Matrix Multiplication I/O-Complexity by Path Routing

Jacob Scott
University of California at Berkeley
jnscott@math.berkeley.edu

Olga Holtz
University of California at Berkeley
holtz@math.berkeley.edu

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

ABSTRACT

We apply a novel technique based on path routings to obtain optimal I/O-complexity lower bounds for all Strassen-like fast matrix multiplication algorithms computed in serial or in parallel, assuming no reuse of nontrivial intermediate linear combinations. Given fast memory of size M, we prove an I/O-complexity lower bound of $\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \cdot M\right)$ for any Strassen-like matrix multiplication algorithm applied to $n \times n$ matrices of arithmetic complexity $\Theta(n^{\omega_0})$ with $\omega_0 < 3$ under this assumption. This generalizes an approach by Ballard, Demmel, Holtz, and Schwartz that provides a tight lower bound for Strassen's matrix multiplication algorithm but which does not apply to algorithms with disconnected encoding or decoding components of the underlying computation graph or algorithms with multiply copied values. We overcome these challenges via a new graph-theoretical approach for proving I/O-complexity lower bounds without the use of edge expansions.

Categories and Subject Descriptors

F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems; Computations on matrices

General Terms

Algorithms, Design, Performance

Keywords

Communication-avoiding algorithms; Fast matrix multiplication; I/O-complexity

SPAA'15, June 13–15, 2015, Portland, OR, USA.

ACM 978-1-4503-3588-1/15/06 ...$15.00.
http://dx.doi.org/10.1145/2755573.2755594 .

1. INTRODUCTION

In practice, most of the runtime of an algorithm is often due to the communication of data within memory hierarchy and between multiple processors, rather than the arithmetic computations. The amount of communication performed during an algorithm depends on the order in which intermediate values are computed and kept in/discarded from cache. While much work has gone into constructing implementations of algorithms that reduce communication, in this paper we show lower bounds on the communication of any implementation of a common class of fast (but not classical; see Lemma 1) matrix multiplication algorithms.

The I/O-complexity of an algorithm is defined as the minimum possible number of cache operations required to compute all outputs of the algorithm using a fixed cache size M. In 2011, Ballard, Demmel, Holtz, and Schwartz showed a tight lower bound on the I/O-complexity of Strassen's fast matrix multiplication algorithm [6]. We prove an analogous I/O-complexity bound via a more general technique for any fast square matrix multiplication algorithm based on a uniform recursive step that does not recompute any intermediate values, subject to the assumption that every intermediate linear combination is used in only one multiplication. We also claim, without proof, that this assumption can be lifted. Because algorithms achieving our I/O-complexity bounds have been found [3], our bounds are optimal.

Machine model

In this paper, we assume a 2-layer memory hierarchy for sequential computations consisting of slow memory and fast memory. The slow memory is of unlimited size and represents the hard drive of a computer, while the fast memory, called cache, is of limited size M and may represent RAM. We model the I/O communication of an algorithm as follows: initially, all data resides in slow memory and the cache is empty. A single value may be input into cache from slow memory or output to slow memory from cache for the cost of one I/O. A computation in the algorithm may only be performed if all input values to that computation already reside in cache; when computed, the result is also put in cache. The algorithm halts when all outputs of the algorithm are stored in slow memory. In this model we assume that no arithmetic computation is ever performed more than once. See [10] for the formalization of this model as a pebble game played on the computation graph.

The number of cache I/Os (henceforth simply called I/Os) required may depend on the order in which intermediate values of the algorithm are computed. The algorithm's *I/O-*

complexity is thus defined as the minimum number of I/Os over all sequences of computations and I/Os that computes the algorithm's outputs.

For parallel computations we consider P processors, each having independent local memory of size M. As in [6] and [16], we define the bandwidth cost of an algorithm executed in parallel to be the number of values communicated between processors along the critical path. In other words, we count the total number of words (single values) sent between processors, except that words sent between processors simultaneously count as only one I/O. We call this the *bandwidth cost* of the algorithm.

Previous Work

In 1981 Hong and Kung [10] proved a tight lower bound on the I/O-complexity of the classical $\Theta(n^3)$ matrix multiplication algorithm (achieved by blocked multiplication) using S-partitions. A different proof of this result was given in [12] and later generalized in [5] via the Loomis-Whitney inequality [13]; this approach was also shown to apply to several other problems in numerical linear algebra. See [1] and [9] for further generalizations using other geometric bounds. However, these proofs apply only to direct numerical linear algebra algorithms, but not to algorithms that use distributivity for cancellation, such as Strassen's algorithm.

The edge expansion approach detailed in [6] relates the I/O-complexity of an algorithm to the edge expansion properties of the underlying computation graph. This technique provides an I/O-complexity lower bound for Strassen's fast matrix multiplication algorithm, but fails for algorithms with base graphs (the computation graph representing one recursive step; see Section 3) containing disconnected encoding or decoding graphs and those involving multiple copying. In [4], this approach is extended to fast recursive matrix multiplication algorithms for rectangular matrices whose base graphs consist of multiple equal-size connected components. This is sufficient to yield lower bounds for some common fast matrix multiplication algorithms, such as Bini's algorithm [8] and the Hopcroft-Kerr algorithm [11], but still does not address algorithms with general base graphs.

In this paper we present the first approach for proving I/O-complexity lower bounds for recursive fast matrix multiplication algorithms involving arbitrary base graphs, as long as the same base graph is used at each recursive step.

2. NEW APPROACH

Most previous lower bounds in this field are based on the Loomis-Whitney inequality (as in [12]), dominator sets/S-partitions (as in [10], [14], and [7]), or edge expansions (as in [6] and [4]). In this paper we apply a new technique, based on the existence of a routing of paths within the underlying computation graph. In particular, we show the existence of a set of paths between all the inputs and all the outputs of sufficiently large matrix multiplication subcomputations such that each vertex is hit relatively few times. We then show that if some, but not all, of these input and output vertices are to be computed in one computation segment, then there must exist many other vertices that contribute cache I/Os as a result. This new approach may generalize to other problems that have sufficient symmetry to guarantee the existence of an efficient routing.

3. PRELIMINARIES

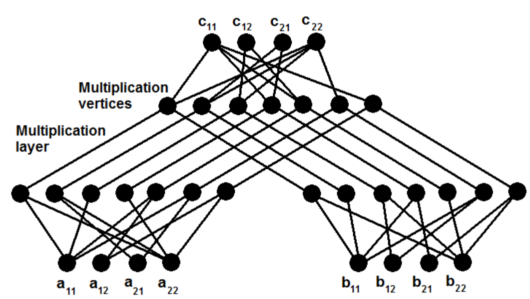
As in [6], we define the computation directed acyclic graph (CDAG) of an algorithm to be the directed graph that contains a vertex for every value in the computation (input, output, or intermediate value) and an edge whenever one value depends directly on another.

Strassen's matrix multiplication algorithm works as follows: to multiply 2×2 matrices A and B, compute specific linear combinations of the entries of A and linear combinations of the entries of B, perform 7 multiplications of these linear combinations, and then take linear combinations of the results to get the entries of $C = AB$. For larger square input matrices, divide each input matrix in half horizontally and vertically and apply the above procedure, recursively computing the necessary products of submatrices.

A *Strassen-like* algorithm is a square matrix multiplication algorithm that takes a similar form: to multiply matrices of dimensions $n_0 \times n_0$, take linear combinations of the input matrices, compute products, and take linear combinations of the results to yield the entries of the output matrix. For larger matrices, divide into blocks and recurse.

Let G_r be the CDAG of a Strassen-like algorithm for $n_0^r \times n_0^r$ square matrix multiplication $C = AB$, necessarily consisting of r recursive levels. We call G_1 the *base graph*. G_1 consists of two *encoding graphs*, which compute linear combinations of entries of A and of B, a *multiplication layer* with b *multiplication vertices*, which compute products of these linear combinations, and then a *decoding graph*, which takes linear combinations of these products to yield the entries of C. Note that G_1 has $2n_0^2$ inputs, n_0^2 from each input matrix. Further note that the same linear combination of input elements may be used as inputs in multiple product vertices. In this paper all figures show computations that proceed from bottom to top; we therefore omit the directions of edges. See Figure 1.

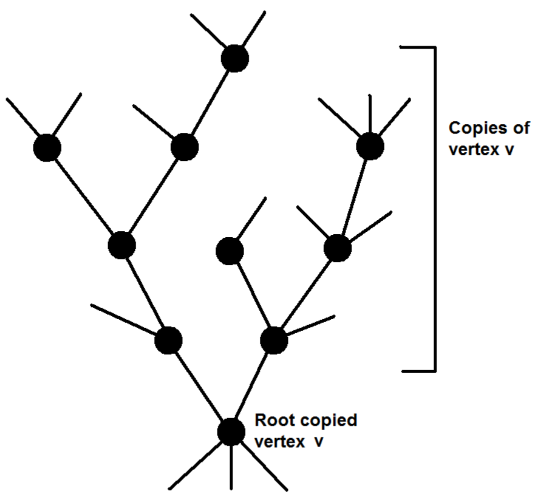
Figure 1: The base graph G_1 of Strassen's algorithm for multiplying two 2×2 matrices A and B. Here $b = 7$.



Note that G_r is a ranked graph, with inputs on rank 0 and outputs on rank $2r$. Ranks 0 through r lie in the encoding graphs and ranks $r+1$ through $2r$ lie in the decoding graph; the multiplication layer occurs between ranks r and $r+1$.

An intermediate vertex in G_r may have a single input vertex and, in this case, may have the same value as its one input. We call this *copying*; if the same value is copied to more than one child vertex, we call it *multiple copying*. We could consider this an artifact of our drawing of G_r and choose to identify these vertices. However, doing so would break the simple ranked, recursive structure of G_r. Instead, we group all vertices that represent the same value into a single *meta-vertex*. The vertices corresponding to each meta-vertex form a chain in the case of single copying and an upwards-branching subtree of the CDAG in the case of multiple copying, where each vertex of the subtree apart from the root has no other edges entering it from below. See Figure 2 for a depiction of a meta-vertex in the case of multiple copying. For most of this paper we consider only vertices, not meta-vertices, and then show that our technique still applies when copying or multiple copying occurs.

Figure 2: The meta-vertex corresponding to copies of the vertex v. Edges whose endpoints are not shown denote edges to vertices not in the shown meta-vertex. If this meta-vertex is in the CDAG for Strassen-like matrix multiplication, the structure of the meta-vertex is actually more regular than depicted due to the simple recursion.



The approach in [6] fails when the decoding graph of the base graph G_1 has a disconnected encoding or decoding graph. Note that the entire CDAG G_r (and similarly G_1) must be connected simply because it computes matrix multiplication (this will be shown in greater detail in the process of proving Lemma 4), but the decoding graph and/or encoding graph may not be connected individually.

In this paper, we first demonstrate a technique to derive the I/O-complexity bound for Strassen's algorithm presented in [6] more easily. We then show how to extend the technique, via use of Theorem 2, to the case of disconnected decoding and/or encoding graphs, allowing us to derive strong lower bounds for all Strassen-like matrix multiplication algorithms in which each linear combination is used in only one multiplication. This will prove Theorem 1, our main result. Finally, we present a proof of Theorem 2.

THEOREM 1 (MAIN THEOREM). *Let a and b be small constants. Consider a Strassen-like matrix multiplication algorithm for $n \times n$ matrices with arithmetic complexity $o(n^3)$ using cache size $M \leq o\left(n^2\right)$ in which the base graph has $2a$ inputs and b outputs. If in the base graph every non-trivial linear combination of elements of the input matrices is used in only one multiplication, then the algorithm has I/O-complexity*

$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{2\log_a b} \cdot M\right).$$

If run on P processors each of local cache size M, then the bandwidth cost is

$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{2\log_a b} \cdot \frac{M}{P}\right).$$

In other words, if a Strassen-like matrix multiplication algorithm performs $\Theta(n^{\omega_0})$ arithmetic operations with $\omega_0 < 3$, then its I/O-complexity is

$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \cdot M\right).$$

If run on P processors, the bandwidth cost is

$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \cdot \frac{M}{P}\right).$$

Furthermore, regardless of the cache size the bandwidth cost is

$$\Omega\left(\frac{n^2}{P^{2/\omega_0}}\right)$$

as long as computation is load balanced per rank of the computation graph.

In [3] an explicit algorithm is given that attains the bounds in Theorem 1. The sequential-to-parallel argument from [2] (as well as [6], [12], and [5]) allows us to take $P = 1$ – that is, work entirely in the serial model – and get the factor of $\frac{1}{P}$ in the parallel case with no additional work. Therefore, the remainder of this paper is devoted to proving a lower bound of $\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{2\log_a b} \cdot M\right)$ in the sequential case, from which Theorem 1 follows. By [3], the lower bounds in Theorem 1 are optimal.

4. DEFINITIONS

The proof presented in [6] relies on the notion of edge expansion; it shows a lower bound for the edge expansion of small subsets of vertices of the CDAG for Strassen's algorithm and then applies a lemma to yield a better edge expansion bound that relies on the fact that G_r contains as subgraphs many edge-disjoint copies of G_k for $k < r$. In our proof we bypass edge expansions entirely by explicitly cutting G_r into many copies of G_k for $k < r$. Both methods rely on the following fact, which is a consequence of the recursive definition of Strassen-like algorithms:

FACT 1. *For $0 \leq k \leq r$, let $G_{r,k}$ be the induced subgraph of G_r formed by the middle $2(k+1)$ levels of vertices (i.e. ranks $r-k$ through r of the encoding graphs and rank 0 through k of the decoding graph). Then $G_{r,k}$ consists of b^{r-k} vertex-disjoint copies of the graph G_k.*

In other words, the middle $2(k+1)$ layers of G_r are responsible for computing b^{r-k} independent matrix multiplications of square matrices of size $n_0^k \times n_0^k$.

Definition 1. *For any subset S of vertices of a computation graph G with directed edges E, define the following:*

1. $R(S) = \{v \in G - S \mid$ *for some* $w \in S,\ (v, w) \in E\}$
2. $W(S) = \{v \in S \mid$ *for some* $w \in G - S,\ (v, w) \in E\}$
3. $\delta(S) = R(S) \bigcup W(S)$

Note that $R(S)$ and $W(S)$ are disjoint, so $|\delta(S)| = |R(S)| + |W(S)|$. If S denotes a set of consecutively-computed vertices of G, then $R(S)$ denotes the set of vertices of G that must be read into cache, if not already present, during the computation of the vertices of S, and $W(S)$ the set of vertices of G that must be written to cache, if not to remain in cache after the computation of S. We assume that no vertex in G is ever computed more than once, meaning that if a vertex is used in the computations of multiple other vertices, it must either remain in cache until all the computations of vertices depending on it have finished or else be written to and read from cache.

We also assume that every linear combination of inputs in the base graph – except for the inputs themselves – is used in at most one multiplication in the base graph; this implies that every meta-vertex in the base graph is either a single vertex or else is rooted at one of the input vertices.

If S' is a subset of meta-vertices of G, we similarly define $\delta'(S') = \{$meta-vertex v' of G not in S' $|$ for some $w' \in S',\ v'$ and w' are adjacent$\}$, where two meta-vertices v' and w' are considered to be adjacent if for some vertex $v \in v'$ and vertex $w \in w'$, $(v, w) \in E$ or $(w, v) \in E$. In other words, $\delta'(S')$ is the set of meta-vertices adjacent to any of those in S'.

The main proof in this paper is based on finding routings of paths between sets of vertices in subgraphs of the CDAG that avoid using any vertex too many times. To this end we make the following definition:

Definition 2. *If X and Y are subsets of the vertices $V(G)$ of a directed graph G, define an m-*routing between X and Y *to be a collection R of $|X||Y|$ paths such that for any $x \in X$ and $y \in Y$ there exists a path, ignoring the directedness of edges, in G between x and y and such that every vertex of G is used collectively amongst all the paths in R at most m times. Similarly, if F is a subset of $V(G) \times V(G)$, define an m-*routing for F *to be a collection of paths, one for every $(v, w) \in F$, such that every vertex of G is hit at most m times.*

We will consider only the case where X and Y are disjoint. Note that m-routings need not be unique, and in fact part of the challenge of our proof is constructing a canonical m-routing with sufficiently small m.

Definition 3. *Let $G = (V, E)$ and $S \subseteq V$. If p is a path in G that contains at least one vertex in S and at least one vertex in $G - S$, then we call p* boundary-crossing *with respect to S in G.*

Note that any boundary-crossing path contains a pair of adjacent vertices such that one is in S and the other is not. Our basic strategy will be to show the existence of m-routings for relatively small m, and then show that such a routing must contain many boundary-crossing paths, implying the existence of many vertices in $\delta(S)$ and thus many meta-vertices in $\delta'(S')$.

5. SIMPLE PROOF FOR STRASSEN'S ALGORITHM

First we use our technique to rederive the lower bound on the I/O-complexity for Strassen's algorithm presented in [6], $\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \cdot M\right)$. As in [6], we consider the sequence of computations of vertices performed by the algorithm. In [6], this sequence is divided up into segments of sufficient length such that the I/O due to each segment is guaranteed to be at least M, the cache size. To do this, the smallest segment length s is found such that for any segment S of size s we are guaranteed that $|\delta(S)| \geq 3M$. All vertices present in $\delta(S)$ contribute to the I/Os due to S, except for vertices in $R(S)$ already present in cache (at most M) and vertices in $W(S)$ that need not be written to cache (at most M). Because [6] considers only the decoding graph of G_r, there are no concerns about vertex copying.

We use the same basic argument, but instead divide the sequence of vertex computations of the CDAG G_r into the smallest segments possible such that each segment S (except perhaps the last segment) contains $66M$ vertices from rank k of the decoding graph (rank $r+k$ of G_r) [1]. When a vertex v is in S we consider every vertex in the same meta-vertex as v to also be in S; however, because there is no copying in the decoding graph every meta-vertex can contain only one vertex from the decoding graph. Note that the size of each segment may be different; we care only about the number of vertices on this specific rank. We let $k = \lceil \log_4(132M) \rceil$, the smallest integer k such that $4^k \geq 2 \cdot 66M$. Because rank k of the decoding graph contains $4^k 7^{r-k}$ vertices, there are $\left\lfloor \frac{4^k 7^{r-k}}{66M} \right\rfloor$ such complete segments. Let S be one such complete segment and let $\bar{S}$ denote the vertices in S on rank k of the decoding graph of G_r. Thus we pick S as small as possible such that $|\bar{S}| = 66M$. If G_r is the CDAG for Strassen's algorithm for multiplying $n_0^r \times n_0^r$ matrices, recall that $G_{r,k}$ contains 7^{r-k} copies of the graph G_k. For $1 \leq i \leq 7^{r-k}$, let G_k^i be the ith such copy, S_i be the subset of S in G_k^i, and $\bar{S}_i$ be the subset of vertices of S_i on rank k of G_r.

Intuitively, we "count" S by the number of vertices of S on this particular rank. It is these vertices that will contribute, perhaps indirectly, to I/Os performed during the computation of S, regardless of what vertices on other ranks lie in S.

Let D_k be the decoding graph of G_k. We now claim that there exists a routing of paths between all the input vertices and output vertices of D_k such that no vertex of D_k is hit too often:

Claim 1. *There exists an $\left(11 \cdot 7^k\right)$-routing in D_k between the set of inputs of D_k and the set of outputs of D_k.*

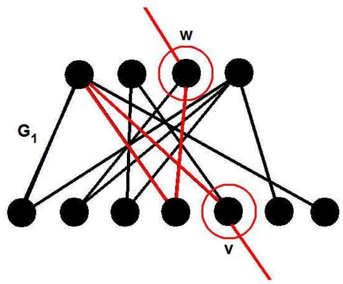
Proof. If D_1 were simply the complete graph $K_{7,4}$, there would exist a very natural routing of paths between inputs and outputs of D_k: for any input and output, there is a unique chain of vertices between them defined by the sequence of subcomputations the input lies in. A vertex on rank i of D_k is then hit $7^i 4^{k-i} \leq 7^k$ times in this routing, once for every pair of input vertex beneath it and output vertex above it.

Unfortunately, D_1 is not a complete graph. However, because D_1 is connected there still exists a path within each copy of D_1 from any input vertex to any output vertex.

[1] We did not optimize for the constant factor.

Where each path previously went directly from an input vertex v to an output vertex w of each D_1, it will now take any path (that doesn't repeat vertices) through the same D_1 component from v to w. This idea is depicted in Figure 3. This multiplies the number of times a vertex is hit in the routing by at most the number of vertices in D_1, 11. $\square$

Figure 3: One of the encoding graphs in G_1 for Strassen's algorithm. Because there is no edge from v to w, a chain must instead take a more indirect path, shown in red, through the encoding graph.



Each G_k^i contains a copy of D_k – for this proof we consider only the decoding piece D_k of G_k, but in the full proof we must consider G_k in its entirety in order to account for base graphs with disconnected encoding/decoding portions. Let D_k^i be the copy of D_k lying in G_k^i and note that $|\bar{S}_i| \leq \frac{1}{2}4^k$, so at most half of the vertices on the top rank of D_k^i are in S. For each $1 \leq i \leq 7^{r-k}$, fix an $\left(11 \cdot 7^k\right)$-routing in D_k^i between the 7^k inputs and 4^k outputs. See Figure 4. There are now two cases:

1. Fewer than half of the 7^k vertices on the bottom rank of D_k^i are in S. In this case, there exist at least $|\bar{S}_i|\frac{1}{2}7^k$ paths in the routing going from an input to D_k^i not in S to an output in S.

2. At least half of the vertices on the bottom rank of D_k^i are in S. In this case, there exist at least $\left(4^k - |\bar{S}_i|\right)\frac{1}{2}7^k$ paths in the routing going from an input in S to an output not in S.

In either case, there are at least $\frac{1}{2}|\bar{S}_i|7^k$ boundary-crossing (between S_i and $D_k^i - S_i$) paths in the routing. Associate to each boundary-crossing path an edge in the path that crosses between S_i and $D_k^i - S_i$. The vertex of this edge that is not in S lies in $\delta(S_i)$. By the definition of m-routing,

$$|\delta(S_i)| \geq \frac{\frac{1}{2}|\bar{S}_i|7^k}{11 \cdot 7^k} = \frac{1}{22}|\bar{S}_i|$$

Adding this up over all the $\bar{S}_i$ yields

$$|\delta(S)| \geq \sum_{i=1}^{7^{r-k}} \frac{1}{22}|\bar{S}_i| = \frac{1}{22}|\bar{S}| \qquad (1)$$

This step relies on the D_k being disjoint and the lack of copying in the decoding graph of Strassen's (or any Strassen-like) matrix multiplication algorithm. If multiple copying did occur, vertices in the different D_k^i need not correspond to distinct computations. This will add an additional layer of complexity to the upcoming proof.

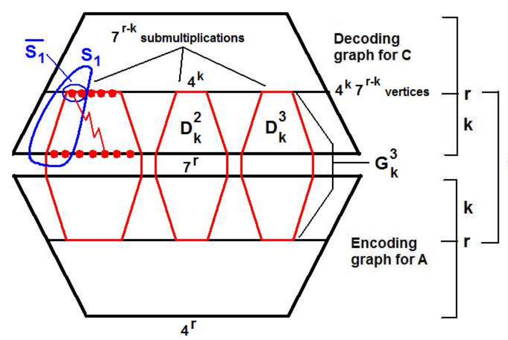
Since $|\bar{S}|$ was chosen to be $66M$, this yields $\delta(S) \geq 3M$. Therefore the computation of S contributes at least M I/Os. Thus the total I/O is at least

$$\left\lfloor \frac{4^k 7^{r-k}}{66M} \right\rfloor \cdot M = \Omega\left(7^r \left(\frac{4}{7}\right)^k\right) = \Omega\left(|V(G_r)|\frac{M}{M^{\log_4 7}}\right)$$

$$= \Omega\left(\left(\frac{n_0}{\sqrt{M}}\right)^{\log_2 7} \cdot M\right)$$

as long as $M \leq o\left(n_0^2\right)$ (which guarantees that $66M \leq 4^k 7^{r-k}$). $\square$

Figure 4: An example of a path considered in the $\left(11 \cdot 7^k\right)$-routing between an input vertex (to D_k^1) that is not in S and an output vertex that is in S. The submultiplications are shown in red, S_1 is shown in blue, and $\bar{S}_1$ is circled. Note that the path zags up and down, as explained in Figure 3. For simplicity, only one encoding graph is shown and only 3 submultiplications are drawn.



6. STRASSEN-LIKE ALGORITHMS

We now turn our attention to Strassen-like square matrix multiplication algorithms. Several nuances prevent our above proof from working as-is:

1. G_1 may have disconnected encoding or decoding graphs. This prevents us from finding an m-routing in the decoding graph D_k because D_k itself may no longer be connected. We will solve this problem by considering G_k, consisting of the decoding graph as well as the two encoding graphs. The paths in our m-routing will no longer be chains or even chains with length 1 "zags," but may need to bounce between inputs and outputs of G_k several times. See Figure 5.

2. Multiple copying may occur in the encoding graphs. This means a collection of m-routings for the G_k^i could potentially hit a meta-vertex more than m times. We will show via Theorem 2 that m-routings will only hit a meta-vertex entirely within G_k at most m times and then change the overall counting argument slightly to prevent meta-vertices between multiple G_k^is from being hit too often.

As before, we divide the sequence of vertex computations of G_r into segments such that each segment S contains enough vertices of a certain type. Again let G_k^i be the ith subcomputation of $G_{r,k}$ for $1 \le i \le b^{r-k}$. Let a *duplicated* vertex be a vertex of the CDAG G_r with at least one other copy (called a *duplicate*) in G_r, that is one whose meta-vertex contains more than one vertex. We call two subcomputations *input-disjoint* if none of their inputs lie in the same meta-vertex.

Let S be a segment of the sequence of vertex computations. Recall that when $v \in S$ we consider every vertex w in the same meta-vertex as v to also be in S. For this argument we count only the vertices on rank k of the decoding graph of G_r and rank $r-k$ of either encoding graph that are in mutually input-disjoint subcomputations $G_{r,k}$. We choose $k = \lceil \log_a 72M \rceil$, the smallest integer k such that $a^k \ge 2 \cdot 36M$.

First we show that counting only vertices lying in subcomputations that do not share inputs reduces the number of vertices on the relevant ranks by only a constant factor.

LEMMA 1. *Let $k \le r-2$. If not every vertex in the encoding graph for A of G_1 is a duplicated vertex and similarly for the encoding graph for B of G_1, then a fraction $\frac{1}{b^2}$ of the subcomputations G_k^i are mutually input-disjoint.*

PROOF. Consider the recursion tree of subcomputations computed by G_r. Let P_1 be the "grandparent" subcomputation of G_k^i – the subcomputation in the recursion tree two levels above G_k^i – and suppose P_1 multiplies matrices A_1 by B_1. Then at least one child subcomputation P_2 of P_1 multiplies matrices A_2 by B_2 such that A_2 shares no meta-vertices with A_1. Similarly, at least one child subcomputation of P_2 multiplies matrices A_3 by B_3 such that B_3 shares no meta-vertices with B_2, and hence with B_1. Thus at least one subsubcomputation of P_1 is input-disjoint from it. P_1 has b^2 subcomputations two levels down from it, so at least a fraction $\frac{1}{b^2}$ of all the subcomputations G_k^i are mutually input-disjoint. □

Fix a collection C of b^{r-k-2} mutually input-disjoint subcomputations G_k^i. Let $\bar{S}$ be the set of vertices of S on the aforementioned ranks in these subcomputations. Formally, for $v \in S$ we let $v \in \bar{S}$ if both conditions below are met:

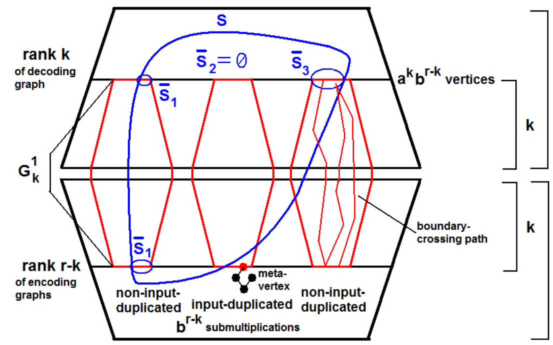
1. v lies on one of the following ranks: rank k of the decoding graph of G_r, rank $r-k$ of the encoding graph of G_r that encodes A, or rank $r-k$ of the encoding graph of G_r that encodes B.

2. The subcomputation G_k^i that v lies in (necessarily as an input or output of) is in C.

Divide the sequence of vertex computations into the smallest segments such that for each segment S we have $|\bar{S}| \ge 36M$. Let S_i be the subset of S in G_k^i and $\bar{S}_i$ be the subset of $\bar{S}$ in G_k^i. Note that if G_k^i is not one of the chosen input-disjoint subcomputations then $\bar{S}_i = \emptyset$. Intuitively, only the vertices in $\bar{S}$ "count" towards our I/O lower bound, regardless of how many other vertices lie in S, and we choose our segment divisions such that each segment has enough counted vertices. See Figure 5.

Note that if the condition of Lemma 1 is not met, then the algorithm never computes linear combinations of one of the input matrices. It is well known that any matrix

Figure 5: The overall idea of the main proof. For simplicity only one encoding graph is explicitly drawn. The set S is shown in blue. Note that only the elements on rank k of the decoding graph and rank $r-k$ of the encoding graphs in input-disjoint G_k^is lie in $\bar{S}$. A typical boundary-crossing path in G_k^3 is shown. (Not shown) The two vertices of the path on the bottom rank of G_k^3 lie in different encoding graphs.



multiplication algorithm that computes linear combinations of only one of the input matrices performs no better than naive matrix multiplication and so does not have $o(n^3)$ arithmetic complexity (i.e., is not a fast matrix multiplication algorithm). Thus from now on we assume the condition of Lemma 1 is met.

Second, we show that our choice of partitioning the sequence of vertex computations into segments S exists. If meta-vertices contained multiple input and/or output vertices counted in $\bar{S}$, then including into S the next vertex v in the sequence of vertex computations – which by definition also includes into S every vertex in the same meta-vertex as v – could increase this count by more than one.

LEMMA 2. *If G_k^i and G_k^j are input-disjoint, then the meta-vertices corresponding to the inputs and outputs of G_k^i and G_k^j are all distinct.*

PROOF. Note that the decoding graph of G_1 cannot contain copying. If it did, then in the base case of $n_0 \times n_0$ matrix multiplication $C_1 = A_1 B_1$ some outputs would be identically equal, which is not the case. Hence the decoding graph of G_r contains no copying, and so every output vertex of G_k^i and G_k^j is non-duplicated. By definition, the input vertices of G_k^i and G_k^j are in distinct meta-vertices, proving the lemma. □

For the remainder of this proof we will consider, for each i, the entire subcomputation graph G_k^i (as opposed to just the decoding portion D_k^i). We must consider the decoding graph and both encoding graphs of G_k^i together because the decoding graph by itself, or even the decoding graph plus one encoding graph, may be disconnected. We now state the main theorem used in our proof, whose proof we defer until Section 7. Compare to the routing found in Section 5 between the input and output vertices of each D_k^i.

Theorem 2 (Routing Theorem). *Let G_k be the CDAG for $n_0^k \times n_0^k$ matrix multiplication, $a = n_0^2$, and let the encoding graph of the base graph G_1 have $2a$ inputs and b outputs. Then there exists a $6a^k$-routing between the set of inputs of G_k and the set of outputs of G_k. Furthermore, every meta-vertex in G_k is also hit by the routing at most $6a^k$ times.*

For each of the mutually input-disjoint G_k^i in C, fix a $6a^k$-routing guaranteed by the Routing Theorem between the inputs and outputs of G_k^i. Because the size of the top rank of G_k^i is a^k and the size of the bottom rank is $2a^k$ and $|\bar{S}_i| \le |\bar{S}| \le \frac{1}{2}a^k$, for every vertex v in $\bar{S}_i$ there exist at least $\frac{1}{2}a^k$ paths in the routing that go either:

1. between a vertex in S on the bottom rank of G_k^i and a vertex not in S on the top rank of G_k^i (if v is on the bottom rank)
2. between a vertex not in S on the bottom rank of G_k^i and a vertex in S on the top rank of G_k^i (if v is on the top rank).

Thus the routing in G_k^i contains at least $\frac{1}{2}a^k|\bar{S}_i|$ boundary-crossing paths; call the set of such paths P_i and let $P = \bigcup_i P_i$ be all these boundary-crossing paths in the above routings for all input-disjoint G_k^i. Then $|P| \ge \sum_i \frac{1}{2}a^k|\bar{S}_i| = \frac{1}{2}a^k|\bar{S}|$. By the Routing Theorem every meta-vertex contained entirely within G_k^i is hit by the routing at most $6a^k$ times. No meta-vertex in G_k^i extends beneath the bottom rank of G_k^i, and so every meta-vertex in G_r intersects at most one of the mutually input-disjoint G_k^i. Therefore every meta-vertex in G_r is hit at most $6a^k$ times by the paths in P.

Let S' be the set of meta-vertices represented by S, and recall that $\delta'(S')$ denotes all meta-vertices adjacent to S' that are not in S' itself. Then

$$|\delta'(S')| \ge \frac{\frac{1}{2}a^k|\bar{S}|}{6a^k} = \frac{1}{12}|\bar{S}| \tag{2}$$

This is a more general analogue of Equation 1.

Every meta-vertex adjacent to S necessarily contributes one to the I/Os due to computing S, except possibly for those meta-vertices already in memory (at most M) and those that need not be written to cache (at most M). Because $|\bar{S}| = 36M$, we have $|\delta'(S')| \ge 3M$, and so computing S requires at least M I/Os.

As indicated above, because G_r has $o(n^3)$ multiplications we may apply Lemma 1. Because rank k of the decoding graph of G_r and rank $r-k$ of the encoding graphs of G_r together have size $3a^kb^{r-k}$ and $\frac{1}{b^2}$ of these vertices are in mutually input-disjoint subcomputations G_k^i, the total I/O from computing G_r is at least

$$\left\lfloor \frac{\frac{1}{b^2}3a^kb^{r-k}}{36M} \right\rfloor \cdot M = \Omega\left(b^r\left(\frac{a}{b}\right)^k\right) = \Omega\left(|V(G_r)|\frac{M}{M^{\log_a b}}\right)$$

$$= \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{2\log_a b} \cdot M\right)$$

as long as $M \le o(n^2)$ (which guarantees that $36M \le \frac{1}{b^2}3a^kb^{r-k}$ and $k \le r-2$).

In the parallel case, we apply the above argument to a processor that computes an above-average number of vertices of $\bar{S}$, yielding a factor of $\frac{1}{P}$ as in [2]. The cache-independent result comes from instead picking $k = \Theta\left(\log_b \frac{n^{\omega_0}}{P}\right)$ and letting S represent the computations performed by just one processor. This proves Theorem 1. □

7. PROOF OF THE ROUTING THEOREM

In this section we prove Theorem 2. Let G_k be the CDAG for a square Strassen-like matrix multiplication algorithm for $C = AB$, let Out be the set of outputs of G_k (corresponding to entries of C), In be the set of inputs, In_A be the set of inputs to the encoding graph for A within G_k, and In_B be the inputs to the encoding graph for B. Then $|Out| = |In_A| = |In_B| = a^k = n_0^{2k}$. For $v \in In$ and $w \in Out$, we say that the input-output pair (v, w) is a *guaranteed dependence* if in any correct matrix multiplication algorithm there exists a chain from v to w, or equivalently if the output element corresponding to w explicitly depends on the input element corresponding to v. It is clear that if $v \in In_A$ represents the input a_{ij} and w represents the output $c_{i'j'}$ then there is a guaranteed dependence between v and w if and only if $i = i'$, and similarly if $v \in In_B$ represents the input b_{ij}, then there is a guaranteed dependence between v and w if and only if $j = j'$.

To prove the Routing Theorem we will combine the following two lemmas, whose proofs follow in the succeeding sections:

Lemma 3. *Let $F \subseteq V(G_k) \times V(G_k)$ be the set of all guaranteed dependencies (v, w) of G_k with $v \in In$ and $w \in Out$. Then there exists a $2n_0^k$-routing for F in G_k consisting only of chains.*

Intuitively, we can route chains between all pairs of input and output vertices where a chain is guaranteed to exist while using no vertex more than $2\sqrt{a^k}$ times. That every path of the routing is a chain is not necessary to complete the proof of the Routing Theorem.

Lemma 4. *Fix a routing for F, where F is as defined in Lemma 3. Then there exists a routing between In and Out such that every path in the routing consists of the concatenation of chains in F – some reversed in direction – such that each chain in F is used $3n_0^k$ times.*

In other words, given any way of routing chains between all guaranteed dependencies, we can combine those chains, backwards and forwards, to give a path between every input and every output vertex while not using any such chain more than $3\sqrt{a^k}$ times.

Given these lemmas, the proof is simple:

Proof of the Routing Theorem. By Lemma 3, fix a $2n_0^k$-routing R_0 for the set of guaranteed dependencies F. By Lemma 4, there exists a routing R between the inputs and outputs of G_r composed of concatenations of chains (some reversed) in R_0 such that every chain in R_0 is used at most $3n_0^k$ times. Thus in the routing R every vertex of G is used at most $2n_0^k \cdot 3n_0^k = 6a^k$ times, and so R is a $6a^k$-routing, as desired.

Because every meta-vertex is an upward-facing subtree (see Figure 2), any path hitting a meta-vertex also hits the root vertex of the meta-vertex. Hence every meta-vertex is also hit at most $6a^k$ times. □

7.1 Proof of Lemma 4

In this section we prove the second, significantly easier, lemma. The proof of this lemma is constructive, yielding an explicit scheme for routing chains between all inputs and outputs given a routing for all guaranteed dependencies. This lemma holds for any correct matrix multiplication algorithm based only on the definition of matrix multiplication.

Proof of Lemma 4. For an input vertex v of G_k and output vertex w corresponding to element $c_{i'j'}$ of C, suppose first that $v \in In_A$. Let v then represent element a_{ij} of A. We form the following sequence of guaranteed dependencies:

$$a_{ij} \to c_{ij'} \to b_{jj'} \to c_{i'j'}$$

That is, $(a_{ij}, c_{ij'})$ is a guaranteed dependence, $(b_{jj'}, c_{ij'})$ is a guaranteed dependence, and $(b_{jj'}, c_{i'j'})$ is a guaranteed dependence. Note that every guaranteed dependence in this chain involves 3 out of the 4 variables i, i', j, and j'. Hence as i, j, i', and j' vary between 1 and n_0^k, each guaranteed dependence above is used n_0^k times, once for each value of the missing variable (for each time it appears in the above sequence). For example, for any i, j, and j', the guaranteed dependence between a_{ij} and $c_{ij'}$ is used exactly once for every $1 \le i' \le n_0^k$. See Figure 6 for another interpretation of this pattern.

Similarly, if $v \in In_B$ let v correspond to element b_{ij} of B. The following sequence of guaranteed dependencies has the same properties:

$$b_{ij} \to c_{i'j} \to a_{i'i} \to c_{i'j'}$$

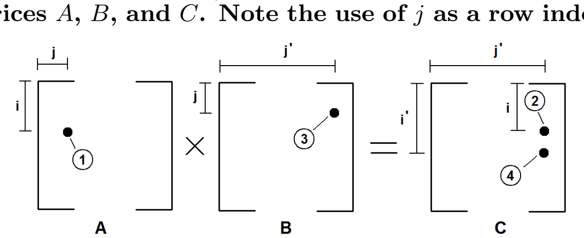
Amongst both these sequences, each guaranteed dependence between an element of A and one of C is used exactly $3 \cdot n_0^k$ times and similarly for every guaranteed dependence between B and C. This proves Lemma 4. □

Note that these sequences are not unique. When routing a_{ij} to $c_{i'j'}$, any sequence of the form

$$a_{ij} \to c_{ij'} \to b_{\_j'} \to c_{i'j'}$$

where the blank is any value forms a set of sequences of guaranteed dependencies. However, unless the values that the blank takes are well-distributed over j for all choices of i, i', and j', this sequence will not have the desired property. This explains the odd use of j as a row index, and similarly the use of i as a column index when routing b_{ij} to $c_{i'j'}$.

Figure 6: The sequence of guaranteed dependencies between a_{ij} and $c_{i'j'}$ shown as elements in the matrices A, B, and C. Note the use of j as a row index.



7.2 Proof of Lemma 3

This lemma is significantly harder to prove. We use the following overall strategy: In order to prove there exists a $2n_0^k$-routing between all guaranteed dependencies, we show there exists a n_0-routing of guaranteed dependencies in the subgraph of G_1 formed by the decoding graph together with the encoding graph for A; by the recursive structure of G_k, this is sufficient to prove it in general. Define a *middle-rank* vertex of G_1 to be a vertex on the top rank of the encoding graph of A. To show the lemma for this $\frac{2}{3}$ of G_1, we show a (several-to-one) matching between guaranteed dependencies and middle-rank vertices on some chain satisfying the dependence. By assumption, every vertex representing a linear combination of elements of A is adjacent to exactly one multiplication vertex; thus a routing of guaranteed dependencies that uses each middle-rank vertex at most n_0 times also uses each multiplication vertex at most n_0 times.

We will prove the existence of this matching via a version of Hall's Matching Theorem. In order to apply this theorem, we will need to show that for every set of d guaranteed dependencies, there exist chains between those dependencies collectively hitting at least $\frac{d}{n_0}$ middle-rank vertices. We demonstrate that if this is not the case, then setting some entries of the $n_0 \times n_0$ input matrix A to be identically 0 results in an algorithm that correctly computes many of the guaranteed dependencies between C and A using relatively few multiplications. Finally, we show that this implies the existence of an algorithm for multiplying a $n_0 \times n_0$ matrix by a length n_0 vector in fewer than n_0^2 operations, which is known to be impossible [15]. This will conclude the proof.

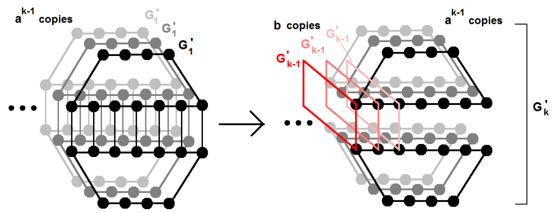
Let G'_k be the induced subgraph of G_k containing the vertices from the decoding graph of G_k and the encoding graph of G_k for A (excluding only the encoding graph for B). Let F' be the subset of F with both vertices lying in G'_k, that is the set of guaranteed dependencies (v, w) between inputs v of A and outputs w of C. For simplicity, we simply call F' the *guaranteed dependencies of* G'_k. We now consider m-routings for the set of guaranteed dependencies (that is, F') of G'_k. It then suffices to find an a^k-routing of guaranteed dependencies in G'_k.

Claim 2. *If there exists an m-routing for the guaranteed dependencies of G'_1, then there exists an m^k-routing for the guaranteed dependencies of G'_k.*

Proof. This lemma follows from the recursive structure of G'_k. Intuitively, the graph G'_k is formed by placing b copies of G'_{k-1} in parallel, connecting up their inputs with a^{k-1} copies of the encoding graph for A, and connecting up their outputs with a^{k-1} copies of the decoding graph for C. See Figure 7. In other words, take a^{k-1} copies of G'_1 and replace their middle two ranks with copies of G'_{k-1}. Any number of copies of G'_1 in parallel still have an m-routing for guaranteed dependencies, and replacing their middle ranks effectively replaces a pair of adjacent vertices on the middle ranks with a guaranteed dependence in G'_{k-1}. Thus if there exists an m^{k-1}-routing for G_{k-1} then there exists an m^k routing for G_k. The claim then follows by induction. □

Therefore it will suffice to prove the existence of an n_0-routing for the guaranteed dependencies of G'_1. We now apply a version of Hall's Matching Theorem:

Figure 7: The construction of G'_k from b copies of G'_{k-1}. A pair of adjacent vertices on the middle two ranks is replaced with a guaranteed dependence in one of the G'_{k-1}.

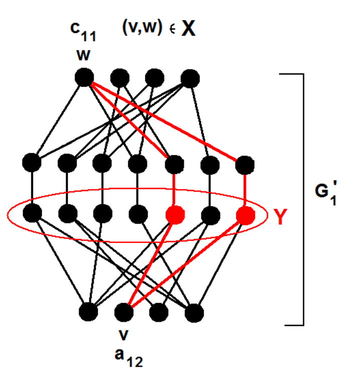


Theorem 3 (Hall's Matching Theorem). *(Many-to-one version) Let $G = (X, Y)$ be a bipartite graph and for $D \subseteq V(G)$ let $N(D)$ denote the set of neighbors of D in G. If for every $D \subseteq X$ we have $|N(D)| \geq \frac{|D|}{p}$, then there exists a many-to-one matching between X and Y such that every vertex in X is used exactly once and every vertex in Y is used at most p times.*

This theorem follows from the standard form of Hall's Matching Theorem by simply duplicating all vertices in Y p times.

We now construct a graph $H = (X, Y)$ to which to apply Theorem 3. For every guaranteed dependence (v, w) in G'_1 (with v an input representing an element of A and w an output representing an element of C), define a corresponding vertex in X. Let Y be the set of middle-rank vertices of G_1: all vertices on the top rank of the encoding graph for A. It suffices to assign to each guaranteed dependence in X a middle-rank vertex from Y through which its chain may pass. To this end, if $x \in X$ corresponds to the guaranteed dependence (v, w) and $y \in Y$ corresponds to the middle-rank vertex t, let there be an edge between x and y if there exists some chain between v and w passing through t. See Figure 8.

Figure 8: The vertices shown in red are those adjacent to the vertex in H corresponding to the guaranteed dependence (v, w), where v corresponds to the input a_{12} of A and w corresponds to the output c_{11} of C. The graph shown is the G'_1 for Strassen's algorithm.



Lemma 5. *For any set $D \subseteq X$, we have $|N(D)| \geq \frac{|D|}{n_0}$.*

From Lemma 5, the proof of Lemma 3 follows, and thus our main result:

Proof of Lemma 3. By Hall's Matching Theorem (Theorem 3), there exists a many-to-one matching from X to Y using every vertex in Y at most n_0 times. Fix such a matching. For every guaranteed dependence (v, w) of G'_1, simply route a chain through the vertex of Y that (v, w) is matched with. Every vertex on the middle two ranks of G'_1 is thus hit at most n_0 times. Every vertex on the top and bottom ranks of G'_1 is hit exactly n_0 times by any routing for guaranteed dependencies that uses only chains, because in $n_0 \times n_0$ matrix multiplication every element of A influences n_0 elements of C, and every element of C depends on n_0 elements of A. Thus there exists a n_0-routing for the guaranteed dependencies in G'_1, and so by Claim 2 there exists a n_0^k-routing for the guaranteed dependencies of G'_k. The same holds for the induced subgraph of G_1 consisting of the decoding graph together with the encoding graph for B, yielding a $2n_0^k$-routing for the guaranteed dependencies of G_k. □

7.3 Proof of Lemma 5

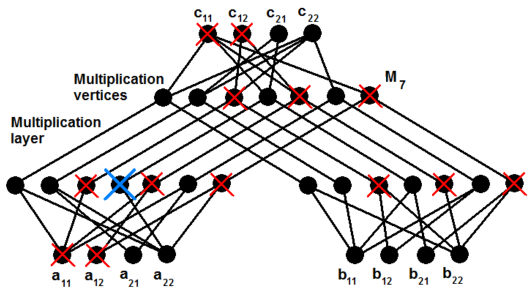
Finally, we prove Lemma 5 to complete the proof of the Routing Theorem and thus our main result, Theorem 1:

Proof of Lemma 5. Suppose by way of contradiction that for some subset $D \subseteq X$ of guaranteed dependencies in G'_1 we have $|N(D)| < \frac{|D|}{n_0}$. Recall that a guaranteed dependence occurs between the vertex representing a_{ij} and the vertex representing $c_{i'j'}$ exactly when $i = i'$. We may thus partition D by the choice of i: let D_i be the subset of D consisting of guaranteed dependencies between a_{ij} and $c_{ij'}$ for some j and j'. Because $1 \leq i \leq n_0$, for some i we have $|D_i| \geq \frac{|D|}{n_0}$. Since $N(D_i) \subseteq N(D)$, we have $|N(D_i)| < \frac{|D|}{n_0} \leq |D_i|$. In other words, the set of guaranteed dependencies D_i is computed using fewer than $|D_i|$ multiplication vertices.

We now demonstrate that this is impossible by using this structure to create a matrix-vector multiplication algorithm that requires fewer than n_0^2 multiplications. For fixed D_i, define the computation graph G_1° as follows: G_1° is the induced subgraph of G_1 containing as inputs vertices corresponding to all the elements of B and their linear combinations, the elements a_{ij} of A for all j, and the elements $c_{ij'}$ of C for all j'. G_i° additionally contains all the middle-rank vertices in $N(D_i)$ and all vertices on the bottom rank of the decoding graph. G_1° may now contain "useless" vertices – we draw G_1° with these vertices additionally removed, but it does not matter for the bounds in this proof.

By the structure of G_1, every multiplication vertex multiplies a linear combination $\sum_{i,j} \lambda_{ij}^A a_{ij}$ by a linear combination $\sum_{i,j} \lambda_{ij}^B b_{ij}$ for some coefficients λ_{ij}^A and λ_{ij}^B in the ground field F ($\mathbb{R}$ or $\mathbb{C}$). We consider linear combinations of the a_{ij}s with coefficients in $F[b_{11}, b_{12}, \ldots, b_{n_0 n_0}]$. In other words, consider b_{ij}s to be coefficients and a_{ij}s to be variables. For $1 \leq j \leq n_0$, let a_{ij} and c_{ij} be the inputs and outputs of G_1° respectively. Note that for all $1 \leq j, j' \leq n_0$, c_{ij} depends on $a_{ij'}$. We now define a boolean-valued function f that represents whether the coefficient of each input is correct in each output: For $1 \leq j, j' \leq n_0$, define $f(j, j')$ to be 1

Figure 9: G_1° for Strassen's algorithm when $i = 2$ and $D_2 = \{(a_{21}, c_{21}), (a_{21}, c_{22}), (a_{22}, c_{22})\}$. The crossed-out vertices are those removed from G_1 to construct this reduced computation graph G_1°. Because the guaranteed dependence (a_{22}, c_{21}) is not included in D_2, the vertex crossed out in blue is removed, and so G_1° does not quite compute vector-matrix multiplication; the coefficient of a_{22} in the computation of c_{21} may not be correct.



exactly when the coefficient of $a_{ij'}$ in c_{ij} is its correct value for matrix multiplication, namely $b_{j'j}$, and otherwise 0.

Let n_f denote the number of pairs (j, j') with $1 \leq j, j' \leq n_0$ at which f takes the value 1 – that is, the number of coefficients correctly set by G_1°. By the definition of G_1° relative to the matching graph H, we have $n_f \geq |D_i|$: If the guaranteed dependence of c_{ij} on $a_{ij'}$ is represented in D_i, then the coefficient of $a_{ij'}$ in c_{ij} must be "correct" for matrix multiplication, since, by definition of G_1°, there exists no chain between the vertices corresponding to c_{ij} and $a_{ij'}$ contained in G_1 (which correctly computes matrix multiplication) but not in G_1°.

Finally, we use G_1° to construct a new, correct, vector-matrix multiplication algorithm. Define $\bar{G}_1^\circ$ to be the CDAG formed as follows: to the CDAG G_1° add $n_0^2 - n_f$ multiplication vertices, one for each pair (j, j') for which $f(j, j') = 0$. For $1 \leq j, j' \leq n_0$ let the coefficient of $a_{ij'}$ in c_{ij} computed by G_1° be $x_{j'j} \in F[b_{11}, b_{12}, \ldots, b_{n_0 n_0}]$ – a linear combination of the "coefficients" b_{ij}. For each such j and j' at which $f(j, j') = 0$, use a multiplication vertex to compute $a_{ij'}(b_{j'j} - x_{j'j})$ and add it to the output vertex representing c_{ij}. In other words, for every incorrect dependence of c_{ij} on $a_{ij'}$ we may use a single multiplication vertex to "fix" the dependence. Now $\bar{G}_1^\circ$ correctly computes $n_0 \times n_0$ vector-matrix multiplication. G_1° contained fewer than $|D_i|$ multiplication vertices and we added at most $n_0^2 - n_f$, so $\bar{G}_1^\circ$ has $< |D_i| + n_0^2 - n_f \leq |D_i| + n_0^2 - |D_i| = n_0^2$ multiplication vertices. Thus we have constructed a correct algorithm for computing $n_0 \times n_0$ vector-matrix multiplication using fewer than n_0^2 multiplications, which is known to be impossible [15]. This concludes the proof of Lemma 5 and hence of our main result Theorem 1. □

We state the result we obtained in the proof of Lemma 5 as its own Lemma:

LEMMA 6. *Let G_1° be a CDAG with inputs a_{ij} and b_{ij} and outputs c_{ij} for $1 \leq i, j \leq n_0$ where each c_{ij} is computed as a product of linear combinations of the a_{ij} and b_{ij}. If for d pairs (j, j'), $1 \leq j, j' \leq n_0$, the coefficient of $a_{ij'}$ in c_{ij} is $b_{j'j}$, then G_1° uses at least d multiplications.*

8. CONCLUSION

We have proven optimal lower bounds for the I/O-complexity of any Strassen-like square matrix multiplication algorithm in which every linear combination in the base graph is used in only one multiplication by proving the existence of a routing between the inputs and outputs of such an algorithm that uses every intermediate computation vertex relatively few times. The proof generalizes easily to algorithms composed of different base graphs, as long as each base graph performs square matrix multiplication with $2a$ inputs and b subcomputations and satisfies the conditions of Lemma 1. This bound holds regardless of the form of the base graph(s), including those that have disconnected encoding or decoding pieces and those that perform multiple copying. Our technique provides a novel alternative to the edge expansion argument in [6] that applies to less straightforward recursive computation graphs.

We believe that the assumption that every linear combination is used in only one multiplication can also be lifted. Without this assumption Lemma 5 no longer holds; vertices representing linear combinations used in multiple multiplications may require too many paths routed through them. Thus a more general approach to routing guaranteed dependencies is required. This difficulty can be overcome by routing paths in response to the choice of S, where paths are now allowed to "jump" to other vertices on the same rank of G_k that have the same membership in S. We believe it can be shown that this optimization does not decrease the number of boundary-crossing edges and still results in every vertex lying on at most $6a^k$ "generalized" paths, thus extending our result to all fast Strassen-like matrix multiplication algorithms.

9. ACKNOWLEDGMENTS

Research is supported by grants 1878/14, and 1901/14 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities) and grant 3-10891 from the Ministry of Science and Technology, Israel. Research is also supported by the Einstein Foundation and the Minerva Foundation.

10. REFERENCES

[1] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz. Communication lower bounds and optimal algorithms for numerical linear algebra. *Acta Numerica*, 23:1–155, 5 2014.

[2] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proc. 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, SPAA '12, pages 77–79, New York, NY, USA, 2012. ACM.

[3] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2012*, 2012.

[4] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Graph expansion analysis for communication costs of fast rectangular matrix

multiplication. *Design and Analysis of Algorithms*, 7659:13–36, 2012.

[5] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. & Appl.*, 32(3):866–901, 2011.

[6] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM*, 59(6), 2012.

[7] G. Bilardi, A. Pietracaprina, and P. D'Alberto. On the space and access complexity of computation dags. *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science, London, UK*, pages 47–58, 2000.

[8] D. Bini, M. Capovani, F. Romani, and G. Lotti. $o(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication. *Information processing letters*, 8(5):234–235, 1979.

[9] M. Christ, J. Demmel, N. Knight, T. Scanlon, and K. A. Yelick. Communication lower bounds and optimal algorithms for programs that reference arrays - part 1. Technical Report UCB/EECS-2013-61, EECS Department, University of California, Berkeley, May 2013.

[10] J. W. Hong and H. T. Kung. The red-blue pebble game. *STOC 1981: Proceedings of the thirteenth annual ACM symposium on theory of computing*, pages 326–333, 1981.

[11] J. Hopcroft and L. Kerr. On minimizing the number of multiplications necessary for matrix multiplication. *SIAM Journal on Applied Mathematics*, 20(1):30–36, 1971.

[12] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, 2004.

[13] L. H. Loomis and H. Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the American Mathematical Society*, 55(10), 1949.

[14] J. Savage. Space-time tradeoffs in memory hierarchies. *Technical report, Brown University, Providence, RI, USA*, 1994.

[15] S. Winograd. On the number of multiplications required to compute certain functions. *Proceedings of the National Academy of Science*, 58(5), 1967.

[16] C.-Q. Yang and B. Miller. Critical path analysis for the execution of parallel and distributed programs. *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 366–373, 1988.