Graph Expansion and Communication Costs of Fast Matrix Multiplication

Grey Ballard * James Demmel † Olga Holtz ‡ Oded Schwartz §

ABSTRACT

The communication cost of algorithms (also known as I/O-complexity) is shown to be closely related to the expansion properties of the corresponding computation graphs. We demonstrate this on Strassen's and other fast matrix multiplication algorithms, and obtain the first lower bounds on their communication costs. For sequential algorithms these bounds are attainable and so optimal.

1. INTRODUCTION

The communication of an algorithm (e.g., transferring data between the CPU and memory devices, or between parallel processors, a.k.a. I/O-complexity) often costs significantly more time than its arithmetic[1]. It is therefore of interest to obtain lower bounds for the communication needed on the one hand, and to design and implement algorithms minimizing communication[2] and attaining these lower bounds on the other hand.

While Moore's Law predicts an exponential speedup of hardware in general, the annual improvement rate of time-per-arithmetic-operation has, over the years, consistently exceeded that of time-per-word read/write [GSP04]. The fraction of running time spent on communication is thus expected to increase further.

Communication model.

We model communication costs of sequential and parallel architecture as follows. In the sequential case, with two levels of memory hierarchy (fast and slow), communication means reading data items (*words*) from slow memory (of unbounded size), to fast memory (of size M) and writing data from fast memory to slow memory[3]. Words that are stored contiguously in slow memory can be read or written in a bundle which we will call a *message*. We assume that a message of n words can be communicated between fast and slow memory in time $\alpha + \beta n$ where α is the *latency* (seconds per message) and β is the *inverse bandwidth* (seconds per word). We define the *bandwidth cost* of an algorithm to be the total number of words communicated and the *latency cost* of an algorithm to be the total number of messages communicated. We assume that the input matrices initially resides in slow memory, and is too large to fit in the smaller fast memory. Our goal then is to minimize bandwidth and latency costs.[4]

In the parallel case, we assume p processors, each with memory of size M. We are interested in the communication among the processors. As in the sequential case, we assume that a message of n consecutively stored words can

*Computer Science Department, University of California, Berkeley, CA 94720. Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). (ballard@eecs.berkeley.edu).

†Mathematics Department and CS Division, University of California, Berkeley, CA 94720. This material is based on work supported by U.S. Department of Energy grants under Grant Numbers DE-SC0003959, DE-SC0004938, and DE-FC02-06-ER25786, as well as Lawrence Berkeley National Laboratory Contract DE-AC02-05CH11231. (demmel@cs.berkeley.edu).

‡Departments of Mathematics, University of California, Berkeley and Technische Universität Berlin. Research supported by the Sofja Kovalevskaja programme of Alexander von Humboldt Foundation and by the National Science Foundation under agreement DMS-0635607, while visiting the Institute for Advanced Study. (holtz@math.berkeley.edu).

§Computer Science Department, University of California, Berkeley, CA 94720. Part of this research was performed while at The Weizmann Institute of Science, and while at Technische Universität Berlin. Research supported by U.S. Department of Energy grants under Grant Numbers DE-SC0003959, by ERC Starting Grant Number 239985, and by the Sofja Kovalevskaja programme of Alexander von Humboldt Foundation. (odedsc@eecs.berkeley.edu).

[1]Communication time varies by orders of magnitude, from 0.5×10^{-9} second for L1 cache reference, to 10^{-2} second for disk access. The variation is even more dramatic when communication occurs over networks or the internet [GSP04].

[2]Communication requires much more energy than arithmetic, and saving energy may be even more important than saving time.

[3]See [BDHS10a] for definition of a model with memory hierarchy, and a reduction from the two levels model. All bounds in this paper thus apply to the model with memory hierarchy as well.

[4]The sequential communication model used here is sometimes called the *two-level I/O model* or *disk access machine (DAM)* model (see [AV88, BBF+07, CR06]). Our bandwidth cost model follows that of [HK81] and [ITT04] in that it assumes the block-transfer size is one word of data ($B = 1$ in the common notation). However, our model allows message sizes to vary from one word up to the maximum number of words that can fit in fast memory.

be communicated in time $\alpha + \beta n$. This cost includes the time required to "pack" non-contiguous words into a single message, if necessary. We assume that the input is initially evenly distributed among all processors, so $M \cdot p$ is at least as large as the input. Again, the bandwidth cost and latency cost are the words and messages counts respectively. However, we count the number of words and messages communicated along the critical path as defined in [YM88] (i.e., two words that are communicated simultaneously are counted only once), as this metric is closely related to the total running time of the algorithm. As before, our goal is to minimize the number of words and messages communicated.

We assume that (1) the cost per flop is the same on each processor and the communication costs (α and β) are the same between each pair of processors, (2) all communication is "blocking": a processor can send/receive a single message at a time, and cannot communicate and compute a flop simultaneously (the latter assumption can be dropped, affecting the running time by a factor of two at most), and (3) there is no communication resource contention among processors. For example, if processor 0 sends a message of size n to processor 1 at time 0, and processor 2 sends a message of size n to processor 3 also at time 0, the cost along the critical path is $\alpha + \beta n$. However, if both processor 0 and processor 1 try to send a message to processor 2 at the same time, the cost along the critical path will be the sum of the costs of each message.

The Computation Graph and Implementations of an Algorithm.

The computation performed by an algorithm on a given input can be modeled as a computation directed acyclic graph *(CDAG)* : We have a vertex for each input / intermediate / output argument, and edges according to direct dependencies (e.g., for the binary arithmetic operation $x := y + z$ we have a directed edge from v_y to v_x and from v_z to v_x, where the vertices v_x, v_y, v_z stand for the arguments x, y, z, respectively).

An implementation of an algorithm determines, in the parallel model, which arithmetic operations are performed by which of the p processors. This corresponds to partitioning the corresponding CDAG into p parts. Edges crossing between the various parts correspond to arguments that are in the possession of one processor, but are needed by another processor, therefore relate to communication. In the sequential model, an implementation determines the order of the arithmetic operations, in a way that respects the partial ordering of the CDAG (see Section 3 relating this to communication cost).

Implementations of an algorithm may greatly vary in their communication costs. The *I/O-complexity of an algorithm* is the minimum bandwidth cost of the algorithm, over all possible implementations. The I/O-complexity of a problem is defined to be the minimum I/O-complexity of all algorithms for this problem.

There are quite a few I/O-complexity lower and upper bounds of specific algorithms (see below). These are results of the form: any implementation of algorithm *Alg* requires at least X communication (or: there is an implementation for algorithm *Alg* that requires at most X communication). However, we are not aware of I/O-complexity lower bounds for a *problem*, i.e., of the form: any algorithm for a problem P requires at least X communication. The lower bounds in this paper are for all the *implementations* for a family of algorithms: Strassen-like[5] fast matrix multiplication.

Previous Work.

Consider the classical $\Theta(n^3)$ algorithm for matrix multiplication. While naïve implementations are communication inefficient, communication-minimizing sequential and parallel variants of this algorithm were constructed, and proved optimal, by matching lower bounds [Can69, HK81, FLPR99, ITT04].

In [BDHS10a, BDHS10b] we generalize the results of [HK81, ITT04] regarding matrix multiplication, to obtain new I/O-complexity lower bounds for a much wider variety of algorithms. Most of our bounds are shown to be tight. This includes algorithms for LU factorization, Cholesky factorization, LDL^T factorization, QR factorization, as well as algorithms for eigenvalues and singular values. Thus we essentially cover all direct methods of linear algebra. The results hold for dense matrix algorithms (most of them have $O(n^3)$ complexity), as well as sparse matrix algorithms (whose running time depends on the number of non-zero elements, and their locations). They apply to sequential and parallel algorithms, to compositions of linear algebra operations (like computing the powers of a matrix), and to certain graph theoretic problems[6].

In [BDHS10a, BDHS10b] we use the approach of [ITT04], based on the Loomis-Whitney geometric theorem [LW49, BZ88], by embedding segments of the computation process into a three dimensional cube. This approach, however, is not suitable when distributivity is used, as is the case in Strassen [Str69] and other fast matrix multiplication algorithms (e.g., [CW90, CKSU05]).

While the I/O-complexity of classic matrix multiplication and algorithms with similar structure is quite well understood, this is not the case for algorithms of more complex structure. Avoiding the communication of parallel classical matrix multiplication was addressed [Can69] almost simultaneously with the publication of Strassen's fast matrix multiplication [Str69]. Moreover, an I/O-complexity lower bound for the classical matrix multiplication algorithm is known for almost three decades [HK81]. Nevertheless, the I/O-complexity of Strassen's fast matrix multiplication and similar algorithms has not yet been resolved.

In this paper we obtain the first communication cost lower bound for Strassen's and other fast matrix multiplication algorithm, in the sequential and parallel models. For sequential algorithms these bounds are attainable and so optimal.

Communication Costs of Fast Matrix Multiplication

Upper bound.

The I/O-complexity $IO(n)$ of Strassen's algorithm (see Algorithm 1, Appendix B), applied to n-by-n matrices on a machine with fast memory of size M, can be bounded above as follows (for actual uses of Strassen's algorithm, see [DHSS94, HLJJ+96, DS04]): Run the recursion until the matrices are sufficiently small. Then, read the two input sub-matrices

[5] See Section 5 for definition.

[6] See [MPP02] for bounds on graph-related problems, and our [BDHS10b] for a detailed list of previously known and recently designed sequential and parallel algorithms that attain the above mentioned lower bounds.

into the fast memory, perform the matrix multiplication inside the fast memory, and write the result into the slow memory[7]. We thus have $IO(n) \leq 7 \cdot IO\left(\frac{n}{2}\right) + O(n^2)$ and $IO\left(\frac{\sqrt{M}}{3}\right) = O(M)$. Thus

$$IO(n) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\lg 7} \cdot M\right). \quad (1)$$

Lower bound.

In this paper, we obtain a tight lower bound:

THEOREM 1. (MAIN THEOREM) *The I/O-complexity $IO(n)$ of Strassen's algorithm on a machine with fast memory of size M, assuming that no arithmetic operation is computed twice[8], is*

$$IO(n) = \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\lg 7} \cdot M\right). \quad (2)$$

It holds for any implementation and any known variant of Strassen's algorithm[9,10]. This includes Winograd's $O(n^{\lg 7})$ variant that uses 15 additions instead of 18, which is the most used fast matrix multiplication algorithm in practice [DHSS94, HLJJ+96, DS04].

For parallel algorithms, using a reduction from the sequential to the parallel model (see e.g., [ITT04] or our [BDHS10b]) this yields:

COROLLARY 2. *Let $IO(n)$ be the I/O-complexity of Strassen's algorithm, run on a machine with p processors, each with a local memory of size M. Assume that no arithmetic operation is computed twice. Then*

$$IO(n) = \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\lg 7} \cdot \frac{M}{p}\right).$$

Note that although recomputation is forbidden here, replication of the input is allowed. Specifically, for multiplying matrices of maximum possible size (i.e., $M = \Theta\left(\frac{n^2}{p}\right)$, a "2D algorithm") we have $IO(n) = \Omega\left(n^2/p^{2-\frac{\lg 7}{2}}\right)$.

We can extend the bounds to a wider class of all Strassen-like fast matrix multiplication algorithms. Let *Alg* be any Strassen-like matrix multiplication algorithm that runs in time $O(n^{\omega_0})$ for some $2 < \omega_0 < 3$. Then, using the same arguments that lead to (1), the I/O-complexity of *Alg* can be shown to be $IO(n) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \cdot M\right)$. We obtain a matching lower bound:

THEOREM 3. *The I/O-complexity $IO(n)$ of a recursive Strassen-like fast matrix multiplication algorithm with $O(n^{\omega_0})$ arithmetic operations, on a machine with fast memory of size M is*

$$IO(n) = \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \cdot M\right). \quad (3)$$

COROLLARY 4. *Let $IO(n)$ be the I/O-complexity of a Strassen-like algorithm (with arithmetic performed as in Theorem 3), run on a machine with p processors, each with a local memory of size M. Assume that no arithmetic operation is computed twice. Then*

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

For the "2D" case $M = \Theta\left(\frac{n^2}{p}\right)$ we have $IO(n) = \Omega\left(\frac{n^2}{p^{2-\frac{\omega_0}{2}}}\right)$. Corollaries 2 and 4 can be generalized to other models, such as the heterogenous model (where processors have different memory sizes and communication and computation speeds), and shared memory model. The reduction is achieved by observing the communication of a single processor.

The Expansion Approach

The proof of the main theorem is based on estimating the edge expansion of the computation directed acyclic graph *(CDAG)* of an algorithm. The I/O-complexity is shown to be closely connected to the edge expansion properties of this graph. As the graph has a recursive structure, the expansion can be analyzed directly (combinatorially, similarly to what is done in [Mih89, ASS08, KKK10]) or by spectral analysis (in the spirit of what was done for the Zig-Zag expanders [RVW02]). There is, however, a new technical challenge. The replacement product and the Zig-Zag product act similarly on all vertices. This is not what happens in our case: multiplication and addition vertices behave differently.

The expansion approach is similar to the one taken by Hong and Kung [HK81]. They use the red-blue pebble game to obtain tight lower-bounds on the I/O-complexity of many algorithms, including ordinary matrix multiplication, matrix-vector multiplication, and FFT. The proof is obtained by showing that the size of any subset of the vertices of the CDAG is bounded by a function of the size of its dominator set.

On the one hand, their dominator set technique has the advantage of allowing recomputation of any intermediate value. We were not able to allow recomputation using our edge expansion approach. On the other hand, the dominator set requires large input or output. Such an assumption is not needed by the edge expansion approach, as the bounds are guaranteed by edge expansion of many (internal) parts of the CDAG. In that regard, one can view the approach of [ITT04] (also in [BDHS10a, BDHS10b]) as an edge expansion assertion on the CDAGs of the corresponding classical algorithms.

The study of expansion properties of a CDAG was also suggested as one of the main motivations of Lev and Valiant [LV83] in their work on superconcentrators. They point out many papers proving that classes of algorithms computing DFT, matrix inversion and other problems all have to have good expansion properties, thus providing lower-bounds on the number of the arithmetic operations required.

Other papers study connections between bounded-space computation, and combinatorial expansion-related proper-

[7] Here we assume that the recursion tree is traversed in the usual depth-first order.

[8] We assume no recomputation throughout the paper.

[9] This lower bound for the sequential case seems to contradict the upper bound from FOCS'99 [FLPR99, BCG+08]), due to a miscalculation [Lei08].

[10] To obtain the lower bounds for latency costs we divide the bandwidth costs by the maximal message length, M. This holds for all the lower bounds here, both in the sequential and parallel models.

ties of the corresponding CDAG (see e.g., [Sav94, BP99, BPD00] and their references).

Paper organization.

Section 2 contains preliminaries on the notions of graph expansion. In Section 3 we state and prove the connection between I/O-complexity and the expansion properties of the computation graph. In Section 4 we analyze the expansion of the CDAG of Strassen's algorithm. We present the generalization of the bounds to other fast matrix multiplication algorithms, conclusions and open problems in Section 5.

2. PRELIMINARIES

Edge expansion.

The edge expansion $h(G)$ of a d-regular undirected graph $G = (V, E)$ is:

$$h(G) \equiv \min_{U \subseteq V, |U| \leq |V|/2} \frac{|E(U, V \setminus U)|}{d \cdot |U|} \tag{4}$$

where $E(A, B) \equiv E_G(A, B)$ is the set of edges connecting the vertex sets A and B. We omit the subscript G when the context makes it clear.

Expansion of small sets.

For many graphs, small sets of vertices have better expansion guarantee. Let $h_s(G)$ denote the edge expansion guarantee for sets of size at most s in G:

$$h_s(G) \equiv \min_{U \subseteq V, |U| \leq s} \frac{|E(U, V \setminus U)|}{d \cdot |U|} \ . \tag{5}$$

In many cases, $h_s(G)$ does not depend on $|V(G)|$, although it may decrease when s increases. One way of bounding $h_s(G)$ is by decomposing G into small subgraphs of large edge expansion.

CLAIM 5. *Let $G = (V, E)$ be a d-regular graph that can be decomposed into edge-disjoint (but not necessarily vertex disjoint) copies of a d'-regular graph $G' = (V', E')$. Then the edge expansion guarantee of G for sets of size at most $|V'|/2$ is $h(G') \cdot \frac{d'}{d}$, namely*

$$h_{\frac{|V'|}{2}}(G) \equiv \min_{U \subseteq V, |U| \leq |V'|/2} \frac{|E_G(U, V \setminus U)|}{d \cdot |U|} \geq h(G') \cdot \frac{d'}{d} \ .$$

See proof in Appendix A.

When G is not regular.

If $G = (V, E)$ is not regular but has a bounded maximal degree d, then we can add ($< d$) loops to vertices of degree $< d$, obtaining a regular graph G'[11]. Note that for any $S \in V$, we have $|E_G(S, V \setminus S)| = |E_{G'}(S, V \setminus S)|$, as none of the added edges (loops) contributes to the edge expansion of G'.

3. I/O-COMPLEXITY AND EDGE EXPANSION

In this section we recall the computation graph of an algorithm, then show how a partition argument connects the expansion properties of the graph and the I/O-complexity of the algorithm. A similar partition argument already appeared in [ITT04], and then in our [BDHS10b]. In both cases it is used to relate I/O-complexity to the Loomis-Whitney geometric bound [LW49], which can be viewed, in this context, as an expansion guarantee for the corresponding graphs.

The computation graph.

For a given algorithm, we consider the computation (directed) graph $G = (V, E)$, where there is a vertex for each arithmetic operation *(AO)* performed, and for every input element. G contains a directed edge (u, v), if the output operand of the AO corresponding to u (or the input element corresponding to u), is an input operand to the AO corresponding to v. The in-degree of any vertex of G is, therefore, at most 2 (as the arithmetic operations are binary). The out-degree is, in general, unbounded[12], i.e., it may be a function of $|V|$. We next show how an expansion analysis of this graph can be used to obtain the I/O-complexity lower bound for the corresponding algorithm.

The partition argument.

Let M be the size of the fast memory. Let O be any total ordering of the vertices that respects the partial ordering of the CDAG G, i.e., all the edges are going from left to right. This ordering can be thought of as the actual order in which the computations are performed. Let P be any partition of V into segments $S_1, S_2, ...$, so that a segment $S_i \in P$ is a subset of the vertices that are contiguous in the total ordering O.

Let R_S and W_S be the set of read and write operands, respectively (see Figure 1). Namely, R_S is the set of vertices outside S that have an edge going into S, and W_S is the set of vertices in S that have an edge going outside of S. Then the total I/O-complexity due to reads of AOs in S is at least $|R_S| - M$, as at most M of the needed $|R_S|$ operands are already in fast memory when the execution of the segment's AOs starts. Similarly, S causes at least $|W_S| - M$ actual write operations, as at most M of the operands needed by other segments are left in the fast memory when the execution of the segment's AOs ends. The I/O-complexity is therefore bounded below by[13]

$$IO \quad \geq \quad \max_{P} \sum_{S \in P} (|R_S| + |W_S| - 2M) \ . \tag{6}$$

Edge expansion and I/O-complexity.

Consider a segment S and its read and write operands R_S and W_S (see Figure 1). If the graph G containing S has $h(G)$ edge expansion[14], maximum degree d and at least $2|S|$

[11] Here we use the convention that a loop adds 1 to the degree of a vertex.

[12] As the lower bounds are derived for the bounded out-degree case, we will show how to convert the corresponding CDAG to obtain constant out-degree, without affecting the I/O-complexity too much.

[13] One can think of this as a game: the first player orders the vertices. The second player partitions them into contiguous segments. The objective of the first player (e.g., a good programmer) is to order the vertices so that any consecutive partitioning by the second player leads to a small communication count.

[14] The direction of the edges does not matter much for the expansion-bandwidth argument: treating all edges as undi-

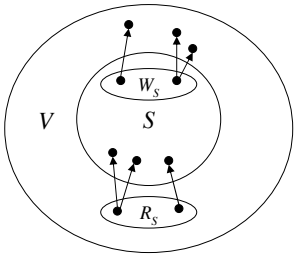

Figure 1: A subset (segment) S and its corresponding read operands R_S, and write operands W_S.

vertices, then (using the definition of $h(G)$), we have

CLAIM 6. $|R_S| + |W_S| \geq \frac{1}{2} \cdot h(G) \cdot |S|$.

PROOF. We have $|E(S, V \setminus S)| \geq h(G) \cdot d \cdot |S|$. Either (at least) half of the edges $E(S, V \setminus S)$ touch R_S or half of them touch W_S. As every vertex is of degree d, we have $|R_S| + |W_S| \geq \max\{|R_S|, |W_S|\} \geq \frac{1}{d} \cdot \frac{1}{2} \cdot |E(S, V \setminus S)| \geq h(G) \cdot |S|/2$. □

Combining this with (6) and choosing to partition V into $|V|/s$ segments of equal size s, we obtain: $IO \geq \max_s \frac{|V|}{s} \cdot \left(\frac{h(G)\cdot s}{2} - 2M\right) = \Omega\left(|V| \cdot h(G)\right)$. In many cases $h(G)$ is too small to attain the desired I/O-complexity lower bound. Typically, $h(G)$ is a decreasing function in $|V(G)|$, namely the edge expansion deteriorates with the increase of the input size and with the running time of the corresponding algorithm. This is the case with matrix multiplication algorithms: the cubic, as well as the Strassen and Strassen-like algorithms. In such cases, it is better to consider the expansion of G on small sets only: $IO \geq \max_s \frac{|V|}{s} \cdot \left(\frac{h_s(G)\cdot s}{2} - 2M\right)$. Choosing[15] the minimal s so that

$$\frac{h_s(G) \cdot s}{2} \geq 3M \tag{7}$$

we obtain

$$IO \geq \frac{|V|}{s} \cdot M \ . \tag{8}$$

In some cases, the computation graph G does not fit this analysis: it may not be regular (with vertices of unbounded degree), or its edge expansion may be hard to analyze. In such cases, we may consider some subgraph G' of G instead to obtain a lower bound on the I/O-complexity:

CLAIM 7. *Let $G = (V, E)$ be a computation graph of an algorithm Alg. Let $G' = (V', E')$ be a subgraph of G, i.e., $V' \subseteq V$ and $E' \subseteq E$. If G' is d regular and $\alpha = \frac{|V'|}{|V|}$, then the I/O-complexity of Alg is*

$$IO \geq \frac{\alpha}{2} \cdot \frac{|V|}{s} \cdot M \tag{9}$$

where s is chosen so that $\frac{h_s(G')\cdot \alpha s}{2} \geq 3M$.

The correctness of this claim follows from Equations (7) and (8), and from the fact that at least an $\alpha/2$ fraction of the segments have at least $\frac{\alpha}{2} \cdot s$ of their vertices in G' (otherwise $V' < \frac{\alpha}{2} \cdot V/s \cdot s + (1 - \frac{\alpha}{2}) \cdot V/s \cdot \frac{\alpha}{2} s < \alpha V$). We therefore have:

LEMMA 8. *Let Alg be an algorithm with $AO(N)$ arithmetic operations (N being the total input size, $N = \Theta(n^2)$ for matrix multiplication) and computation graph $G(N) = (V, E)$. Let $G'(N) = (V', E')$ be a regular constant degree subgraph of G, with $\frac{|V'|}{|V|} = \Theta(1)$. Then the I/O-complexity of Alg[16] on a machine with fast memory of size M is*

$$IO = \Omega\left(|V'| \cdot h_s(G'(N))\right) \quad \text{for } s = AO(M) \ . \tag{10}$$

As $AO(N) = \Theta(|V'|)$ and $h_s(G'(N))$ for $s = AO(M)$ is $\Theta(h(G'(M)))$ (recall Claim 5) we obtain, equivalently,

$$IO = \Omega\left(AO(N) \cdot h(G'(M))\right) \ . \tag{11}$$

4. EXPANSION PROPERTIES OF STRASSEN'S ALGORITHM

Recall Strassen's algorithm for matrix multiplication (see Algorithm 1 in Appendix B) and consider its computation graph (see Figure 2). Let $H_{\lg n}$ be the computation graph of Strassen's algorithm on input matrices of size $n \times n$. $H_{\lg n}$ has the following structure: encode A: generate weighted sums of elements of A. Similarly encode B. Then multiply the encodings of A and B element-wise. Finally, decode C, by taking weighted sums of the products. This is the structure of all the fast matrix multiplication algorithms that were obtained since Strassen's[17].

Assume w.l.o.g. that n is an integer power of 2. Denote by $Enc_{\lg n}A$ the part of $H_{\lg n}$ that corresponds to the encoding of matrix A. Similarly, $Enc_{\lg n}B$, and $Dec_{\lg n}C$ correspond to the parts of $H_{\lg n}$ that compute the encoding of B and the decoding of C, respectively.

Duplicate Dec_1C 7^i times. Duplicate Dec_iC four times. We next identify the $4 \cdot 7^i$ output vertices of the copies of Dec_1C with the $4 \cdot 7^i$ input vertices of the copies of Dec_iC. Recall that each Dec_1C has four output vertices. The first output vertex of the 7^i Dec_1C graphs are identified with the 7^i input vertices of the first copy of Dec_iC. The second output vertex of the 7^i Dec_1C graphs are identified with the 7^i input vertices of the second copy of Dec_iC. And so on. We make sure that the jth input vertex of a copy of Dec_iC is identified with an output vertex of the jth copy of Dec_1C.

We similarly obtain $Enc_{i+1}A$ from Enc_iA and Enc_1A (and $Enc_{i+1}B$ from Enc_iB and Enc_1B). For every i, H_i is obtained by connecting edges from the jth output vertices of Enc_iA and Enc_iB to the jth input vertex of Dec_iC.

rected, changes the I/O-complexity estimate by a factor of 2 at most. For simplicity, we will treat G as undirected.

[15]The existence of an s that satisfies the condition is not always guaranteed. In the next section we confirm this for Strassen, for sufficiently large $|V(G)|$ (in particular, $|V(G)|$ has to be larger than M). Indeed this is the interesting case, as otherwise all computations can be performed inside the fast memory, with no communication, except for reading the input once.

[16]In Strassen's algorithm, $N = 2n^2$ is the number of input matrices elements and $T(N) = \Theta\left(n^{\omega_0}\right) = \Theta\left(N^{\omega_0/2}\right)$. G' is the graph Dec_kC for $k = \lg M$, see Section 4 for the definition of Dec_kC.

[17]Indeed, any fast matrix multiplication algorithm can be converted into one of this form [Raz03].

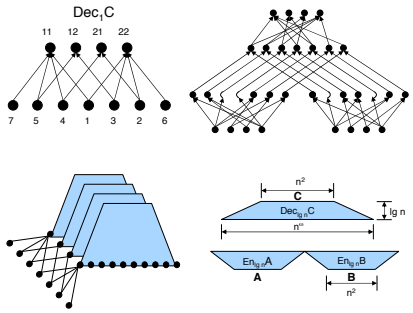

Figure 2: The computation graph of Strassen's algorithm (See Algorithm 1 in Appendix). Top left: Dec_1C. Top right: H_1. Bottom left: $Dec_{\lg n}C$. Bottom right: $H_{\lg n}$. Vertices drawn with in-degrees larger than 2 indicate a (weighted) summation. A vertex v with l incoming edges represents a full binary tree (not necessarily balanced) with root v and l leaves.

The graph Dec_1C has no vertices which are both input and output, therefore:

FACT 9. *$Dec_{\lg n}C$ is of maximal degree 6.*

However, Enc_1A and Enc_1B have vertices which are both input and output (e.g., A_{11}), therefore $Enc_{\lg n}A$ and $Enc_{\lg n}B$ have vertices of out-degree $\Theta(\lg n)$. All in-degrees are at most 2, as an arithmetic operation has at most two inputs.

As $H_{\lg n}$ contains vertices of large degrees, it is easier to consider $Dec_{\lg n}C$: it contains only vertices of constant bounded degree, yet at least one third of the vertices of $H_{\lg n}$ are in it.

LEMMA 10. (MAIN LEMMA) *The edge expansion of Dec_kC is*

$$h(Dec_kC) = \Omega\left(\left(\frac{4}{7}\right)^k\right)$$

See proof in Section 4.

Assume w.l.o.g. that n is an integer power of $\sqrt{M}$.[18] Then $Dec_{\lg n}C$ can be split into edge-disjoint copies of $Dec_{\frac{1}{2}\lg M}C$. Using Claim 5, we thus deduce the expansion of $Dec_{\lg n}C$ on small sets:

COROLLARY 11. *$s \cdot h_s(Dec_{\lg n}C) \geq 3M$ for $s = 9 \cdot M^{\lg 7/2}$.*

As $Dec_{\lg n}C$ contains $\alpha = \frac{1}{3}$ of the vertices of $H_{\lg n}$, Lemma 8 now yields Main Theorem 1. Note that $Dec_{\lg n}C$ has no input vertices, so no restriction on input replication is needed.

Dec_1C is presented, for simplicity, with vertices of in-degree larger than two (but constant). A vertex of degree larger than two, in fact, represents a full binary tree. Note that replacing these high in-degree vertices with trees changes the edge expansion of the graph by a constant factor at most (as this graph is of constant size, and connected). Moreover, as there is no change in the number of input and output vertices, the arguments in the following proof of Lemma 10 still hold.

[18] We may assume this, as we are dealing with a lower bound here, so it suffices to prove the assertion for an infinite number of n's. Alternatively, in the following decomposition argument, we leave out a few of the top or bottom levels of vertices of $Dec_{\lg n}C$, so that n is an integer power of $\sqrt{M}$ and so that at most $|S|/2$ vertices of S are cut off.

Combinatorial Estimation of the Expansion.

PROOF OF LEMMA 10. Let $G_k = (V, E)$ be Dec_kC, and let $S \subseteq V, |S| \leq |V|/2$. We next show that $|E(S, V \setminus S)| \geq c \cdot d \cdot |S| \cdot \left(\frac{4}{7}\right)^k$, where c is some universal constant, and d is the constant degree of Dec_kC (after adding loops to make it regular).

The proof works as follows. Recall that G_k is a layered graph, so all edges (excluding loops) connect between consecutive levels of vertices. We argue (in Claim 15) that each level of G_k contains about the same fraction of S vertices, or else we have many edges leaving S. We also observe (in Fact 16) that such homogeneity (of a fraction of S vertices) does not hold between distinct parts of the lowest level, or, again, we have many edges leaving S. We then show that the homogeneity between levels combined with the heterogeneity of the lowest level, guarantees that there are many edges leaving S.

Let l_i be the ith level of vertices of G_k, so $4^k = |l_1| < |l_2| < \cdots < |l_i| = 4^{k-i+1}7^{i-1} < \cdots < |l_{k+1}| = 7^k$. Let $S_i \equiv S \cap l_i$. Let $\sigma = \frac{|S|}{|V|}$ be the fractional size of S and $\sigma_i = \frac{|S_i|}{|l_i|}$ be the fractional size of S in level i. Due to averaging, we observe the following:

FACT 12. *There exist i and i' such that $\sigma_i \leq \sigma \leq \sigma_{i'}$.*

From the geometric sum, we now have:

FACT 13.

$$\begin{aligned} |V| &= \sum_{i=1}^{k+1} |l_i| = \sum_{i=1}^{k+1} |l_{k+1}| \cdot \left(\frac{4}{7}\right)^i \\ &= |l_{k+1}| \cdot \left(1 - \left(\frac{4}{7}\right)^{k+2}\right) \cdot \frac{7}{3} \\ &= \left(\frac{4}{7}\right)^k \cdot |l_1| \cdot \left(1 - \left(\frac{4}{7}\right)^{k+2}\right) \cdot \frac{7}{3} \end{aligned}$$

so $\frac{3}{7} \leq \frac{|l_{k+1}|}{|V|} \leq \frac{3}{7} \cdot \frac{1}{1-\left(\frac{4}{7}\right)^{k+2}}$, and $\frac{3}{7} \cdot \left(\frac{4}{7}\right)^k \leq \frac{|l_1|}{|V|} \leq \frac{3}{7} \cdot \left(\frac{4}{7}\right)^k \cdot \frac{1}{1-\left(\frac{4}{7}\right)^{k+2}}$.

CLAIM 14. *There exists $c' = c'(G_1)$ so that $|E(S, V \setminus S) \cap E(l_i, l_{i+1})| \geq c' \cdot d \cdot |\delta_i| \cdot |l_i|$.*

PROOF. Let G' be a G_1 component connecting l_i with l_{i+1} (so it has four vertices in l_i and seven in l_{i+1}). G' has no edges in $E(S, V \setminus S)$ if all or none of its vertices are in S. Otherwise, as G' is connected, it contributes at least one edge to $E(S, V \setminus S)$. The number of such G_1 components with all their vertices in S is at most $\min\{\sigma_i, \sigma_{i+1}\} \cdot \frac{|l_i|}{4}$. Therefore, there are at least $|\sigma_i - \sigma_{i+1}| \cdot \frac{|l_i|}{4}$ G_1 components with at least one vertex in S and one vertex that is not. □

CLAIM 15 (HOMOGENEITY BETWEEN LEVELS). *If there exists i so that $\frac{|\sigma - \sigma_i|}{\sigma} \geq \frac{1}{10}$, then*

$$|E(S, V \setminus S)| \geq c \cdot d \cdot |S| \cdot \left(\frac{4}{7}\right)^k$$

where $c > 0$ is some constant depending on G_1 only.

PROOF. Assume that there exists j so that $\frac{|\sigma-\sigma_j|}{\sigma} \geq \frac{1}{10}$. Let $\delta_i \equiv \sigma_{i+1} - \sigma_i$. By Claim 14, we have

$$\begin{aligned} |E(S, V \setminus S)| &\geq \sum_{i\in[k]} |E(S, V \setminus S) \cap E(l_i, l_{i+1})| \\ &\geq \sum_{i\in[k]} c' \cdot d \cdot |\delta_i| \cdot |l_i| \\ &\geq c' \cdot d \cdot |l_1| \sum_{i\in[k]} |\delta_i| \\ &\geq c' \cdot d \cdot |l_1| \cdot \left(\max_{i\in[k+1]} \sigma_i - \min_{i\in[k+1]} \sigma_i \right). \end{aligned}$$

By the initial assumption, there exists j so that $\frac{|\sigma-\sigma_j|}{\sigma} \geq \frac{1}{10}$, therefore $\max_i \sigma_i - \min_i \sigma_i \geq \frac{\sigma}{10}$, then $|E(S, V \setminus S)| \geq c' \cdot d \cdot |l_1| \cdot \frac{\sigma}{10}$. By Fact 13, $\geq c' \cdot d \cdot \frac{3}{7} \cdot \left(\frac{4}{7}\right)^k \cdot |V| \cdot \frac{\sigma}{10} \geq c \cdot d \cdot |S| \cdot \left(\frac{4}{7}\right)^k$, for any $c \leq \frac{c'}{10} \cdot \frac{3}{7}$. □

Let T_k be a tree corresponding to the recursive construction of G_k in the following way (see Figure 3). T_k is a tree of height $k+1$, where each internal node has four children. The root r of T_k corresponds to l_{k+1} (the largest level of G_k). The four children of r correspond to the largest levels of the four graphs that one can obtain by removing the level of vertices l_{k+1} from G_k. And so on. For every node u of T_k, denote by V_u the set of vertices in G_k corresponding to u. We thus have $|V_r| = 7^k$ where r is the root of T_k, $|V_u| = 7^{k-1}$ for each node u that is a child of r; and in general we have 4^i tree nodes u corresponding to a set of size $|V_u| = 7^{k-i+1}$. Each leaf l correspond to a set of size 1.

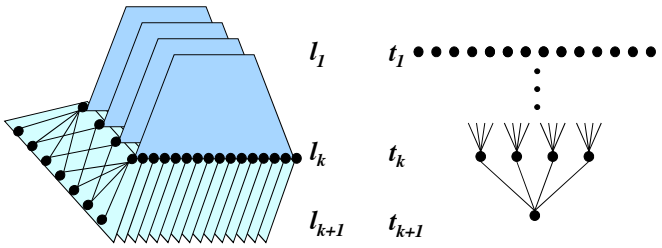

Figure 3: The graph G_k and its corresponding tree T_k.

For a tree node u, let us define $\rho_u = \frac{|S\cap V_u|}{|V_u|}$ to be the fraction of S nodes in V_u, and $\delta_u = |\rho_u - \rho_{p(u)}|$, where $p(u)$ is the parent of u (for the root r we let $p(r) = r$). We let t_i be the ith level of T_k, counting from the bottom, so t_{k+1} is the root and t_1 are the leaves.

FACT 16. *As $V_r = l_{k+1}$ we have $\rho_r = \sigma_{k+1}$. For a tree leaf $u \in t_1$, we have $|V_u| = 1$. Therefore $\rho_u \in \{0, 1\}$. The number of vertices u in t_1 with $\rho_u = 1$ is $\sigma_1 \cdot |l_1|$.*

CLAIM 17. *Let u_0 be an internal tree node, and let u_1, u_2, u_3, u_4 be its four children. Then*

$$\sum_i |E(S, V \setminus S) \cap E(V_{u_i}, V_{u_0})| \geq c'' \cdot d \cdot \sum_i |\rho_{u_i} - \rho_{u_0}| \cdot |V_{u_i}|$$

where $c'' = c''(G_1)$.

PROOF. The proof follows that of Claim 14. Let G' be a G_1 component, connecting V_{u_0} with $\bigcup_{i\in[4]} V_{u_i}$ (so it has seven vertices in V_{u_0} and one in each of V_{u_1},V_{u_2},V_{u_3},V_{u_4}). G' has no edges in $E(S, V \setminus S)$ if all or none of its vertices are in S. Otherwise, as G' is connected, it contributes at least one edge to $E(S, V \setminus S)$. The number of G_1 components with all their vertices in S is at most $\min\{\rho_{u_0}, \rho_{u_1}, \rho_{u_2}, \rho_{u_3}, \rho_{u_4}\} \cdot \frac{|V_{u_1}|}{4}$. Therefore, there are at least $\max_{i\in[4]}\{|\rho_{u_0} - \rho_{u_i}|\} \cdot \frac{|V_{u_1}|}{4} \geq \frac{1}{16} \cdot \sum_{i\in[4]} |\rho_{u_i} - \rho_{u_0}| \cdot |V_{u_i}|$ G_1 components with at least one vertex in S and one vertex that is not. □

We have $|E(S, V\setminus S)| = \sum_{u\in T_k} |E(S, V\setminus S) \cap E(V_u, V_{p(u)})|$. By Claim 17, this is at least $\sum_{u\in T_k} c'' \cdot d \cdot |\rho_u - \rho_{p(u)}| \cdot |V_u| = c'' \cdot d \cdot \sum_{i\in[k]} \sum_{u\in t_i} |\rho_u - \rho_{p(u)}| \cdot 7^{i-1} \geq c'' \cdot d \cdot \sum_{i\in[k]} \sum_{u\in t_i} |\rho_u - \rho_{p(u)}| \cdot 4^{i-1}$. As each internal node has four children, this is $c'' \cdot d \cdot \sum_{v\in t_1} \sum_{u\in v\sim r} |\rho_u - \rho_{p(u)}|$, where $v \sim r$ is the path from v to the root r. This is at least $c'' \cdot d \cdot \sum_{v\in t_1} |\rho_u - \rho_r|$, by the triangle inequality for $|\cdot|$. By Fact 16, it is $\geq c'' \cdot d \cdot |l_1| \cdot ((1-\sigma_1) \cdot \rho_r + \sigma_1 \cdot (1-\rho_r))$. By Claim 15, w.l.o.g., $|\sigma_{k+1} - \sigma|/\sigma \leq \frac{1}{10}$ and $|\sigma_1 - \sigma|/\sigma \leq \frac{1}{10}$. As $\rho_r = \sigma_{k+1}$, $|E(S, V \setminus S)| \geq \frac{3}{4} \cdot c'' \cdot d \cdot |l_1| \cdot \sigma$, and by Fact 13, $\geq c \cdot d \cdot |S| \cdot \left(\frac{4}{7}\right)^k$, for any $c \leq \frac{3}{4} \cdot c''$. This completes the proof of Lemma 10.

5. CONCLUSIONS AND OPEN PROBLEMS

We obtained a tight lower bound for the I/O-complexity of Strassen's and Strassen-like fast matrix multiplication algorithms. These bounds are optimal for the sequential model with two memory levels and with memory hierarchy. The lower bounds extend to the parallel model and other models. We are not aware of matching upper bounds for the parallel cases. However recently these bounds were attained by 2.5D parallel implementations for Strassen's algorithm and for Strassen-like algorithms [BDH$^+$11].

Strassen-like Algorithms.

A Strassen-like algorithm has a recursive structure that utilizes a base case: multiplying two n_0-by-n_0 matrices using $m(n_0)$ multiplications. Given two matrices of size n-by-n, it splits them into n_0-by-n_0 blocks (each of size $\frac{n}{n_0}$-by-$\frac{n}{n_0}$), and works blockwise, according to the base case algorithm. Additions (and subtractions) in the base case are interpreted as additions (and subtractions) of blocks. These are performed element-wise. Multiplications in the base case are interpreted as multiplications of blocks. These are performed by recursively calling the algorithm. The arithmetic count of the algorithm is then $T(n) = m(n_0) \cdot T\left(\frac{n}{n_0}\right) + O(n^2)$, so $T(n) = \Theta(n^{\omega_0})$ where $\omega_0 = \log_{n_0} m(n_0)$. We further demand that the Dec_1C is a connected graph.

This class includes Winograd's variant of Strassen's algorithm [Win71], which uses 15 additions rather than 18, but not the cubic algorithm, where Dec_1C is composed of four disconnected graphs (corresponding to the four outputs). We conjecture that Dec_1C is connected for all the following fast matrix-multiplication algorithms

and they are all Strassen-like: [Pan80, Bin80, Sch81, Rom82, CW82, Str87, CW87], (see [BCS97] for discussion of these algorithms), as well as [CKSU05], where the base case utilizes a novel group-theoretic approach.

To prove Theorem 3, which generalizes the I/O-complexity

lower bound of Strassen's algorithm (Theorem 1) to all Strassen-like algorithms, we note the following:

The entire proof of Theorem 1, and in particular, the computations in the proof of Lemma 10, hold for any Strassen-like algorithm, where we plug in $n_0^2, m(n_0)$, and $\frac{n_0}{m(n_0)}$ instead of $4, 7$, and $\frac{4}{7}$. For bounding the asymptotic I/O-complexity , we do not care about the number of internal vertices of Dec_1C; we need only to know that it is connected, and to know the sizes n_0 and $m(n_0)$. The only nontrivial adjustment is to show the equivalent of Fact 9: that the $Dec_{\log n}C$ graph is of bounded degree.

CLAIM 18. *The $Dec_{\log n}C$ graph of any Strassen-like algorithm is of degree bounded by a constant.*

PROOF. If the set of input vertices of Dec_1C, and the set of its output vertices are disjoint, then $Dec_{\log n}C$ is of constant bounded degree (its maximal degree is at most twice the largest degree of Dec_1C).

Assume (towards contradiction) that the base graph Dec_1C has an input vertex which is also an output vertex. An output vertex represents the inner product of two n_0-long vectors. The corresponding bilinear polynomial is irreducible. This is a contradiction, since an input vertex represents the multiplication of a (weighted) sum of elements of A with a (weighted) sum of elements of B. □

Other Fast Matrix Multiplication Algorithms.

Another class of matrix multiplication algorithms, the *uniform, non-stationary* algorithms, allows mixing of schemes of the previous (Strassen-like) class. In each recursive level, a different scheme may be used. The CDAG has a repeating structure inside one level, but the structure may differ between two distinct levels. This class includes, for example, algorithms that optimize for input sizes, (for size that is not an integer power of a constant integer). The class also includes algorithms that cut the recursion off at some point, and then switch to the classical algorithm. For these and other implementation issues, see [DHSS94, HLJJ+96] (sequential model) and [DS04] (parallel model). The I/O-complexity lower-bound generalizes to this class, and will appear in a separate note [BDHS11c].

A third class, the *non-uniform, non-stationary* algorithms, allows recursive calls to have different structure, even when they refer to multiplication of matrices in the same recursive level. It is not clear how to analyze the expansion of the CDAG of an algorithm in the third class, although we are not aware of any algorithm in this class. Such an analysis, applied to the base case of [CKSU05], may improve the I/O-complexity lower bound by a (large) constant.

Multiplication of rectangular matrices have seen a series of increasingly fast algorithms culminating in Coppersmith's algorithm [Cop97]. We suggest the first lower bounds of the communication costs for these algorithms, and show that in some cases they are attainable, and therefore optimal [BDHS11b].

As fast matrix multiplication is a basic building block in many fast algorithms in linear-algebra (e.g., LU, QR, Sylvester equation) their I/O-complexity can in many cases be determined using our approach [BDHS11a].

Other Algorithms, Other Hardware.

Our lower bounds, as well as most of the previous lower bounds deal with linear algebra and numerical analysis algorithms[19]. Our new approach, however, seems general enough to address I/O-complexity lower bounds of other algorithms (recall Lemma 8).

In many cases, the simplest recursive implementation of an algorithm turns out to be communication optimal (e.g., in the cases of matrix multiplication [FLPR99] and Cholesky decomposition [AP00, BDHS10a], but not in the case of LU decomposition [Tol97]). This leads to the question: when is the communication optimality of an algorithm determined by the expansion properties of the corresponding computation graphs? In this work we showed that such is the case for Strassen-like fast matrix multiplication algorithms.

It is of great interest to construct new models general enough to capture the rich and evolving design space of current and predicted future computers. Such models can be *homogeneous*, consisting of many layers, where the components of each layer are the same (e.g., a supercomputer with many identical multi-core chips on a board, many identical boards in a rack, many identical racks, and many identical levels of associated memory hierarchy); or *heterogeneous*, with components with different properties residing on the same level (e.g., CPUs alongside GPUs, where the latter can do some computations very quickly, but are much slower to communicate with).

Some experience has been acquired with such systems (e.g., using GPU assisted linear algebra computation [VD08]). A first step in analyzing such systems has been recently introduced by Ballard, Demmel, and Gearhart [BDG11], where they modeled heterogenous shared memory architectures, such as mixed GPU/CPU architecture, and obtained tight lower and upper bounds for $O(n^3)$ matrix multiplication.

However, there is currently no systematic theoretic way of obtaining upper and lower bounds for arbitrary hardware models. Expanding such results to other architectures and algorithmic techniques is a challenging goal. For example, recursive algorithms tend to be cache oblivious and communication optimal for the sequential hierarchy model. Finding an equivalent technique that would work for an arbitrary architecture is a fundamental open problem.

6. ACKNOWLEDGMENT

We thank Eran Rom, Edgar Solomonik, and Chris Umans for helpful discussions.

7. REFERENCES

[AP00] N. Ahmed and K. Pingali. Automatic generation of block-recursive codes. In *Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 368–378, London, UK, 2000. Springer-Verlag.

[ASS08] N. Alon, O. Schwartz, and A. Shapira. An elementary construction of constant-degree expanders. *Combinatorics, Probability & Computing*, 17(3):319–327, 2008.

[AV88] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related

[19]With the exception of a few results related to graph algorithms, e.g., in [MPP02] and in our [BDHS10b].

problems. *Commun. ACM*, 31(9):1116–1127, 1988.

[Bai88] D. H. Bailey. Extra-high speed matrix multiplication on the Cray-2. *SIAM J. Sci. Stat. Comput*, 9:603–607, 1988.

[BBF+07] M. A. Bender, G. S. Brodal, R. Fagerberg, R. Jacob, and E. Vicari. Optimal sparse matrix dense vector multiplication in the I/O-model. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 61–70, New York, NY, USA, 2007. ACM.

[BCG+08] G. E. Blelloch, R. A. Chowdhury, P. B. Gibbons, V. Ramachandran, S. Chen, and M. Kozuch. Provably good multicore cache performance for divide-and-conquer algorithms. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 501–510, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[BCS97] P. Bűrgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*. Number 315 in Grundlehren der mathematischen Wissenschaften. Springer Verlag, 1997.

[BDG11] G. Ballard, J. Demmel, and A. Gearhart. Communication bounds for heterogeneous architectures. In *23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2011)*, 2011. (to appear as a "brief announcement").

[BDH+11] G. Ballard, J. Demmel, O. Holtz, E. Rom, and O. Schwartz. Communication-Minimizing Parallel Implementation for Strassen's Algorithm. Unpublished, 2011.

[BDHS10a] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Communication-optimal parallel and sequential Cholesky decomposition. *SIAM Journal on Scientific Computing*, 32(6):3495–3523, December 2010.

[BDHS10b] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in linear algebra. Submitted. Available from http://arxiv.org/abs/0905.2485, 2010.

[BDHS11a] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing Communication in Fast Linear Algebra. Unpublished, 2011.

[BDHS11b] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Revisiting Coppersmith's "Rectangular matrix multiplication revisited" for I/O-Complexity. Unpublished, 2011.

[BDHS11c] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. The Communication Costs of Hybrid Algorithms for Fast Matrix Multiplication. Unpublished, 2011.

[Bin80] D. Bini. Relations between exact and approximate bilinear algorithms. applications. *Calcolo*, 17:87–97, 1980. 10.1007/BF02575865.

[BP99] G. Bilardi and F. Preparata. Processor-time tradeoffs under bounded-speed message propagation: Part II, lower boundes. *Theory of Computing Systems*, 32(5):1432–4350, 1999.

[BPD00] G. Bilardi, A. Pietracaprina, and P. D'Alberto. On the space and access complexity of computation DAGs. In *WG '00: Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 47–58, London, UK, 2000. Springer-Verlag.

[BZ88] Y. D. Burago and V. A. Zalgaller. *Geometric Inequalities*, volume 285 of *Grundlehren der Mathematische Wissenschaften*. Springer, Berlin, 1988.

[Can69] L. Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis, Montana State University, Bozeman, MN, 1969.

[CKSU05] H. Cohn, R. D. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic algorithms for matrix multiplication. In *FOCS*, pages 379–388, 2005.

[Cop97] D. Coppersmith. Rectangular matrix multiplication revisited. *J. Complex.*, 13:42–49, March 1997.

[CR06] R. A. Chowdhury and V. Ramachandran. Cache-oblivious dynamic programming. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 591–600, New York, NY, USA, 2006. ACM.

[CW82] D. Coppersmith and S. Winograd. On the asymptotic complexity of matrix multiplication. *SIAM Journal on Computing*, 11(3):472–492, 1982.

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[DHSS94] C. C. Douglas, M. Heroux, G. Slishman, and R. 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.

[DS04] F. Desprez and F. Suter. Impact of mixed-parallelism on parallel implementations of the Strassen and Winograd matrix multiplication algorithms: Research articles. *Concurrency and Computation: Practice and Experience*, 16(8):771–797, 2004.

[FLPR99] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 285, Washington, DC, USA, 1999. IEEE Computer Society.

[GSP04] S. L. Graham, M. Snir, and C. A. Patterson, editors. *Getting up to Speed: The Future of Supercomputing*. Report of National Research

Council of the National Academies Sciences. The National Academies Press, Washington, D.C., 2004. 289 pages, http://www.nap.edu.

[HK81] J. W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 326–333, New York, NY, USA, 1981. ACM.

[HLJJ+96] S. Huss-Lederman, E. M. Jacobson, J. R. Johnson, A. Tsao, and T. Turnbull. Implementation of Strassen's algorithm for matrix multiplication. In *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, page 32, Washington, DC, USA, 1996. IEEE Computer Society.

[ITT04] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, 2004.

[KKK10] M. Koucky, V. Kabanets, and A. Kolokolova. Expanders made elementary, 2010. In preparation, Available from http://www.cs.sfu.ca/∼kabanets/papers/expanders.pdf.

[Lei08] C. E. Leiserson. Personal communication with G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, 2008.

[LV83] G. Lev and L. G. Valiant. Size bounds for superconcentrators. *Theoretical Computer Science*, 22(3):233–251, 1983.

[LW49] L. H. Loomis and H. Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the AMS*, 55:961–962, 1949.

[Mih89] M. Mihail. Conductance and convergence of Markov chains: A combinatorial treatment of expanders. In *Proceedings of the Thirtieth Annual IEEE Symposium on Foundations of Computer Science*, pages 526–Ű531, 1989.

[MPP02] J. P. Michael, M. Penner, and V. K. Prasanna. Optimizing graph algorithms for improved cache performance. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2002), Fort Lauderdale, FL*, pages 769–782, 2002.

[Pan80] V. Y. Pan. New fast algorithms for matrix operations. *SIAM Journal on Computing*, 9(2):321–342, 1980.

[Raz03] R. Raz. On the complexity of matrix product. *SIAM J. Comput.*, 32(5):1356–1369 (electronic), 2003.

[Rom82] F. Romani. Some properties of disjoint sums of tensors related to matrix multiplication. *SIAM Journal on Computing*, 11(2):263–267, 1982.

[RVW02] O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, 155(1):157–187, 2002.

[Sav94] J. Savage. Space-time tradeoffs in memory hierarchies. Technical report, Brown University, Providence, RI, USA, 1994.

[Sch81] A. Schönhage. Partial and total matrix multiplication. *SIAM Journal on Computing*, 10(3):434–455, 1981.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[Str87] V. Strassen. Relative bilinear complexity and matrix multiplication. *Journal fűr die reine und angewandte Mathematik (Crelles Journal)*, 1987(375–376):406–443, 1987.

[Tol97] S. Toledo. Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 18(4):1065–1081, 1997.

[VD08] V. Volkov and J. Demmel. Benchmarking GPUs to tune dense linear algebra. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, 2008. IEEE Press.

[Win71] S. Winograd. On the multiplication of 2 × 2 matrices. *Linear Algebra Appl.*, 4(4):381–388., October 1971.

[YM88] C.-Q. Yang and B.P. Miller. Critical path analysis for the execution of parallel and distributed programs. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 366–373, Jun. 1988.

APPENDIX

A. EXPANSION ESTIMATION BY GRAPH DECOMPOSITION

DEFINITION 1 (GRAPH DECOMPOSITION). *We say that the set of graphs* $\{G'_i = (V_i, E_i)\}_{i\in[l]}$ *is an* edge-disjoint decomposition *of* $G = (V, E)$ *if* $V = \bigcup_i V_i$ *and* $E = \biguplus_i E_i$.

PROOF. (of Claim 5) Let $U \subseteq V$ be of size $U \leq |V'|/2$. Let $\{G'_i = (V_i, E_i)\}_{i\in[l]}$ be an edge-disjoint decomposition of G, where every G_i is isomorphic to G'. Then

$$\begin{aligned} |E_G(U, V \setminus U)| &= \sum_{i\in[l]} |E_{G'_i}(U_i, V_i \setminus U_i)| \geq \sum_{i\in[l]} h(G'_i) \cdot d' \cdot |U_i| \\ &= h(G') \cdot d' \cdot \sum_{i\in[l]} |U_i| \geq h(G') \cdot d' \cdot |U| \ . \end{aligned}$$

Therefore $\frac{|E_G(U,V\setminus U)|}{d\cdot|U|} \geq h(G') \cdot \frac{d'}{d}$. □

B. STRASSEN'S FAST MATRIX MULTIPLICATION ALGORITHM

Strassen's original algorithm follows [Str69]. See [Win71] for Winograd's variant, which reduces the number of additions.

Algorithm 1 Matrix Multiplication: Strassen's Algorithm

Input: Two $n \times n$ matrices, A and B.

if $n = 1$ **then**
  $C_{11} = A_{11} \cdot B_{11}$
else
  {Decompose A into four equal square blocks $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ and the same for B.}
  $M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$
  $M_2 = (A_{21} + A_{22}) \cdot B_{11}$
  $M_3 = A_{11} \cdot (B_{12} - B_{22})$
  $M_4 = A_{22} \cdot (B_{21} - B_{11})$
  $M_5 = (A_{11} + A_{12}) \cdot B_{22}$
  $M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$
  $M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$
  $C_{11} = M_1 + M_4 - M_5 + M_7$
  $C_{12} = M_3 + M_5$
  $C_{21} = M_2 + M_4$
  $C_{22} = M_1 - M_2 + M_3 + M_6$
end if
return C

See [DHSS94, HLJJ+96, Bai88] for implementation issues of Strassen's algorithm.