idealized two-level storage*, Complexity of Computer Computations: Proceedings of a Symposium on the Complexity of Computer Computations, Springer, 1972, pp. 105–109.

[GG13] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 3rd ed., Cambridge University Press, Cambridge, 2013. MR 3087522

[Hen66] F. C. Hennie, *On-line Turing machine computations*, IEEE Transactions on Electronic Computers **EC-15** (1966), no. 1, 35–44.

[Hoe14] J. van der Hoeven, *Faster relaxed multiplication*, Proc. ISSAC '14 (Kobe, Japan), July 2014, pp. 405–412.

[HvdH19] D. Harvey and J. van der Hoeven, *Faster polynomial multiplication over finite fields using cyclotomic coefficient rings*, J. Complexity **54** (2019), 101404, 18. MR 3983215

[HvdH21] ———, *Integer multiplication in time $O(n \log n)$*, Ann. of Math. (2) **193** (2021), no. 2, 563–617. MR 4224716

[HvdH22] ———, *Polynomial multiplication over finite fields in time $O(n \log n)$*, J. ACM **69** (2022), no. 2, Art. 12, 40. MR 4433320

[HvdHL16] D. Harvey, J. van der Hoeven, and G. Lecerf, *Even faster integer multiplication*, J. Complexity **36** (2016), 1–30. MR 3530637

[Kir88] W. W. Kirchherr, *Transposition of an $l \times l$ matrix requires $\omega(\log l)$ reversals on conservative Turing machines*, Information processing letters **28** (1988), no. 2, 55–59.

[Knu98] D. E. Knuth, *The Art of Computer Programming. Vol. 3*, Addison-Wesley, Reading, MA, 1998, Sorting and searching, Second edition [of MR0445948]. MR 3077154

[Mor73] J. Morgenstern, *Note on a lower bound on the linear complexity of the fast Fourier transform*, JACM **20** (1973), no. 2, 305–306.

[Pap94] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Company, Reading, MA, 1994. MR 1251285 (95f:68082)

[Pau73] W. J. Paul, *Optimale Algorithmen zum Transponieren quadratischer Matrizen*, Dritte Jahrestagung der Gesellschaft für Informatik (Hamburg, 1973), Lecture Notes in Comput. Sci., vol. Vol. 1, Springer, Berlin-New York, 1973, pp. 72–80. MR 375830

[PFM74] M. S. Paterson, M. J. Fischer, and A. R. Meyer, *An improved overlap argument for on-line multiplication*, Complexity of computation (Proc. Sympos., New York, 1973), Amer. Math. Soc., Providence, R. I., 1974, pp. 97–111. SIAM–AMS Proc., Vol. VII. MR 0423875

[Pol71] J. M. Pollard, *The fast Fourier transform in a finite field*, Math. Comp. **25** (1971), 365–374. MR 0301966

[Rad68] C. M. Rader, *Discrete Fourier transforms when the number of data samples is prime*, Proc. IEEE **56** (1968), no. 6, 1107–1108.

[Reg95] K. W. Regan, *On superlinear lower bounds in complexity theory*, Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference, IEEE, 1995, pp. 50–64.

[Sch82] A. Schönhage, *Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients*, Computer algebra (Marseille, 1982), Lecture Notes in Comput. Sci., vol. 144, Springer, Berlin, 1982, pp. 3–15. MR 680048 (83m:68064)

[SS71] A. Schönhage and V. Strassen, *Schnelle Multiplikation grosser Zahlen*, Computing (Arch. Elektron. Rechnen) **7** (1971), 281–292. MR 0292344 (45 #1431)

[Sto71] H. S. Stone, *Parallel processing with the perfect shuffle*, IEEE Transactions on Computers **C-20** (1971), no. 2, 153–161.

[Sto73] H.-J. Stoß, *Rangierkomplexität von Permutationen*, Acta Informatica **2** (1973), no. 1, 80–96.

School of Mathematics and Statistics, University of New South Wales, Sydney NSW 2052, Australia

Email address: d.harvey@unsw.edu.au

CNRS, Laboratoire d'informatique, École polytechnique, 91128 Palaiseau, France

Email address: vdhoeven@lix.polytechnique.fr