The I/O Complexity of Hybrid Algorithms for Square Matrix Multiplication

Lorenzo De Stefani
Department of Computer Science, Brown University, United States of America
lorenzo@cs.brown.edu

Abstract

Asymptotically tight lower bounds are derived for the I/O complexity of a general class of hybrid algorithms computing the product of $n \times n$ square matrices combining "*Strassen-like*" fast matrix multiplication approach with computational complexity $\Theta\left(n^{\log_2 7}\right)$, and "*standard*" matrix multiplication algorithms with computational complexity $\Omega\left(n^3\right)$. We present a novel and tight $\Omega\left(\left(\frac{n}{\max\{\sqrt{M},n_0\}}\right)^{\log_2 7}\left(\max\{1,\frac{n_0}{M}\}\right)^3 M\right)$ lower bound for the I/O complexity of a class of "*uniform, non-stationary*" hybrid algorithms when executed in a two-level storage hierarchy with M words of fast memory, where n_0 denotes the threshold size of sub-problems which are computed using standard algorithms with algebraic complexity $\Omega\left(n^3\right)$.

The lower bound is actually derived for the more general class of "*non-uniform, non-stationary*" hybrid algorithms which allow recursive calls to have a different structure, even when they refer to the multiplication of matrices of the same size and in the same recursive level, although the quantitative expressions become more involved. Our results are the first I/O lower bounds for these classes of hybrid algorithms. All presented lower bounds apply even if the recomputation of partial results is allowed and are asymptotically tight.

The proof technique combines the analysis of the Grigoriev's flow of the matrix multiplication function, combinatorial properties of the encoding functions used by fast Strassen-like algorithms, and an application of the Loomis-Whitney geometric theorem for the analysis of standard matrix multiplication algorithms. Extensions of the lower bounds for a parallel model with P processors are also discussed.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases I/O complexity, Hybrid Algorithm, Matrix Multiplication, Recomputation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.33

Related Version A full version of the paper is available at `https://arxiv.org/abs/1904.12804`.

Funding This research was supported by NSF Award ISS 1813444.

Acknowledgements I want to thank Gianfranco Bilardi at the University of Padova (Italy) for inital conversations on the topic of this work, and Megumi Ando at Brown University (USA) for her feedback on early versions of this manuscript.

1 Introduction

Data movement plays a critical role in the performance of computing systems, in terms of both time and energy. This technological trend [16] appears destined to continue, as physical limitations on minimum device size and on maximum message speed lead to inherent costs when moving data, whether across the levels of a hierarchical memory system or between processing elements of a parallel system [13]. While the communication requirements of algorithms have been widely investigated in the literature, obtaining significant and tight lower bounds based on such requirements remains an important and challenging task.


© Lorenzo De Stefani;
licensed under Creative Commons License CC-BY

In this paper, we focus on the I/O complexity of a general class of hybrid algorithms for computing the product of square matrices. Such algorithms combine *fast* algorithms with base case 2×2 similar to Strassen's matrix multiplication algorithm [36] with algebraic (or *computational*) complexity $\mathcal{O}\left(n^{\log_2 7}\right)$ with *standard* (or *classic*) matrix multiplication algorithms with algebraic complexity $\Omega\left(n^3\right)$. Further, these algorithms allow recursive calls to have a different structure, even when they refer to the multiplication of matrices in the same recursive level and of the same input size. These algorithms are referred in literature as "*non-uniform, non-stationary*". This class includes, for example, algorithms that optimize for input sizes [19, 20, 23]. Matrix multiplication is a pervasive primitive utilized in many applications.

While of actual practical importance, to the best of our knowledge, no characterization of the I/O complexity of such algorithms has presented before this work. This is likely due to the the fact that the irregular nature of hybrid algorithms and, hence, the irregular structure of the corresponding Computational Directed Acyclic Graphs (CDAGs), complicates the analysis of the combinatorial properties of the CDAG which is the foundation of many of I/O lower bound technique presented in literature (e.g., [8, 25, 32]).

The technique used in this work overcomes such challenges and yields asymptotically tight I/O lower bounds which hold even if recomputation of intermediate values is allowed.

Previous and Related Work. Strassen [36] showed that two $n \times n$ matrices can be multiplied with $O(n^\omega)$ operations, where $\omega = \log_2 7 \approx 2.8074$, hence with asymptotically fewer than the n^3 arithmetic operations required by the straightforward implementation of the definition of matrix multiplication. This result has motivated a number of efforts which have led to increasingly faster algorithms, at least asymptotically, with the current record being at $\omega < 2.3728639$ [28].

I/O complexity has been introduced in the seminal work by Hong and Kung [25]; it is essentially the number of data transfers between the two levels of a memory hierarchy with a fast memory of M words and a slow memory with an unbounded number of words. Hong and Kung presented techniques to develop lower bounds to the I/O complexity of computations modeled by *computational directed acyclic graphs* (CDAGs). The resulting lower bounds apply to all the schedules of the given CDAG, including those with recomputation, that is, where some vertices of the CDAG are evaluated multiple times. Among other results, they established a $\Omega\left(n^3/\sqrt{M}\right)$ lower bound to the I/O complexity of standard, definition-based matrix multiplication algorithms, which matched a known upper bound [15]. The techniques of [25] have also been extended to obtain tight communication bounds for the definition-based matrix multiplication in some parallel settings [4, 24]. Ballard et al. generalized the results on matrix multiplication of Hong and Kung [25] in [7, 6] by using the approach proposed in [24] based on the Loomis-Whitney geometric theorem [29, 37].

In an important contribution, Ballard et al. [8], obtained an $\Omega((n/\sqrt{M})^{\log_2 7} M)$ I/O lower bound for Strassen's algorithm, using the "*edge expansion approach*". The authors extend their technique to a class of "*Strassen-like*" fast multiplication algorithms and to fast recursive multiplication algorithms for rectangular matrices [3]. This result was later generalized to increasingly broader classes of "*Strassen-like*" algorithms by Scott et. al [33] using the "*path routing*" technique, and De Stefani [17] using a combination the concept of Grigoriev's flow of a function and the "*dichotomy width*" technique [14]. While the previously mentioned results hold only under the restrictive assumption that no intermediate result may be more than once (i.e., the *no-recomputation assumption*), in [10] Bilardi and De Stefani

introduced the first asymptotically tight I/O lower bound which holds if recomputation is allowed. Their technique was later extended to the analysis of Strassen-like algorithms with base case 2×2 [30], and to the analysis of Toom-Cook integer multiplication algorithms [11].

A parallel, "*communication avoiding*" implementation of Strassen's algorithm whose performance matches the known lower bound [8, 33], was proposed by Ballard et al. [5].

In [34], Scott derived a lower bound for the I/O complexity of a class of uniform, non-stationary algorithms combining Strassen-like algorithm with recursive standard algorithms. This result holds only under the restrictive no-recomputation assumption and considers only compositions of recursive matrix multiplication algorithms.

To the best of our knowledge, ours is the first work presenting asymptotically tight I/O lower bounds for non-uniform, non-stationary hybrid algorithms for matrix multiplication that hold when recomputation is allowed.

On the impact of recomputation. While it is of interest to study the I/O complexity under the no-recomputation assumption, it is also very important to investigate what can be achieved with recomputation. For some CDAGs, recomputing intermediate values allows reducing the space and/or the I/O complexity of an algorithm [32]. As shown [12], some algorithms admit a "*portable schedule*" (i.e., a schedule which achieves optimal performance across memory hierarchies with different access costs) only if recomputation is allowed. Recomputation can also enhance the performance of simulations among networks (see [27] and references therein) and plays a key role in the design of efficient area-universal VLSI architectures with constant slowdown [9].

Our results. In our main result, we present the first I/O lower bound for a class $\mathfrak{H}$ of non-uniform, non-stationary hybrid matrix multiplication algorithms when executed in a two-level storage hierarchy with M words of fast memory. Algorithms in $\mathfrak{H}$ combine fast Strassen-like algorithms with base case 2×2 with algebraic complexity $\Theta\left(n^{\log_2 7}\right)$, and standard algorithms based on the definition with algebraic complexity $\Omega\left(n^3\right)$. These algorithms allow recursive calls to have a different structure, even when they refer to the multiplication of matrices in the same recursive level and of the same input size. The result in Theorem 9 relates the I/O complexity of algorithms in $\mathfrak{H}$ to the number and the input size of an opportunely selected set of the sub-problems generated by the algorithms themselves.

By specializing the result in Theorem 9, we also present, in Theorem 11, a novel $\Omega\left(\left(\frac{n}{\max\{\sqrt{M},n_0\}}\right)^{\log_2 7}\left(\max\{1,\frac{n_0}{M}\}\right)^3 M\right)$ lower bound for the I/O complexity of algorithms in a subclass $\mathfrak{UH}\left(n_0\right)$ of $\mathfrak{H}$ composed by *uniform non-stationary* hybrid algorithms where n_0 denotes the threshold input size of sub-problems which are computed using standard algorithms with algebraic complexity $\Omega\left(n^3\right)$.

The result in Theorem 11 considerably extends a previous result by Scott [34] as the latter covers only a sub-class of $\mathfrak{UH}\left(n_0\right)$ composed by uniform, non-stationary algorithms combining Strassen-like algorithms with the recursive standard algorithm, and holds only assuming that no intermediate value is recomputed.

Our lower bounds in Theorem 9 and Theorem 11 allow for recomputation of intermediate values and are asymptotically tight. As the matching upper bounds do not recompute any intermediate value, we conclude that using recomputation may reduce the I/O complexity of the considered classes of hybrid algorithms by at most a constant factor.

Our proof technique is of independent interest since it exploits to a significant extent the "*divide and conquer*" nature exhibited by many algorithms. Our approach combines elements from the "*G-flow*" I/O lower bound technique originally introduced by Bilardi

and De Stefani, with an application of the Loomis-Whitney geometric theorem, which has been used by Irony et al. to study the I/O complexity of standard matrix multiplication algorithms [24], to recover an information which relates to the concept of "*Minimum set*" introduced in Hong and Kung's method. We follow the "*dominator set*" approach pioneered by Hong and Kung in [25]. However, we focus the dominator analysis only on a select set of target vertices, which, depending on the algorithm structure, correspond either to the outputs of the sub-CDAGs that correspond to sub-problems of a suitable size (i.e., chosen as a function of the fast memory capacity M) computed using a fast Strassen-like algorithm, or to the the vertices corresponding to the elementary products evaluated by a standard (definition) matrix multiplication algorithm.

We derive our main results for the hierarchical memory model (or external memory model). Our results generalize to parallel models with P processors. For these parallel models, we derive lower bounds for the "*bandwith cost*", that is for the number of messages (and, hence, the number of memory words) that must be either sent or received by at least one processor during the execution of the algorithm.

Paper organization. In Section 2 we outline the notation and the computational models used in the rest of the presentation. In Section 3 we rigorously define the class of hybrid matrix multiplication algorithms $\mathfrak{H}$ being considered. In Section 4 we discuss the construction and important properties of the CDAGs corresponding to algorithms in $\mathfrak{H}$. In Section 5 we introduce the concept of Maximal Sup-Problem (MSP) and describe their properties which lead to the I/O lower bounds for algorithms in $\mathfrak{H}$ in Section 6.

2 Preliminaries

We consider algorithms that compute the product of two square matrices $\mathbf{A} \times \mathbf{B} = \mathbf{C}$ with entries from a ring $\mathcal{R}$. We use A to denote the set of variables each corresponding to an entry of matrix $\mathbf{A}$. We refer to the number of entries of a matrix $\mathbf{A}$ as its "*size*" and we denote it as $|A|$. We denote the entry on the i-th row of the j-th column of matrix $\mathbf{A}$ as $\mathbf{A}[i][j]$.

In this work we focus on algorithms whose execution can be modeled as a *computational directed acyclic graph* (CDAG) $G = (V, E)$. Each vertex $v \in V$ represents either an input value or the result of a unit-time operation (i.e., an intermediate result or one of the output values) which is stored using a single memory word. For example, each of the input (resp., output) vertices of G corresponds to one of the $2n^2$ entries of the factor matrices $\mathbf{A}$ and $\mathbf{B}$ (resp., to the n^2 entries of the product matrix $\mathbf{C}$). The *directed* edges in E represent data dependencies. That is, a pair of vertices $u, v \in V$ are connected by an edge (u, v) directed from u to v if and only if the value corresponding to u is an operand of the unit time operation which computes the value corresponding to v. A *directed path* connecting vertices $u, v \in V$ is an ordered sequence of vertices starting with u and ending with v, such that there is an edge in E directed from each vertex in the sequence to its successor.

We say that $G' = (V', E')$ is a *sub-CDAG* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. Note that, according to this definition, every CDAG is a sub-CDAG of itself. We say that two sub-CDAGs $G' = (V', E')$ and $G'' = (V'', E'')$ of G are *vertex-disjoint* if $V' \cap V'' = \emptyset$. Analogously, two directed paths in G are vertex-disjoint if they do not share any vertex.

When analyzing the properties of CDAGs we make use of the concept of *dominator set* originally introduced in [25]. We use the following – slightly different – definition:

▶ **Definition 1** (Dominator set). *Given a CDAG $G = (V, E)$, let $I \subset V$ denote the set of its input vertices. A set $D \subseteq V$ is a* dominator set *for $V' \subseteq V$ with respect to $I' \subseteq I$ if every path from a vertex in I' to a vertex in V' contains at least a vertex of D. When $I' = I$, D is simply referred as "a dominator set for V'".*

I/O model and machine models. We assume that sequential computations are executed on a system with a two-level memory hierarchy, consisting of a fast memory or *cache* of size M (measured in memory words) and a *slow memory* of unlimited size. An operation can be executed only if all its operands are in cache. We assume that each entry of the input and intermediate results matrices (including entries of the output matrix) is maintained in a single memory word (the results trivially generalize if multiple memory words are used).

Data can be moved from the slow memory to the cache by `read` operations and, in the other direction, by `write` operations. These operations are also called *I/O operations*. We assume the input data to be stored in slow memory at the beginning of the computation. The evaluation of a CDAG in this model can be analyzed by means of the "*red-blue pebble game*" [25]. The number of I/O operations executed when evaluating a CDAG depends on the "*computational schedule*", that is, it depends on the order in which vertices are evaluated and on which values are kept in/discarded from cache.

The *I/O complexity* $IO_G(M)$ of a CDAG G is defined as the minimum number of I/O operations over all possible computational schedules. We further consider a generalization of this model known as the "*External Memory Model*" by Aggarwal and Vitter [2], where $B \geq 1$ values can be moved between cache and consecutive slow memory locations with a single I/O operation. For $B = 1$, this model clearly reduces to the red-blue pebble game.

Given an algorithm $\mathcal{A}$, we only consider "*parsimonious execution schedules*", that is schedules such that: (i) each time an intermediate result (excluding the output entries of $\mathbf{C}$) is computed, such value is then used to compute at least one of the values of which it is an operand before being removed from the memory (either the cache or slow memory); and (ii) any time an intermediate result is read from slow to cache memory, such value is then used to compute at least one of the values of which it is an operand before being removed from the memory or moved back to slow memory using a *write* I/O operation. Clearly, any non-parsimonious schedule $\mathcal{C}$ can be reduced to a parsimonious schedule $\mathcal{C}'$ by removing all the steps which violate the definition of parsimonious computation. $\mathcal{C}'$ has therefore less computational and I/O operations than $\mathcal{C}$. Hence, restricting the analysis to parsimonious computations leads to no loss of generality.

We also consider a parallel model where P processors, each with a local memory of size $2n^2/P \leq M < n^2$, are connected by a network. We do not, however, make any assumption on the initial distribution of the input data nor regarding the balance of the computational load among the P processors. Processors can exchange point-to-point messages, with every message containing up to B_m memory words. For this parallel model, we derive lower bounds for the number of messages that must be either sent or received by at least one processor during the CDAG evaluation. The notion of "*parsimonious execution schedules*" straightforwardly extends to this parallel model.

3 Hybrid matrix multiplication algorithms

In this work, we consider a family of hybrid matrix multiplication algorithms obtained by hybridizing the two following classes of algorithms:

Standard matrix multiplication algorithms. This class includes all the square matrix multiplication algorithms which, given the input factor matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$, satisfy the following properties:

- The n^3 *elementary products* $\mathbf{A}[i][j]\mathbf{B}[j][i]$, for $i, j = 0, \ldots, n-1$, are *directly computed*;
- Each of the $\mathbf{C}[i][j]$ is computed by summing the values of the n elementary products $\mathbf{A}[i][z]\mathbf{B}[z][j]$, for $z = 0, \ldots, n-1$, through a *summation tree* by additions and subtractions only;
- The evaluations of the $\mathbf{C}[i][j]$'s are *independent of each other*. That is, internal vertex sets of the summation trees of all the $\mathbf{C}[i][j]$'s are *disjoint from each other*.

This class, also referred in literature as *classic*, *naive* or *conventional* algorithms, correspond to that studied for the by Hong and Kung [25] (for the sequential setting) and by Irony et al. [24] (for the parallel setting). Algorithms in this class have computational complexity $\Omega\left(n^3\right)$. This class includes, among others, the sequential iterative *definition* algorithm, the sequential recursive divide and conquer algorithm based on block partitioning, and parallel algorithms such as Cannon's "*2D*" algorithm [15], the "*2.5D*" algorithm by Solomonik and Demmel [35], and "*3D*" algorithms [1, 26].

Fast Strassen-like matrix multiplication algorithms with base case 2×2. This class includes algorithms following a structure similar to that of Strassen's [36] and Winograd's variation [38] (which reduces the leading coefficient of the arithmetic complexity reduced from 7 to 6). For further details on Strassen's algorithm we refer the reader to the extended version of this work [18]). Algorithms in this class follow three common steps:

1. **Encoding:** Generate the inputs, of size $n/2 \times n/2$ of seven *sub-problems*, as linear sums of the input matrices;
2. **Recursive multiplications:** Compute (recursively) the seven generated matrix multiplication sub-products;
3. **Decoding:** Computing the entries of the product matrix $\mathbf{C}$ via linear combinations of the output of the seven sub-problems.

Algorithms in this class have algebraic complexity $\mathcal{O}\left(n^{\log_2 7}\right)$, which is optimal for algorithms with base case 2×2 [38].

Remarkably, the only properties of relevance for the analysis of the I/O complexity of algorithms in these classes are those used in the characterization of the classes themselves.

In this work we consider a general class of *non-uniform, non-stationary* hybrid square matrix multiplication algorithms, which allow mixing of schemes from the fast Strassen-like class with algorithms from the standard class. Given an algorithm $\mathcal{A}$ let P denote the problem corresponding to the computation of the product of the input matrices $\mathcal{A}$ and $\mathcal{B}$. Consider an "*instruction function*" $f_{\mathcal{A}}(P)$, which, given P as input returns either (a) indication regarding the algorithm from the standard class which is to be used to compute P, or (b) indication regarding the fast Strassen-like algorithm to be used to recursively generate seven sub-problems $P_1, P_2, \ldots, P_7$ and the instruction functions $f_{\mathcal{A}}(P_i)$ for each of the seven sub-problems. We refer to the class of non-uniform, non-stationary algorithms which can be characterized by means of such instruction functions as $\mathfrak{H}$. Algorithms in $\mathfrak{H}$ allow recursive calls to have a different structure, even when they refer to the multiplication of matrices in the same recursive level. E.g., some of the sub-problems with the same size may be computed using algorithms form the standard class while others may be computed using recursive algorithms from the fast class. This class includes, for example, algorithms that optimize for input sizes, (for sizes that are not an integer power of a constant integer).

This corresponds to actual practical scenarios, as the use of Strassen-like algorithms is mostly beneficial for large input size. As the size of the input of the recursively generated sub-problems decreases, the asymptotic benefit of fast algorithms is lost due to the increasing relative impact of the constant multiplicative factor, and algorithms in the standard class exhibit lower *actual* algebraic complexity. For a discussion on such hybrid algorithms and their implementation issues we refer the reader to [20, 23] (sequential model) and [19] (parallel model).

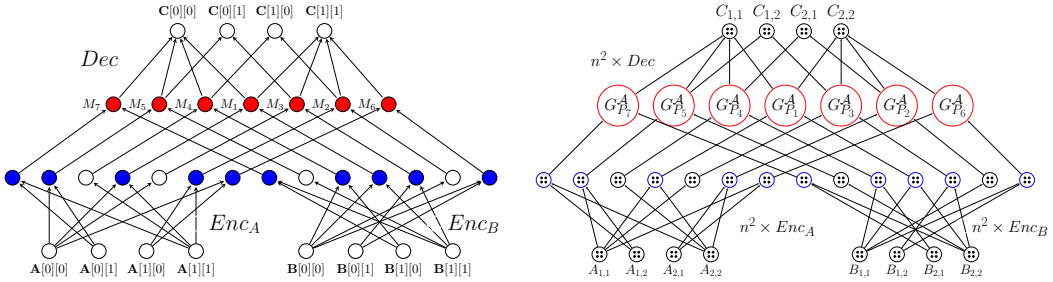
We also consider a sub-class $\mathfrak{UH}(n_0)$ of $\mathfrak{H}$ constituted by *uniform, non-stationary* hybrid algorithms which allow mixing of schemes from the fast Strassen-like class for the initial ℓ recursion levels, and then cut the recursion off once the size the generated sub-problems is smaller or equal to a set threshold $n_0 \times n_0$, and switch to using algorithm form the standard class. Algorithms in this class are *uniform*, i.e., sub-problems of the same size are all either recursively computed using a scheme form the fast class, or are all computed using algorithms from the standard class.

4 The CDAG of algorithms in $\mathfrak{H}$

Let $G^{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ denote the CDAG that corresponds to an algorithm $\mathcal{A} \in \mathfrak{H}$ used for multiplying input matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$. The challenge in the characterization of $G^{\mathcal{A}}$ comes from the fact that rather than considering to a single algorithm, we want to characterize the CDAG corresponding to the class $\mathfrak{H}$. Further, the class $\mathfrak{H}$ is composed of a rich variety of vastly different and irregular algorithm. Despite such variety, we show a general template for the construction of $G^{\mathcal{A}}$ and we identify some of its properties which crucially hold *regardless* of the implementation details of $\mathcal{A}$ and, hence, of $G^{\mathcal{A}}$.

Construction. $G^{\mathcal{A}}$ can be obtained by using a recursive construction that mirrors the recursive structure of the algorithm itself. Let P denote the entire matrix multiplication problem computed by $\mathcal{A}$. Consider the case for which, according to the *instruction function* $f_{\mathcal{A}}(P)$, P is to be computed using an algorithm from the standard class. As we do not fix a specific algorithm, we do not correspondingly have a fixed CDAG. The only feature of interest for the analysis is that, in this case, the CDAG $G^{\mathcal{A}}$ corresponds to the execution of an algorithm from the standard class for input matrices of size $n \times n$.

Consider instead the case for which, according to $f_{\mathcal{A}}(P)$, P is to be computed using an algorithm from the fast class. In the base case for $n = 2$ the problem P is computed without generating any further sub-problems. As an example, we present in Figure 1a the base case for Strassen's original algorithm [36]. If $n > 2$, then $f_{\mathcal{A}}(P)$ specifies the divide and conquer scheme to be used to generate the seven sub-problems $P_1, P_2, \ldots, P_7$, and the *instruction function* for each of them. The sub-CDAGs of $G^{\mathcal{A}}$ corresponding to each of the seven sub-problems P_i, denoted as $G^{\mathcal{A}}_{P_i}$ are constructed according to $f_{\mathcal{A}}(P_i)$, following recursively the steps discussed previously. $G^{\mathcal{A}}$ can then be constructed by composing the seven sub-CDAGs $G^{\mathcal{A}}_{P_i}$. n^2 disjoint copies of an *encoder sub-CDAG* Enc_A (resp., Enc_B) are used to connect the input vertices of $G^{\mathcal{A}}$, which correspond to the values of the input matrix $\mathbf{A}$ (resp., $\mathbf{B}$) to the appropriate input vertices of the seven sub-CDAGs $G^{\mathcal{A}}_{P_i}$; the output vertices of the sub-CDAGs $G^{\mathcal{A}}_{P_i}$ (which correspond to the outputs of the seven sub-products) are connected to the appropriate output vertices of the entire $G^{\mathcal{A}}$ CDAG using n^2 copies of the decoder sub-CDAG *Dec*. We present an example of such recursive construction in Figure 1b.



(a) $G^{\mathcal{A}}$ CDAG for base case $n = 2$, using Strassen's algorithm [36] (for details see extended version [18]).

(b) Recursive construction of $G^{\mathcal{A}}$. $A_{i,j}$, $B_{i,j}$ and $C_{i,j}$ denote the block-partition of $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$.

Figure 1 Blue vertices represent combinations of the input values from the factor matrices $\mathbf{A}$ and $\mathbf{B}$ used as input values for the seven sub-problems; red vertices represent the output of the seven sub-problems which are used to compute the values of the output matrix $\mathbf{C}$.

Properties of $G^{\mathcal{A}}$. While the actual internal structure $G^{\mathcal{A}}$, and, in particular, the structure of encoder and decoder sub-CDAGs depends on the specific Strassen-like algorithm being used by $\mathcal{A}$, all versions share some properties of great importance. Let $G(X, Y, E)$ denote an encoder CDAG for a fast multiplication algorithm 2×2 base case, with X (resp., Y) denoting the set of input (resp., output) vertices, and E denoting the set of edges directed from X to Y.

▶ **Lemma 2** (Lemma 3.3 [30]). *Let $G = (X, Y, E)$ denote an encoder graph for a fast matrix multiplication algorithm with base case 2×2. There are no two vertices in Y with identical neighbors sets.*

While the correctness of this Lemma can be simply verified by inspection in the case of Strassen's algorithm [36], Lemma 2 generalizes the statement to *all* encoders corresponding to fast matrix multiplication algorithms with base case 2×2. From Lemma 2 we have:

▶ **Lemma 3.** *Let $\mathcal{A} \in \mathfrak{H}$ and let P_1 and P_2 be any two sub-problems generated by $\mathcal{A}$ with input size greater than $n_0 \times n_0$, such that P_2 is not recursively generated while computing P_1 and vice versa. Then, the sub-CDAGs of $G^{\mathcal{A}}$ corresponding, respectively, to P_1 and to P_2 are vertex-disjoint.*

The following lemma, originally introduced for Strassen's algorithm in [10] and then generalized for Strassen-like algorithms with base case 2×2 in in [30], captures a connectivity property of encoder sub-CDAGs.

▶ **Lemma 4** (Lemma 3.1 [30]). *Given an encoder CDAG for any Strassen-like algorithm with base case 2×2, for any subset Y of its output vertices, there exists a subset X of its input vertices, with $\min\{|Y|, 1 + \lceil (|Y| - 1)/2 \rceil\} \leq |X| \leq |Y|$, such that there exist $|X|$ vertex-disjoint paths connecting the vertices in X to vertices in Y.*

The proofs of Lemma 2 and Lemma 4 are based on an argument originally presented by Hopcroft and Kerr [22]. We refer the reader to [30] for the proofs.

5 Maximal sub-problems and their properties

For an algorithm $\mathcal{A} \in \mathfrak{H}$, let P' denote a sub-problem generated by $\mathcal{A}$. In our presentation we consider the entire matrix multiplication problem as an *improper* sup-problem generated by $\mathcal{A}$. Given a sub=problem P' let $P'_0, P'_2, \ldots, P'_i$ be the sequence of sub-problems generated

by $\mathcal{A}$ such that P'_{j+1} was recursively generated to compute P'_j for $j = 0, 1, \ldots, i-1$, and such that P' was recursively generated to compute P'_i. We refer to the sub-problems in such sequence as the *ancestor sub-problems* of P'. If P' is the entire problem, it has no ancestors.

Towards studying the I/O complexity of algorithms in $\mathfrak{H}$ we focus on the analysis of a particular set of sub-problems.

▶ **Definition 5** (Maximal Sub-Problems (MSP)). *Let $\mathcal{A} \in \mathfrak{H}$ be an algorithm used to multiply matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$. If $n \leq 2\sqrt{M}$ we say that $\mathcal{A}$ does not generate any Maximal Sub-Problem (MSP).*

Let P_i be a sub-problem generated by $\mathcal{A}$ with input size $n_i \times n_i$, with $n_i \geq 2M$, and such that all its ancestors sub-problems are computed, according to $\mathcal{A}$ using algorithms from the fast class. We say that:

- *P_i is a Type 1 MSP of $\mathcal{A}$ if, according to $\mathcal{A}$, is computed using an algorithm from the standard class. If the entire problem is solved using an algorithm from the standard class, we say that the entire problem is the unique (improper) Type 1 MSP generated by $\mathcal{A}$.*
- *P_i is a Type 2 MSP of $\mathcal{A}$ if, according to $\mathcal{A}$, is computed by generating 7 sub-problems according to the recursive scheme corresponding to an algorithm from the fast (Strassen-like) class, and if the generated sub-problems have input size strictly smaller than $2\sqrt{M} \times 2\sqrt{M}$. If the entire problem uses a recursive algorithm from the fast class to generate 7 sub-problems with input size smaller than $2\sqrt{M} \times 2\sqrt{M}$, we say that the entire problem is the unique, improper, Type 2 MSP generated by $\mathcal{A}$.*

In the following we denote as ν_1 (resp., ν_2) the number of Type 1 (resp., Type 2) MSPs generated by $\mathcal{A}$.

Let P_i denote the i-th MSP generated by $\mathcal{A}$ and let $G^{\mathcal{A}}_{P_i}$ denote the corresponding sub-CDAG of $G^{\mathcal{A}}$. We denote as $\mathbf{A}_i$ and $\mathbf{B}_i$ (resp., $\mathbf{C}_i$) the input factor matrices (resp., the output product matrix) of P_i.

Properties of MSPs and their corresponding sub-CDAGs. By Definition 5, we have that for each pair of distinct MMSPs P_1 and P_2, P_2 is not recursively generated by $\mathcal{A}$ in order to compute P_1 or vice versa. Hence, by Lemma 3, the sub-CDAGs of $G^{\mathcal{A}}$ that correspond each to one of the MSPs generated by $\mathcal{A}$ are vertex-disjoint.

In order to obtain our I/O lower bound for algorithms in $\mathfrak{H}$, we characterize properties regarding the minimum dominator size of an arbitrary subset of $\mathcal{Y}$ and $\mathcal{Z}$.

▶ **Lemma 6.** *Let $G^{\mathcal{A}}$ be the CDAG corresponding to an algorithm $\mathcal{A} \in \mathfrak{H}$ which admits n_1 Type 1 MSPs. For each Type 1 MSP P_i let $\mathcal{Y}_i$ denote the set of input vertices of the associated sub-CDAG $G^{\mathcal{A}}_{P_i}$ which correspond each to an entry of the input matrices $\mathbf{A}_i$ and $\mathbf{B}_i$. Further, we define $\mathcal{Y} = \cup_{i=1}^{\nu_1} \mathcal{Y}_i$.*

Let $Y \subseteq \mathcal{Y}$ in $G^{\mathcal{A}}$ such that $|Y \cap \mathcal{Y}_i| = a_i/\sqrt{b_i}$, with $a_i, b_i \in \mathbb{N}$, $a_i \geq b_i$ for $i = 1, 2, \ldots, \nu_1$, and such that $b_i = 0$ if and only if $a_i = 0$.[1] *Any dominator set D of Y satisfies $|D| \geq \min\{2M, \sum_{i=1}^{\nu_1} a_i / \sqrt{\sum_{i=1}^{\nu_1} b_i}\}$.*

▶ **Lemma 7.** *Let $G^{\mathcal{A}}$ be the CDAG corresponding to an algorithm $\mathcal{A} \in \mathfrak{H}$ which admits n_2 Type 2 MSPs. Further let $\mathcal{Z}$ denote the set of vertices corresponding to the entries of the output matrices of the n_2 Type 2 MSPs. Given any subset $Z \subseteq \mathcal{Z}$ in $G^{\mathcal{A}}$ with $|Z| \leq 4M$, any dominator set D of Z satisfies $|D| \geq |Z|/2$.*

[1] Here we use as convention that $0/0 = 0$.

For each Type 1 MSP P_i generated by $\mathcal{A}$, with input size[2] $n_i \times n_i$, we denote as T_i the set of variables whose value correspond to the n_i^3 elementary products $\mathbf{A}_i[j][k]\mathbf{B}_i[k][j]$ for $j, k = 0, 1, \ldots, n_i - 1$. Further, we denote as $\mathcal{T}_i$ the set of vertices corresponding to the variables in T_i, and we define $\mathcal{T} = \cup_{i=1}^{\nu_1} \mathcal{T}_i$.

▶ **Lemma 8.** *For any Type 1 MSPs generated by $\mathcal{A}$ consider $T'_i \subseteq T_i$. Let $\mathcal{Y}_i^{(\mathbf{A})} \subseteq \mathcal{Y}_i$ (resp., $\mathcal{Y}_i^{(\mathbf{B})} \subseteq \mathcal{Y}_i$) denote a subset of the vertices corresponding to entries of $\mathbf{A}_i$ (resp., $\mathbf{B}_i$) which are multiplied in at least one of the elementary products in T'_i. Then any dominator D of the vertices corresponding to T'_i with respect to the the vertices in $\mathcal{Y}_i$ is such that*

$$|D| \geq \max\{|\mathcal{Y}'_i \cap A_i|, |\mathcal{Y}'_i \cap B_i|\}.$$

For the proofs of Lemmas 6, 7 and 8 we refer the reader to the extended version of this work [18]. The proofs are based on the analysis of the combinatorial properties of Strassen-like algorithms, and on the analysis of the *Grigoriev's information flow* of the square matrix multiplication function [21, 31].

6 I/O lower bounds for algorithms in $\mathfrak{H}$ and $\mathfrak{UH}(n_0)$

▶ **Theorem 9.** *Let $\mathcal{A} \in \mathfrak{H}$ be an algorithm to multiply two square matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$. If run on a sequential machine with cache of size M and such that up to B memory words stored in consecutive memory locations can be moved from cache to slow memory and vice versa using a single memory operation, $\mathcal{A}$'s I/O complexity satisfies:*

$$IO_{\mathcal{A}}(n, M, B) \geq \max\{2n^2, c|\mathcal{T}|M^{-1/2}, \nu_2 M\}B^{-1} \tag{1}$$

for $c = 0.38988157484$, where $|\mathcal{T}|$ denotes the total number of internal elementary products computed by the Type 1 MSPs generated by $\mathcal{A}$ and ν_2 denotes the total number of Type 2 MSPs generated by $\mathcal{A}$.

If run on P processors each equipped with a local memory of size $M < n^2$ and where for each I/O operation it is possible to move up to $B_m \leq M$ words, $\mathcal{A}$'s I/O complexity satisfies:

$$IO_{\mathcal{A}}(n, M, B_m, P) \geq \max\{c|\mathcal{T}|M^{-1/2}, \nu_2 M\}(PB_m)^{-1} \tag{2}$$

Proof. We prove the result in (1) (resp., (2)) for the case $B = 1$ (resp., $B_m = 1$). The result then trivially generalizes for a generic B (resp., B_m). We first prove the result for the sequential case in in (1). The bound for the parallel case in (2) will be obtained as a simple generalization. For simplicity of presentation, we assume $\sqrt{M} \in \mathbb{N}^+$.

The fact that $IO_{\mathcal{A}}(n, M, 1) \geq 2n^2$ follows trivially from the fact that as in our model the input matrices $\mathbf{A}$ and $\mathbf{B}$ are initially stored in slow memory, it will necessary to move the entire input to the cache at least once using at least $2n^2$ I/O operations. If $\mathcal{A}$ does not generate any MSPs the statement in (1) is trivially verified. In the following, we assume $\nu_1 + \nu_2 \geq 1$.

Let $G^{\mathcal{A}}$ denote the CDAG associated with algorithm $\mathcal{A}$ according to the construction in Section 4. By definition, and from Lemma 3, the $\nu_1 + \nu_2$ sub-CDAGs of $G^{\mathcal{A}}$ corresponding each to one of the MSPs generated by $\mathcal{A}$ are vertex-disjoint. Hence, the $\mathcal{T}_i$'s are a partition of $\mathcal{T}$ and $|\mathcal{T}| = \sum_{i=1}^{\nu_1} |\mathcal{T}_i|$.

[2] In general, different Type 1 MSP may have different input sizes

By Definition 5, the MSP generated by $\mathcal{A}$ have input (resp., output) matrices of size greater or equal to $2\sqrt{M} \times 2\sqrt{M}$. Recall that we denote as $\mathcal{Z}$ the set of vertices which correspond to the outputs of the ν_2 Type 2 MSPs, we have $|\mathcal{Z}| \geq 4M\nu_l$.

Let $\mathcal{C}$ be any computation schedule for the sequential execution of $\mathcal{A}$ using a cache of size M. We partition $\mathcal{C}$ into non-overlapping segments $\mathcal{C}_1, \mathcal{C}_2, \ldots$ such that during each $\mathcal{C}_j$ either (a) exactly $M^{3/2}$ distinct values corresponding to vertices in $\mathcal{T}$, denoted as $\mathcal{T}^{(j)}$, are explicitly computed (i.e., not loaded from slow memory), or (b) $4M$ distinct values corresponding to vertices in $\mathcal{Z}$ (denoted as $\mathcal{Z}_j$) are evaluated for the *first time*. Clearly there are at least $\max\{|\mathcal{T}|/M^{3/2}, \nu_2\}$ such segments. Below we show that the number g_j of I/O operations executed during each $\mathcal{C}_j$ satisfies $g_j \geq cM$ for case (a) and $g_j \geq M$ for case (b), from which the theorem follows.

Case (a): For each Type 1 MSP P_i let $\mathcal{T}_i^{(j)} = \mathcal{T}^{(j)} \cap \mathcal{T}_i$. As the ν_1 sub-CDAGs corresponding each to one of the Type 1 MSPs are vertex-disjoint, so are the sets $\mathcal{T}_i$. Hence, the $\mathcal{T}_i^{(j)}$'s constitute a partition of $\mathcal{T}^{(j)}$. Let $\mathbf{A}_i$ and $\mathbf{B}_i$ (resp., $\mathbf{C}_i$) denote the input matrices (resp., output matrix) of P_i with $\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i \in \mathcal{R}^{n_i \times n_i}$, and let A_i and B_i (resp., C_i) denote the set of the variables corresponding to the entries of $\mathbf{A}_i$ and $\mathbf{B}_i$ (resp., $\mathbf{C}_i$). Further, we denote as T_i the set of values corresponding to the vertices in $\mathcal{T}_i$. For $r, s = 0, 1, \ldots, n_i - 1$, we say that $\mathbf{C}_i[r][s]$ is "*active during* $\mathcal{C}_j$" if *any* of the elementary multiplications $\mathbf{A}_i[r][k]\mathbf{B}_i[k][s]$, for $k = 0, 1, \ldots, n_i - 1$, correspond to any of the vertices in $\mathcal{T}_i^{(j)}$. Further we say that a $\mathbf{A}_i[r][s]$ (resp., $\mathbf{B}_i[r][s]$) is "*accessed during* $\mathcal{C}_j$" if *any* of the elementary multiplications $\mathbf{A}_i[r][s]\mathbf{B}_i[s][k]$ (resp., $\mathbf{A}_i[k][r]\mathbf{B}_i[r][s]$), for $k = 0, 1, \ldots, n_i - 1$, correspond to any of the vertices in $\mathcal{T}_i^{(j)}$. Our analysis makes use of the following property of standard matrix multiplication algorithms:

▶ **Lemma 10** (Loomis-Whitney inequality [24, Lemma 2.2]). *Let $\mathcal{Y}'_{i,\mathbf{A}}$ (resp., $\mathcal{Y}'_{i,\mathbf{B}}$) denote the set of vertices corresponding to the entries of $\mathbf{A}_i$ (resp., $\mathbf{B}_i$) which are accessed during $\mathcal{C}_j$, and let $\mathcal{Z}'_i$ denote the set of vertices corresponding to the entries of $\mathbf{C}_i$ which are active during $\mathcal{C}_j$. Then*

$$|\mathcal{T}_i^{(j)}| \leq \sqrt{|\mathcal{Y}'_{i,\mathbf{A}}||\mathcal{Y}'_{i,\mathbf{B}}||\mathcal{Z}'_i|}. \tag{3}$$

Lemma 10 is a reworked version of a property originally presented by Irony et al. [24, Lemma 2.2], which itself is a consequence of the Loomis-Whitney geometric theorem [29]. We refer the reader to [24, Lemma 2.2] for the proof of of Lemma 10.

Let $\mathbf{C}_i[r][s]$ be active during $\mathcal{C}_j$. In order to compute $\mathbf{C}_i[r][s]$ *entirely* during $\mathcal{C}_j$ (i.e., without using partial accumulation of the summation $\sum_{k=0}^{n_i-1} \mathbf{A}_i[r][k]\mathbf{B}[k][s]$), it will be necessary to evaluate all the n_i elementary products $\mathbf{A}_i[r][k]\mathbf{B}_i[k][s]$, for $k = 0, 1, \ldots, n_i - 1$, during $\mathcal{C}_j$ itself. Thus, at most $\lfloor |\mathcal{T}_i^{(j)}|/n_i \rfloor$ entries of $\mathbf{C}_i[r][s]$ can be entirely computed during $\mathcal{C}_j$.

Let $\mathbf{C}_i[r][s]$ denote an entry of $\mathbf{C}_i$ which is active but not entirely computed during $\mathcal{C}_j$. There are two possible scenarios:

- $\mathbf{C}_i[r][s]$ is computed during $\mathcal{C}_j$: The computation thus requires for a partial accumulation of $\sum_{k=0}^{n_i-1} \mathbf{A}_i[r][k]\mathbf{B}[k][s]$ to have been previously computed and either held in the cache at the beginning of $\mathcal{C}_j$, or to moved to cache using a `read` I/O operation during $\mathcal{C}_j$;
- $\mathbf{C}_i[r][s]$ is not computed during $\mathcal{C}_j$: As $\mathcal{C}$ is a parsimonious computation, the partial accumulation of $\sum_{k=0}^{n_i-1} \mathbf{A}_i[r][k]\mathbf{B}[k][s]$ obtained from the elementary products computed during $\mathcal{C}_j$ must either remain in the cache at the end of $\mathcal{C}_j$, or be moved to slow memory using a `write` I/O operation during $\mathcal{C}_j$;

In both cases, any partial accumulation either held in memory at the beginning (resp., end) of $\mathcal{C}_j$ or read from slow memory to cache (resp., written from cache to slow memory) during $\mathcal{C}_j$ is, by definition, not shared between multiple entries in $\mathbf{C}_i$.

Let $G^{\mathcal{A}}_{P_i}$ denote the sub-CDAG of $G^{\mathcal{A}}$ corresponding to the Type 1 MSP P_i. In the following, we refer as D'_i to the set of vertices of $G^{\mathcal{A}}_{P_i}$ corresponding to the values of such partial accumulators. For each of the least $|D'_i| = \max\{0, |\mathcal{Z}'_i| - |\mathcal{T}_i^{(j)}|/n_i\}$ entries of $\mathbf{C}_i$ which are active but not entirely computed during $\mathcal{C}_j$, either one of the entries of the cache must be occupied at the beginning of $\mathcal{C}_j$, or one I/O operation is executed during $\mathcal{C}_j$. Let $D' = \cup_{i=1}^{\nu_1} D'_i$. As, by Lemma 3, the sub-CDAGs corresponding to the ν_1 Type 1 MSPs are vertex-disjoint, so are the the sets D'_i. Let $\mathcal{Z}' = \sum_{i=1}^{\nu_l} |\mathcal{Z}'_i|$. We have:

$$|D'| = \sum_{i=1}^{\nu_1} |D'_i| = \sum_{i=1}^{\nu_1} \max\{0, |\mathcal{Z}'_i| - |\mathcal{T}_i^{(j)}|/n_i\} \geq |\mathcal{Z}| - |\mathcal{T}^{(j)}|/2\sqrt{M}, \tag{4}$$

where the last passage follows from the fact that, by Definition 5, $n_i \geq 2M$.

From Lemma 10, the set of vertices $\mathcal{Y}'_{i,\mathbf{A}}$ (resp., $\mathcal{Y}'_{i,\mathbf{B}}$) which correspond to entries of $\mathbf{A}_i$ (resp., $\mathbf{B}_i$) which are accessed during $\mathcal{C}_j$ satisfies $|\mathcal{Y}'_{i,\mathbf{A}}||\mathcal{Y}'_{i,\mathbf{B}}| \geq |\mathcal{T}_i^{(j)}|^2/|\mathcal{Z}'_i|$. Hence, at least $|\mathcal{Y}'_{i,\mathbf{A}}| + |\mathcal{Y}'_{i,\mathbf{B}}| \geq 2|\mathcal{T}_i^{(j)}|/\sqrt{|\mathcal{Z}'_i|}$ entries from the input matrices of P_i are accessed during $\mathcal{C}_j$. Let $\mathcal{Y}$ denote the set of vertices corresponding to the entries of the input matrices $\mathbf{A}_i, \mathbf{B}_i$ of P_i. From Lemma 8 we have that there exists a set $\mathcal{Y}'_i \subseteq \mathcal{Y}_i$, with$|\mathcal{Y}'_i| \geq \max\{|\mathcal{Y}'_{i,\mathbf{A}}|, |\mathcal{Y}'_{i,\mathbf{B}}|\} \geq |\mathcal{T}_i^{(j)}|/\sqrt{|\mathcal{Z}'_i|}$, such that the vertices in $\mathcal{Y}'_i$ are connected by vertex-disjoint pats to the vertices in $\mathcal{T}_i^{(j)}$. Let $Y = \cup_{i=1}^{\nu_1} \mathcal{Y}'_i$. As, by Lemma 3, the sub-CDAGs corresponding to the ν_l Type 1 MSPs are vertex-disjoint, so are the the sets $\mathcal{Y}'_i$ for $i = 1, 2, \ldots, \nu_1$. Hence

$$|Y| = \sum_{i=1}^{\nu_1} |\mathcal{Y}'_i| \geq \sum_{i=1}^{\nu_1} \frac{|\mathcal{T}_i^{(j)}|}{\sqrt{|\mathcal{Z}'_i|}}.$$

From Lemma 6 any dominator D_Y of Y, must be such that

$$|D_Y| \geq \min\Big\{2M, \frac{\sum_{i=1}^{\nu_1} |\mathcal{T}_i^{(j)}|}{\sum_{i=1}^{\nu_1} \sqrt{|\mathcal{Z}'_i|}}\Big\} = \min\Big\{2M, \frac{|\mathcal{T}^{(j)}|}{\sqrt{|\mathcal{Z}'|}}\Big\}.$$

Hence, we can conclude that any dominator D'' of $\mathcal{T}^{(j)}$ must be such that

$$|D''| \geq \min\big\{2M, |\mathcal{T}^{(j)}|/\sqrt{|\mathcal{Z}'|}\big\}. \tag{5}$$

Consider the set D of vertices of $G_{\mathcal{A}}$ corresponding to the at most M values stored in the cache at the beginning of $\mathcal{C}_j$ and to the at most g_j values loaded into the cache form the slow memory (resp., written into the slow memory from the cache) during $\mathcal{C}_j$ by means of a `read` (resp., `write`) I/O operation. Clearly, $|D| \leq M + g_j$.

In order for the $M^{3/2}$ values from $\mathcal{T}^{(j)}$ to be computed during $\mathcal{C}_j$ there must be no path connecting any vertex in $\mathcal{T}^{(j)}$, and, hence, Y, to any input vertex of $G^{\mathcal{A}}$ which does not have at least one vertex in D, that is D has to admit a subset $D'' \subseteq D$ such that D'' is a *dominator set* of $\mathcal{T}^{(j)}$. Note that, as the values corresponding to vertices in $\mathcal{T}^{(j)}$ are actually computed during C_J (i.e., not loaded from memory using a `read` I/O operation), D'' does not include vertices in $\mathcal{T}^{(j)}$ itself. Further, as motivated in the previous discussion, D must include all the vertices in the set D' corresponding to values of partial accumulators of the active output values of Type 1 MSPs during $\mathcal{C}_j$.

By construction, D' and D'' are vertex-disjoint. Hence, from (4) and (5) we have:

$$|D| \geq |D'| + |D''| \geq |\mathcal{Z}'| - |\mathcal{T}^{(j)}|/2\sqrt{M} + \min\big\{2M, |\mathcal{T}^{(j)}|/\sqrt{|\mathcal{Z}'|}\big\}.$$

As, by construction, $|\mathcal{T}^{(j)}| = M^{3/2}$, we have:

$$|D| > |\mathcal{Z}'| - M/2 + \min\{2M, M^{3/2}/\sqrt{|\mathcal{Z}'|}\}. \tag{6}$$

By studying its derivative after opportunely accounting for the minimum, we have that (6) is minimized for $|\mathcal{Z}'| = 2^{-2/3}M$. Hence we have: $|D| > 2^{-2/3}M + 2^{1/3}M^{3/2} - M/2 = 1.38988157484M$. Whence $|D| \leq M + g_j$, which implies $g_j \geq |D| - M > 0.38988157484M$, as stated above.

Case (b): In order for the $4M$ values from $\mathcal{Z}_j$ to be computed during $\mathcal{C}_j$ there must be no path connecting any vertex in $\mathcal{Z}_j$ to any input vertex of $G_{\mathcal{A}}$ which does not have at least one vertex in D_j, that is D_j has to be a *dominator set* of $\mathcal{Z}_j$. From Lemma 7, any dominator set D of any subset $Z \subseteq \mathcal{Z}$ with $|Z| \leq 4M$ satisfies $|D| \geq |Z|/2$, whence $M + g_i \geq |D_i| \geq |\mathcal{Z}_j|/2 = 2M$, which implies $g_j \geq M$ as stated above. This concludes the proof for the sequential case in (1).

The proof for the bound for the parallel model in (2), follows from the observation that at least one of the P processors, denoted as P^*, must compute at least $|\mathcal{T}|/P$ values corresponding to vertices in $\mathcal{T}$ or $|\mathcal{Z}|/P$ values corresponding to vertices in $\mathcal{Z}$ (or both). The bound follows by applying the same argument discussed for the sequential case to the computation executed by P^*. ◀

Note that if $\mathcal{A}$ is such that the product is entirely computed using an algorithm from the standard class (resp., a fast matrix multiplication algorithm), the bounds of Theorem 9 corresponds asymptotically to the results in [25] for the sequential case, and in [24] for the parallel case (resp., the results in [10]).

For the sub-class of uniform, non stationary algorithms $\mathfrak{U}\mathfrak{H}(n_0)$, given the values of n, M and n_0 is possible to compute a closed form expression for the values of ν_1, ν_2 and $|\mathcal{T}|$. Then, by applying Theorem 9 we have:

▶ **Theorem 11.** *Let $\mathcal{A} \in \mathfrak{U}\mathfrak{H}(n_0)$ be an algorithm to multiply two square matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n\times n}$. If run on a sequential machine with cache of size M and such that up to B memory words stored in consecutive memory locations can be moved from cache to slow memory and vice versa using a single memory operation, $\mathcal{A}$'s I/O complexity satisfies:*

$$IO_{\mathcal{A}}(n, M, B) \geq \max\{2n^2, \left(\frac{n}{\max\{n_0, 2\sqrt{M}\}}\right)^{\log_2 7} \left(\max\{1, \frac{n_0}{2\sqrt{M}}\}\right)^3 M\} B^{-1} \tag{7}$$

If run on P processors each equipped with a local memory of size $M < n^2$ and where for each I/O operation it is possible to move up to B_m memory words, $\mathcal{A}$'s I/O complexity satisfies:

$$IO_{\mathcal{A}}(n, M, B_m, P) \geq \left(\frac{n}{\max\{n_0, 2\sqrt{M}\}}\right)^{\log_2 7} \left(\max\left\{1, \frac{n_0}{2\sqrt{M}}\right\}\right)^3 \frac{M}{PB_m}. \tag{8}$$

Proof. The proof follows by bounding the values ν_1,ν_2 and $|\mathcal{T}|$ for $\mathcal{A} \in \mathfrak{U}\mathfrak{H}(n_0)$, and by applying the general result from Theorem 9. To simplify the presentation, in the following we assume that the values n, n_0 and$\sqrt{M}$ are powers of two. If that is not the case, the theorem holds with some minor adjustments to the constant multiplicative factor.

Let let i be the smallest value in $\mathbb{N}$ such that $n/2^i = \max\{n_0, 2\sqrt{M}\}$. By definition of $\mathfrak{H}$, at each of the i recursive levels $\mathcal{A}$ generates 7^i sub-problems of size $n/2^i$.

- If $n_0 > 2\sqrt{M}$, $\mathcal{A}$ generates $\nu_1 = 7^i = 7^{\log_2 n/n_0} = (n/n_0)^{\log_2 7}$ Type 1 MSP each with input size $n_0 \times n_0$. As, by Definition 5, the Type 1 MSP are input-disjoint we have $|\mathcal{T}| = \nu_1 {n_0}^3 = (n/n_0)^{\log_2 7} {n_0}^3$.

- Otherwise, if $n_0 \leq 2\sqrt{M}$, $\mathcal{A}$ generates $\nu_2 = 7^i = 7^{\log_2 n/2\sqrt{M}} = \left(n/2\sqrt{M}\right)^{\log_2 7}$ Type 2 MSP each with input size $2\sqrt{M} \times 2\sqrt{M}$.

The statement then follows by applying the result in Theorem 9. ◀

The constants terms in (7) and (8) hold under the assumption that n, n_0 and $\sqrt{M}$ are powers of two. If that is not the case the statement holds with minor adjustments to said constant factors.

Theorem 11 extends the result by Scott [34] by expanding the class of hybrid matrix multiplication algorithms being considered (e.g., it does not limit the class of standard matrix multiplication to the divide and conquer algorithm based on block-partitioning), and by removing the assumption that no intermediate value may be recomputed.

On the tightness of the bound. An opportune composition of the cache-optimal version of the Strassen's algorithm [36] (as discussed in [5]) with the standard cache-optimal divide and conquer algorithm for square matrix multiplication based on block-partitioning [15] leads to a sequential hybrid algorithms in $\mathfrak{H}$ (resp., $\mathfrak{UH}(n_0)$) whose I/O cost asymptotically matches the I/O complexity lower bounds in Theorem 9 (1) (resp., Theorem 11 (7)).

Parallel algorithms in $\mathfrak{H}$ (resp., $\mathfrak{UH}(n_0)$) asymptotically matching the I/O lower bounds in for the parallel case in Theorem 9 (2) (resp., Theorem 11 (8)) can be obtained by composing the communication avoiding version of Strassen's algorithm by Ballard et al. [5] with the communication avoiding "*2.5*" standard algorithm by Solomonik and Demmel [35].

Hence, the lower bounds in Theorem 9 and Theorem 11 are asymptotically tight and the mentioned algorithms form $\mathfrak{H}$ and $\mathfrak{UH}(n_0)$ whose I/O cost asymptotically match the lower bounds are indeed I/O optimal. Further, as the mentioned I/O optimal algorithms from $\mathfrak{H}$ and $\mathfrak{UH}(n_0)$ do not recompute any intermediate value, we can conclude that using recomputation may lead to at most a constant factor reduction of the I/O cost of hybrid algorithms in $\mathfrak{H}$ and $\mathfrak{UH}(n_0)$. Note that the replication of the input used by the mentioned algorithms as recomputation it should be considered as *repeated access* to the input values, and not as recomputation.

Generalization to fast matrix multiplication model with base other than 2×2. The general statement of Theorem 9 can be extended by enriching $\mathfrak{H}$ to include any fast Strassen-like algorithm with base case other than 2×2 provided that the associated encoder CDAG satisfies properties equivalent to those expressed by Lemma 3 (i.e., the input disjointedness of the sub-problems generated at each recursive step) and Lemma 4 (i.e., the connectivity between input and output of the encoder CDAGs via vertex disjoint paths) for the 2×2 base. If these properties hold, so does the general structure of Theorem 9, given an opportune adjustment of the definition of maximal sub-problem.

7 Conclusion

This work contributed to the characterization of the I/O complexity of hybrid matrix multiplication algorithms combining fast Strassen-like algorithms with standard algorithms. We established asymptotically tight lower bounds that hold even when recomputation is allowed. The generality of the technique used for the analysis makes it promising for the analysis of other hybrid recursive algorithms, e.g., hybrid algorithms for integer multiplication [11].

References

1. Alok Aggarwal, Bowen Alpern, Ashok Chandra, and Marc Snir. A model for hierarchical memory. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 305–314. ACM, 1987.
2. Alok Aggarwal, Jeffrey Vitter, et al. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
3. G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Graph expansion analysis for communication costs of fast rectangular matrix multiplication. In *Design and Analysis of Algorithms*, pages 13–36. Springer, 2012.
4. Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 77–79. ACM, 2012.
5. Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 193–204. ACM, 2012.
6. Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Communication-optimal parallel and sequential Cholesky decomposition. *SIAM Journal on Scientific Computing*, 32(6):3495–3523, 2010.
7. Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32(3):866–901, 2011.
8. Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM (JACM)*, 59(6):32, 2012.
9. Sandeep N Bhatt, Gianfranco Bilardi, and Geppino Pucci. Area-time tradeoffs for universal VLSI circuits. *Theoretical Computer Science*, 408(2-3):143–150, 2008.
10. Gianfranco Bilardi and Lorenzo De Stefani. The I/O complexity of Strassen's matrix multiplication with recomputation. In *Workshop on Algorithms and Data Structures*, pages 181–192. Springer, 2017.
11. Gianfranco Bilardi and Lorenzo De Stefani. The I/O complexity of Toom-Cook integer multiplication. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2034–2052. SIAM, 2019.
12. Gianfranco Bilardi and Enoch Peserico. A characterization of temporal locality and its portability across memory hierarchies. In *International Colloquium on Automata, Languages, and Programming*, pages 128–139. Springer, 2001.
13. Gianfranco Bilardi and Franco P Preparata. Horizons of parallel computation. *Journal of Parallel and Distributed Computing*, 27(2):172–182, 1995.
14. Gianfranco Bilardi and Franco P Preparata. Processor—Time Tradeoffs under Bounded-Speed Message Propagation: Part II, Lower Bounds. *Theory of Computing Systems*, 32(5):531–559, 1999.
15. Lynn Elliot Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis, Montana State University-Bozeman, College of Engineering, 1969.
16. National Research Council et al. *Getting up to speed: The future of supercomputing*. National Academies Press, 2005.
17. Lorenzo De Stefani. *On space constrained computations*. PhD thesis, University of Padova, 2016.
18. Lorenzo De Stefani. The I/O complexity of hybrid algorithms for square matrix multiplication. *arXiv preprint*, 2019. `arXiv:1904.12804`.

19 Frédéric Desprez and Frédéric Suter. Impact of mixed-parallelism on parallel implementations of the Strassen and Winograd matrix multiplication algorithms. *Concurrency and Computation: practice and experience*, 16(8):771–797, 2004.

20 Craig C Douglas, Michael Heroux, Gordon Slishman, and Roger M Smith. GEMMW: a portable level 3 BLAS Winograd variant of Strassen's matrix-matrix multiply algorithm. *Journal of Computational Physics*, 110(1):1–10, 1994.

21 Dmitrii Yur'evich Grigor'ev. Application of separability and independence notions for proving lower bounds of circuit complexity. *Zapiski Nauchnykh Seminarov POMI*, 60:38–48, 1976.

22 John E Hopcroft and Leslie R Kerr. On minimizing the number of multiplications necessary for matrix multiplication. *SIAM Journal on Applied Mathematics*, 20(1):30–36, 1971.

23 Steven Huss-Lederman, Elaine M Jacobson, Jeremy R Johnson, Anna Tsao, and Thomas Turnbull. Implementation of Strassen's algorithm for matrix multiplication. In *Supercomputing'96: Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, pages 32–32. IEEE, 1996.

24 Dror Irony, Sivan Toledo, and Alexander Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

25 Hong Jia-Wei and Hsiang-Tsung Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 326–333. ACM, 1981.

26 S Lennart Johnsson. Minimizing the communication time for matrix multiplication on multiprocessors. *Parallel Computing*, 19(11):1235–1257, 1993.

27 Richard R. Koch, F. T. Leighton, Bruce M. Maggs, Satish B. Rao, Arnold L. Rosenberg, and Eric J. Schwabe. Work-preserving Emulations of Fixed-connection Networks. *J. ACM*, 44(1):104–147, January 1997. doi:10.1145/256292.256299.

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

29 Lynn H Loomis and Hassler Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the American Mathematical Society*, 55(10):961–962, 1949.

30 Roy Nissim and Oded Schwartz. Revisiting the I/O-Complexity of Fast Matrix Multiplication with Recomputations. In *Proceedings of the 33rd IEEE International Parallel and Distributed Processing Symposium*, pages 714–716, 2019.

31 J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.

32 John E Savage. Extending the Hong-Kung model to memory hierarchies. In *International Computing and Combinatorics Conference*, pages 270–281. Springer, 1995.

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

34 Jacob N Scott. *An I/O-Complexity Lower Bound for All Recursive Matrix Multiplication Algorithms by Path-Routing*. PhD thesis, UC Berkeley, 2015.

35 Edgar Solomonik and James Demmel. Communication-optimal parallel 2.5 D matrix multiplication and LU factorization algorithms. In *European Conference on Parallel Processing*, pages 90–109. Springer, 2011.

36 Volker Streets. Gaussian elimination is not optimal. *numerical mathematics*, 13(4):354–356, 1969.

37 Y. D. Burago V. A. Zalgaller, A. B. Sossinsky. Geometric Inequalities. *The American Mathematical Monthly*, 96(6):544–546, 1989.

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