

# Graph Expansion and Communication Costs of Fast Matrix Multiplication

Grey Ballard, University of California at Berkeley  
 James Demmel, University of California at Berkeley  
 Olga Holtz, University of California at Berkeley and Technische Universität Berlin  
 Oded Schwartz, University of California at Berkeley

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.

In the sequential case, where the processor has a fast memory of size  $M$ , too small to store three  $n$ -by- $n$  matrices, the lower bound on the number of words moved between fast and slow memory is, for a large class of matrix multiplication algorithms,  $\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \cdot M\right)$ , where  $\omega_0$  is the exponent in the arithmetic count (e.g.,  $\omega_0 = \lg 7$  for Strassen, and  $\omega_0 = 3$  for conventional matrix multiplication). With  $p$  parallel processors, each with fast memory of size  $M$ , the lower bound is asymptotically lower by a factor of  $p$ . These bounds are attainable both for sequential and for parallel algorithms and hence optimal.

Categories and Subject Descriptors: F.2.1 [Analysis of Algorithms and Problem Complexity]: Computations on Matrices

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Communication-avoiding algorithms, Fast matrix multiplication, I/O-Complexity

## ACM Reference Format:

Ballard, G., Demmel, J., Holtz, O., and Schwartz, O. 2011. Graph Expansion and Communication Costs of Fast Matrix Multiplication. *J. ACM* V, N, Article A (January 20XX), 25 pages.  
 DOI = 10.1145/0000000.0000000 <http://doi.acm.org/10.1145/0000000.0000000>

## 1. INTRODUCTION

The communication, or I/O complexity, of an algorithm (e.g., transferring data between the CPU and memory devices, or between parallel processors) often costs significantly more time than its arithmetic. It is therefore of interest (1) to obtain lower bounds

---

A preliminary version of this paper appeared in Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'11) [Ballard et al. 2011c] and received the best paper award.

This work is supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227); Additional support from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, and Samsung. Research 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; Research supported by the Sofja Kovalevskaja programme of Alexander von Humboldt Foundation and by the National Science Foundation under agreement DMS-0635607; Research supported by ERC Starting Grant Number 239985.

Author's addresses: G. Ballard, Computer Science Division, University of California, Berkeley, CA 94720; J. Demmel, Mathematics Department and Computer Science Division, University of California, Berkeley, CA 94720; O. Holtz, Department of Mathematics, University of California, Berkeley; O. Schwartz, Computer Science Division, University of California, Berkeley, CA 94720.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).

© 20XX ACM 0004-5411/20XX/01-ARTA \$10.00

DOI 10.1145/0000000.0000000 <http://doi.acm.org/10.1145/0000000.0000000>

for the communication needed, and (2) to design and implement algorithms attaining these lower bounds. Communication also requires much more energy than arithmetic, and saving energy may be even more important than saving time.

Communication time per data unit varies by orders of magnitude, from order of  $10^{-9}$  seconds for an L1 cache reference, to order of  $10^{-2}$  seconds for disk access. The variation can be even more dramatic when communication occurs over networks or the internet. While Moore’s Law predicts an exponential increase of hardware density in general, the annual improvement rate of time-per-arithmetic-operation has, over the years, consistently exceeded that of time-per-word read/write [Graham et al. 2004; Fuller and Millett 2011]. The fraction of running time spent on communication is thus expected to increase further.

### 1.1. 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<sup>1</sup>. 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  $\alpha$  is the *latency* (seconds per message) and  $\beta$  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 reside in slow memory, and are too large to fit in the smaller fast memory. Our goal then is to minimize both bandwidth and latency costs.<sup>2</sup>

In the parallel case, we assume  $p$  processors, each with memory of size  $M$  (or with larger memory size, as long as we never use more than  $M$  in each processor). We are interested in the communication among the processors. As in the sequential case, we assume that a message of  $n$  words can 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 word and message counts respectively. However, we count the number of words and messages communicated along the critical path as defined in [Yang and Miller 1988] (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 ( $\alpha$  and  $\beta$ ) are the same between each pair of processors (this assumption is for ease of presentation and can be dropped, using the approach of [Ballard et al. 2011b]; see Section 6.2), (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

<sup>1</sup>See [Ballard et al. 2010] for definition of a model with memory hierarchy, and a reduction from the two-level model. All bounds in this paper thus apply to the model with memory hierarchy as well.

<sup>2</sup>The sequential communication model used here is sometimes called the *two-level I/O model* or *disk access machine (DAM)* model (see [Aggarwal and Vitter 1988; Bender et al. 2010; Chowdhury and Ramachandran 2006]). Our bandwidth cost model follows that of [Hong and Kung 1981] and [Irony et al. 2004] 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.

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. Note that assuming all-to-all connectivity among processors yields valid lower bounds for all distributed computers, even though real machines have more restrictive networks.

### 1.2. The Computation Graph and Implementations of an Algorithm

The computation performed by an algorithm on a given input can be modeled (see Section 3) 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 (or scheduling) of a CDAG determines, in the parallel model, which arithmetic operations are performed by which of the  $p$  processors. This corresponds to partitioning the 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 a CDAG may vary greatly in their communication costs. The *I/O-complexity of an algorithm* is the minimum bandwidth cost of the algorithm as a function of the input size, over all possible implementations of the CDAG corresponding to the given input size. The *I/O-complexity of a problem* is defined to be the minimum I/O-complexity of all algorithms for this problem. A lower bound of the I/O-complexity of an algorithm is therefore a result of the form: any implementation of algorithm  $Alg$  and input size  $n$  requires at least  $X(n)$  communication. An upper bound is of the form: there is an implementation for algorithm  $Alg$  and input size  $n$  that requires at most  $X(n)$  communication. We detail below some of the I/O-complexity lower and upper bounds of specific algorithms, or a class of algorithms. *I/O-complexity lower bounds for a problem* are claims of the form: any algorithm for a problem  $P$  and input size  $n$  requires at least  $X(n)$  communication. These are much harder to find (but see for example [Demmel et al. 2012]).

The lower bounds in this paper are for all *implementations* for a family of algorithms: *Strassen-like* fast matrix multiplication. Generally speaking, a Strassen-like algorithm utilizes an algorithm for multiplying two constant-size matrices in order to recursively multiply matrices of arbitrary size; see Section 5 for precise definition and technical assumptions.

### 1.3. Previous Work on Classical Algorithms

Consider the classical  $\Theta(n^3)$  algorithm for matrix multiplication<sup>3</sup>. While its naïve implementations are communication inefficient, communication-minimizing sequential and parallel variants of this algorithm were constructed, and proved optimal, by matching lower bounds [Cannon 1969; Hong and Kung 1981; Frigo et al. 1999; Irony et al. 2004].

In [Ballard et al. 2010; Ballard et al. 2011d] we generalize the results of [Hong and Kung 1981; Irony et al. 2004] regarding matrix multiplication, to obtain new

<sup>3</sup>Here we mean any algorithm that computes using the  $n^3$  scalar multiplications, whether this is done recursively, iteratively, block-wise or any other way.

I/O-complexity lower bounds for a much wider variety of algorithms. This includes all “classical” algorithms for  $LU$  factorization, Cholesky factorization, and  $LDL^T$  factorization, as well as many algorithms for the  $QR$  factorization, eigenvalue decompositions, and the singular value decomposition (SVD). 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), and to certain graph-theoretic problems<sup>4</sup>. They apply to sequential and parallel algorithms, and most of our bounds are shown to be tight.

Communication cost optimal algorithms for square classical matrix multiplication are well known. Algorithms for dense LU, Cholesky, QR, eigenvalue problems and the SVD with optimal communication costs are more recent. These include [Gustavson 1997; Toledo 1997; Elmroth and Gustavson 1998; Frigo et al. 1999; Ahmed and Pingali 2000; Frens and Wise 2003; Demmel et al. 2012; Grigori et al. 2008; Grigori et al. 2011; Ballard et al. 2009; David et al. 2010; Ballard et al. 2011a] and have not yet all been implemented and made available as part of standard libraries like LAPACK [Anderson et al. 1992] and ScaLAPACK [Blackford et al. 1997]. See [Ballard et al. 2011d] for more details.

In [Ballard et al. 2010; Ballard et al. 2011d] we use the approach of [Irony et al. 2004], based on the Loomis-Whitney geometric theorem [Loomis and Whitney 1949; Burago and Zalgaller 1988], 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 [Strassen 1969] and other fast matrix multiplication algorithms (e.g., [Coppersmith and Winograd 1990; Cohn et al. 2005]).

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. The problem of minimizing communication in parallel classical matrix multiplication was addressed in [Cannon 1969] almost simultaneously with the publication of Strassen’s fast matrix multiplication algorithm [Strassen 1969]. Moreover, an I/O-complexity lower bound for the classical matrix multiplication algorithm has been known for three decades [Hong and Kung 1981]. Nevertheless, the I/O-complexity of Strassen’s fast matrix multiplication and similar algorithms has not been resolved.

In this paper we obtain first communication cost lower bounds for Strassen’s and other fast matrix multiplication algorithms, in the sequential and parallel models. These bounds are attainable both for sequential (see below) and for parallel algorithms (by our recent work [Ballard et al. 2012b]) and so optimal.

#### 1.4. Communication Costs of Fast Matrix Multiplication

*1.4.1. Upper bounds.* The I/O-complexity  $IO(n, M)$  of Strassen’s algorithm (see Algorithm 1, Appendix A), applied to  $n$ -by- $n$  matrices on a machine with fast memory of size  $M$ , can be bounded above as follows: the recursion consists of computing seven subproblems and performing matrix additions, where the base case occurs when the problem fits entirely in the fast memory ( $3n^2 \leq M$ ). In the base case, read the two input sub-matrices into the fast memory, perform the matrix multiplication inside the fast memory, and write the result into the slow memory<sup>5</sup>. We thus have

<sup>4</sup>See [Michael et al. 2002] for bounds on graph-related problems, and our [Ballard et al. 2011d] for a detailed list of previously known and recently designed sequential and parallel algorithms that attain the above mentioned lower bounds.

<sup>5</sup>Here we assume that the recursion tree is traversed in the usual depth-first order.

$IO(n, M) \leq 7 \cdot IO\left(\frac{n}{2}, M\right) + O(n^2)$  and  $IO\left(\frac{\sqrt{M}}{3}, M\right) = O(M)$ . Thus

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

Note that this matches the lower bound stated in Theorem 1.1 below. In order to attain the latency lower bound as well, a careful choice of matrix layout is necessary. Morton-ordering (also known as bit-interleaved layout) enables the recursive algorithm to attain the latency lower bound; see [Frigo et al. 1999; Wise 2000] for more details.

The recent parallel algorithm for Strassen's matrix multiplication [Ballard et al. 2012b] has I/O-complexity

$$IO(n, p, M) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\lg 7} \cdot \frac{M}{p} + \frac{n^2}{p^{\frac{2}{\lg 7}}}\right)$$

where  $p$  is the number of processors and  $M$  is the size of the local memory. Note that this matches the lower bounds of Corollary 1.2 and Theorem 1.3 below. The latency lower bound is attained to within a logarithmic factor (in  $p$ ). A similar optimal algorithm for Strassen's matrix multiplication in the BSP model is presented in [McColl and Tiskin 1999].

*1.4.2. Lower bounds.* In this paper, we obtain a tight lower bound:

**THEOREM 1.1. (MAIN THEOREM)** *Consider Strassen's algorithm implemented on a sequential machine with fast memory of size  $M$ . Then for  $M \leq n^2$ , and assuming no recomputation<sup>6</sup>, the I/O-complexity of Strassen's algorithm is*

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

It holds for any implementation and any known variant of Strassen's algorithm<sup>7</sup> that is based on performing  $2 \times 2$  matrix multiplication with 7 scalar multiplications. 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 [Douglas et al. 1994; Huss-Lederman et al. 1996; Desprez and Suter 2004]. Note that Theorem 1.1 does not hold for values of  $M$  which are so large that the entire problem can fit into fast memory simultaneously. In the case that the input matrices start in fast memory and the output matrix finishes in fast memory, no communication is necessary.

For parallel algorithms, using a reduction from the sequential to the parallel model (see e.g., [Irony et al. 2004] or our [Ballard et al. 2011d]) this yields:

**COROLLARY 1.2.** *Consider Strassen's algorithm implemented on a parallel machine with  $p$  processors, each with a local memory of size  $M$ . There exists a constant  $c$  such that for  $M \leq c \cdot \frac{n^2}{p^{2/\lg 7}}$ , and assuming no recomputation, the I/O-complexity of*

<sup>6</sup>We assume no recomputation throughout the paper. By this we mean each vertex of the CDAG is computed exactly once. If recomputation is allowed then a write operation, for example, may be replaced by recomputing the flop when its output is needed, making the lower bound harder to prove in some cases, and incorrect in others. See Section 3.2 and the  $R2/D2$  discussion in [Ballard et al. 2011d].

<sup>7</sup>This lower bound for the sequential case seems to contradict the upper bound from [Frigo et al. 1999; Blelloch et al. 2008], due to a miscalculation in the former which is propagated in the latter ([Leiserson 2008]).

*Strassen's algorithm is*

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

While Corollary 1.2 does not hold for all sizes of local memory (relative to the problem size and number of processors), the following memory-independent lower bound can be proved using similar techniques [Ballard et al. 2012a] and holds for all local memory sizes, though it requires separate assumptions.

**THEOREM 1.3** ([BALLARD ET AL. 2012A]). *Suppose a parallel algorithm performing Strassen's matrix multiplication load balances the computation in an asymptotic sense and performs no redundant computation. Then, for sufficiently large  $p$ ,*

$$IO(n, p) = \Omega \left( \frac{n^2}{p^{2/\lg 7}} \right).$$

Note that the bound in Corollary 1.2 dominates the one in Theorem 1.3 for  $M = O \left( \frac{n^2}{p^{2/\lg 7}} \right)$ . See Section 5.3 for further discussion.

We can extend these bounds to a wider class of all Strassen-like matrix multiplication algorithms. Note that this class does not include all fast matrix multiplication algorithms or the classical algorithm (see Section 5.1 for definition of Strassen-like algorithms, and in particular the technical assumption in Section 5.1.1). 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 as for Strassen's algorithm, the I/O-complexity of  $Alg$  can be shown to be  $IO(n, M) = O \left( \left( \frac{n}{\sqrt{M}} \right)^{\omega_0} \cdot M \right)$ . We obtain a matching lower bound:

**THEOREM 1.4.** *Consider a recursive Strassen-like fast matrix multiplication algorithm with  $O(n^{\omega_0})$  arithmetic operations implemented on a sequential machine with fast memory of size  $M$ . There exists a constant  $c$  such that for  $M \leq c \cdot n^2$ , and assuming no recomputation, the I/O-complexity of the Strassen-like algorithm is*

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

Note that for the cubic recursive algorithm for matrix multiplication,  $\omega_0 = \lg 8 = 3$ , and the above formula is  $IO(n, M) = \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^3 \cdot M \right) = \Omega \left( \frac{n^3}{\sqrt{M}} \right)$  and identifies with the lower bounds of [Hong and Kung 1981] and [Irony et al. 2004]. While the lower bounds for  $\omega_0 = 3$  and for  $\omega_0 < 3$  have the same form, the proofs are completely different, and it is not clear whether our approach can be used to prove their lower bounds and vice versa.

**COROLLARY 1.5.** *Consider a Strassen-like algorithm implemented on a parallel machine with  $p$  processors, each with a local memory of size  $M$ . There exists a constant  $c$  such that for  $M \leq c \cdot \frac{n^2}{p^{2/\omega_0}}$ , and assuming no recomputation, the I/O-complexity of the Strassen-like algorithm is*

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

Theorem 1.3 also extends to Strassen-like algorithms [Ballard et al. 2012a], though we omit the statement of the more general theorem here. Both of the bounds are

attained by the generalizations of the parallel Strassen algorithm in [Ballard et al. 2012b].

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.

### 1.5. 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 related 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 [Mihail 1989; Alon et al. 2008; Koucky et al. 2010]) or by spectral analysis (in the spirit of what was done for the Zig-Zag expanders [Reingold et al. 2002]). 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 [Hong and Kung 1981] (see also [Savage 1995]). They use the red-blue pebble game to obtain tight lower bounds on the I/O-complexity of many algorithms, including classical  $\Theta(n^3)$  matrix multiplication, matrix-vector multiplication, and FFT. The proof is obtained by considering other properties of the CDAG (using dominator sets and minimal sets). While we assume no recomputation occurs, Hong and Kung's approach does allow recomputation. Comparing our approach to that of [Irony et al. 2004], one can view their approach (also used in [Ballard et al. 2011d; Ballard et al. 2012a]) 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 [Lev and Valiant 1983] 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 CDAGs with 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 properties of the corresponding CDAG (see e.g., [Savage 1994; Bilardi and Preparata 1999; Bilardi et al. 2000] and references therein).

### 1.6. 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 discuss the generalization of the bounds to other algorithms in Section 5, and present conclusions and open problems in Section 6.

## 2. PRELIMINARIES

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

$$h(G) \equiv \min_{U \subseteq V, |U| \leq |V|/2} \frac{|E(U, V \setminus U)|}{d \cdot |U|} \quad (1)$$

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.

*When  $G$  is not regular.* Note that CDAGs are typically not regular. If a graph  $G = (V, E)$  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  $G'$ . We use the convention that a loop adds 1 to the degree of a vertex. Note that for any  $S \subseteq V$ , we have  $|E_G(S, V \setminus S)| = |E_{G'}(S, V \setminus S)|$ , as none of the added loops contributes to the edge expansion of  $G'$ .

*Expansion of small sets.* For many graphs, small sets expand better than larger sets. Let  $h_s(G)$  denote the edge expansion for sets of size at most  $s$  in  $G$ :

$$h_s(G) \equiv \min_{U \subseteq V, |U| \leq s} \frac{|E(U, V \setminus U)|}{d \cdot |U|}. \quad (2)$$

For many interesting graph families,  $h_s(G)$  does not depend on  $|V(G)|$  when  $s$  is fixed, 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.

**LEMMA 2.1.** *Let  $G = (V, E)$  be a  $d$ -regular graph that can be decomposed into edge-disjoint (but not necessarily vertex-disjoint) copies of a graph  $G' = (V', E')$  with maximum degree  $d'$ . Then the edge expansion of  $G$  for sets of size at most  $|V'|/2$  is  $h(G') \cdot \frac{d'}{d}$ , namely*

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

Note that if  $G'$  is not regular then  $h(G')$  is not well-defined. We abuse this notation to mean the edge expansion of  $G'$  made regular by adding at most  $d'$  loops to each vertex. For proving this lemma, recall the definition of graph decomposition:

**Definition 2.2 (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  $E = \bigsqcup_i E_i$ .

**PROOF OF LEMMA 2.1.** 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  $G'$ . Let  $U_i = V_i \cap U$ . 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  $\frac{|E_G(U, V \setminus U)|}{d \cdot |U|} \geq h(G') \cdot \frac{d'}{d}$ .  $\square$

### 3. I/O-COMPLEXITY AND EDGE EXPANSION

In this section we recall the notion of the computation graph of an algorithm, then show how a partition argument connects the expansion properties of the computation graph and the I/O-complexity of the algorithm. A similar partition argument already appeared in [Irony et al. 2004], and then in our [Ballard et al. 2011d]. In both cases it is used to relate I/O-complexity to the Loomis-Whitney geometric bound [Loomis and Whitney 1949], which can be viewed, in this context, as an expansion guarantee for the corresponding graphs.

#### 3.1. The computation graph

For a given algorithm, we consider the CDAG  $G = (V, E)$ , 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<sup>8</sup>, 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.

### 3.2. 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 directed upwards in the total order. This total ordering can be thought of as the actual order in which the computations are performed. Let  $P$  be any partition of  $V$  into segments  $S_1, S_2, \dots$ , 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 total I/O-complexity is therefore bounded below by<sup>9</sup>

$$IO \geq \max_P \sum_{S \in P} (|R_S| + |W_S| - 2M) . \quad (3)$$

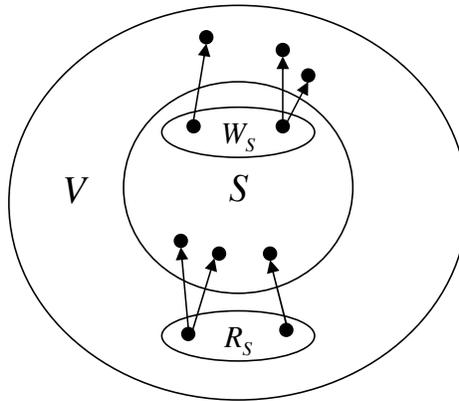


Fig. 1. A subset (segment)  $S$  and its corresponding read operands  $R_S$ , and write operands  $W_S$ .

<sup>8</sup>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.

<sup>9</sup>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.

### 3.3. 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<sup>10</sup>, maximum degree  $d$  and at least  $2|S|$  vertices, then (using the definition of  $h(G)$ ), we have

CLAIM 3.1.  $|R_S| + |W_S| \geq h(G) \cdot |S|$ .

PROOF. We have  $|E(S, V \setminus S)| \geq h(G) \cdot d \cdot |S|$ . Since  $E(S, V \setminus S) = E(R_S, S) \uplus E(W_S, V \setminus S)$  we have  $|E(S, V \setminus S)| = |E(R_S, S)| + |E(W_S, V \setminus S)| \leq d \cdot |R_S| + d \cdot |W_S|$  where the last inequality is by the degree bound. The claim follows.  $\square$

Combining this with (3) and choosing to partition  $V$  into  $|V|/s$  segments of equal size  $s$ , we obtain:  $IO \geq \max_s \frac{|V|}{s} \cdot (h(G) \cdot s - 2M) = \Omega(|V| \cdot h(G))$ . 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:  $IO \geq \max_s \frac{|V|}{s} \cdot (h_s(G) \cdot s - 2M)$ . Choosing the minimal  $s$  so that

$$h_s(G) \cdot s \geq 3M \quad (4)$$

we obtain

$$IO \geq \frac{|V|}{s} \cdot M. \quad (5)$$

The existence of a value  $s \leq \frac{|V|}{2}$  that satisfies condition (4) is not always guaranteed. In the next section we confirm the existence of such  $s$  for Strassen's CDAG, for sufficiently large  $|V|$ . 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.

In some cases, the computation graph  $G$  does not fit this analysis: it may not be regular, it may have 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:

LEMMA 3.2. *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, for sufficiently large  $|V'|$ , the I/O-complexity of  $Alg$  is*

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

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

The correctness of this lemma follows from Equations (4) and (5), and from the fact that at least an  $\alpha/2$  fraction of the segments have at least  $\alpha \cdot s$  of their vertices in  $G'$  (otherwise  $V' < \frac{\alpha}{2} \cdot V/s \cdot s + (1 - \frac{\alpha}{2}) \cdot V/s \cdot \frac{\alpha}{2}s < \alpha V$ ).

<sup>10</sup>The direction of the edges does not matter much for the expansion-bandwidth argument: treating all edges as undirected changes the I/O-complexity estimate by a factor of 2 at most. For simplicity, we will treat  $G$  as undirected.

#### 4. EXPANSION PROPERTIES OF STRASSEN'S ALGORITHM

Recall Strassen's algorithm for matrix multiplication (see Algorithm 1 in Appendix A) and consider its computation graph (see Figure 2). Let  $H_i$  be the computation graph of Strassen's algorithm for recursion of depth  $i$ , hence  $H_{\lg n}$  corresponds to the computation for input matrices of size  $n \times n$ .  $H_{\lg n}$  has the following structure:

- Encode  $A$ : generate weighted sums of elements of  $A$  (this corresponds to the left factors of lines 5-11 of the algorithm).
- Similarly encode  $B$  (this corresponds to the right factors of lines 5-11 of the algorithm).
- Then multiply the encodings of  $A$  and  $B$  element-wise (this corresponds to line 2 of the algorithm).
- Finally, decode  $C$ , by taking weighted sums of the products (this corresponds to lines 12-15 of the algorithm).

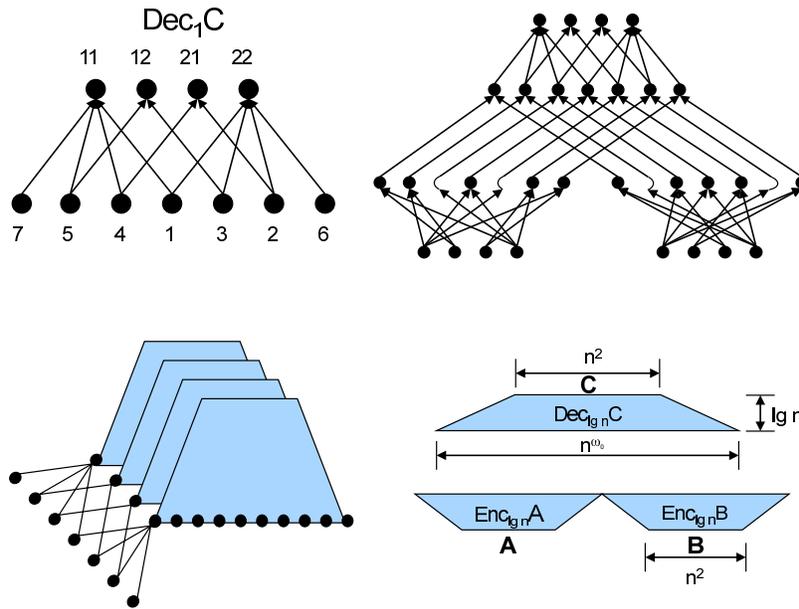


Fig. 2. The computation graph of Strassen's algorithm (See Algorithm 1 in Appendix). Top left:  $Dec_1 C$ . Top right:  $H_1$ . Bottom left:  $Dec_{\lg n} C$ . Bottom right:  $H_{\lg n}$ .

**COMMENT 4.1.**  $Dec_1 C$  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 (not necessarily balanced) 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, there is no change in the number of input and output vertices. Therefore the arguments in the proof of Lemma 4.9 still hold.

#### 4.1. The computation graph for $n$ -by- $n$ matrices

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.

*4.1.1. A top-down construction of the computation graph.* We next construct the computation graph  $H_{i+1}$  by constructing  $Dec_{i+1} C$  (from  $Dec_i C$  and  $Dec_1 C$ ) and similarly constructing  $Enc_{i+1} A$  and  $Enc_{i+1} B$ , then composing the three parts together.

- Replicate  $Dec_1 C$   $7^i$  times.
- Replicate  $Dec_i C$  four times.
- Identify the  $4 \cdot 7^i$  output vertices of the copies of  $Dec_1 C$  with the  $4 \cdot 7^i$  input vertices of the copies of  $Dec_i C$ :
  - Recall that each  $Dec_1 C$  has four output vertices.
  - The set of each first output vertex of the  $7^i$   $Dec_1 C$  graphs is identified with the set of  $7^i$  input vertices of the first copy of  $Dec_i C$ .
  - The set of each second output vertex of the  $7^i$   $Dec_1 C$  graphs is identified with the set of  $7^i$  input vertices of the second copy of  $Dec_i C$ . And so on.
  - We make sure that the  $j$ th input vertex of a copy of  $Dec_i C$  is identified with an output vertex of the  $j$ th copy of  $Dec_1 C$ .
- We similarly obtain  $Enc_{i+1} A$  from  $Enc_i A$  and  $Enc_1 A$ ,
- and  $Enc_{i+1} B$  from  $Enc_i B$  and  $Enc_1 B$ .
- For every  $i$ ,  $H_i$  is obtained by connecting edges from the  $j$ th output vertices of  $Enc_i A$  and  $Enc_i B$  to the  $j$ th input vertex of  $Dec_i C$ .

This completes the construction. Let us note some properties of these graphs.

The graph  $Dec_1 C$  has no vertices which are both input and output. As all out-degrees are at most 4 and all in degree are at most 2 (Recall Comment 4.1) we have:

**FACT 4.2.** *All vertices of  $Dec_{\lg n} C$  are of degree at most 6.*

However,  $Enc_1 A$  and  $Enc_1 B$  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.

*4.1.2. Combinatorial Estimation of the Expansion.* Let  $G_k = (V, E)$  be  $Dec_k C$ , 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  $Dec_k C$  (after adding loops to make it regular).

Let  $l_i$  be the  $i$ th level of vertices of  $G_k$ , so  $4^k = |l_1| < |l_2| < \dots < |l_i| = 4^{k-i+1} 7^{i-1} < \dots < |l_{k+1}| = 7^k$ . 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  $S$  at level  $i$ . Due to averaging, we observe the following:

**FACT 4.3.** *There exist  $i$  and  $i'$  such that  $\sigma_i \leq \sigma \leq \sigma_{i'}$ .*

FACT 4.4.

$$\begin{aligned} |V| &= \sum_{i=1}^{k+1} |l_i| = \sum_{i=0}^k |l_{k+1}| \cdot \left(\frac{4}{7}\right)^i \\ &= |l_{k+1}| \cdot \left(1 - \left(\frac{4}{7}\right)^{k+1}\right) \cdot \frac{7}{3} \\ &= \left(\frac{7}{4}\right)^k \cdot |l_1| \cdot \left(1 - \left(\frac{4}{7}\right)^{k+1}\right) \cdot \frac{7}{3} \end{aligned}$$

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

CLAIM 4.5. *Let  $\delta_i \equiv \sigma_{i+1} - \sigma_i$ . Then  $|E(S, V \setminus S) \cap E(l_i, l_{i+1})| \geq c_1 \cdot d \cdot |\delta_i| \cdot |l_i|$ , where  $c_1$  is a constant which depends on  $G_1$ .*

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  $\min\left\{\frac{\sigma_i \cdot |l_i|}{4}, \frac{\sigma_{i+1} \cdot |l_{i+1}|}{7}\right\} = \min\{\sigma_i, \sigma_{i+1}\} \cdot \frac{|l_i|}{4}$ . Similarly, the number of such  $G_1$  components with none of their vertices in  $S$  is at most  $\min\{1 - \sigma_i, 1 - \sigma_{i+1}\} \cdot \frac{|l_i|}{4}$ . 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. The claim follows with  $c_1 = \frac{1}{4d}$ .  $\square$

CLAIM 4.6 (HOMOGENEITY BETWEEN LEVELS). *If there exists  $i$  so that  $\frac{|\sigma - \sigma_i|}{\sigma} \geq \frac{1}{10}$ , then*

$$|E(S, V \setminus S)| \geq c_2 \cdot d \cdot |S| \cdot \left(\frac{4}{7}\right)^k$$

where  $c_2$  is a constant which depends on  $G_1$ .

PROOF. Assume that there exists  $j$  so that  $\frac{|\sigma - \sigma_j|}{\sigma} \geq \frac{1}{10}$ . By Claim 4.5, we have

$$\begin{aligned} |E(S, V \setminus S)| &\geq \sum_{i \in [k]} |E(S, V \setminus S) \cap E(l_i, l_{i+1})| \\ &\geq \sum_{i \in [k]} c_1 \cdot d \cdot |\delta_i| \cdot |l_i| \\ &\geq c_1 \cdot d \cdot |l_1| \sum_{i \in [k]} |\delta_i| \\ &\geq c_1 \cdot d \cdot |l_1| \cdot \left(\max_{i \in [k+1]} \sigma_i - \min_{i \in [k+1]} \sigma_i\right). \end{aligned}$$

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

$$|E(S, V \setminus S)| \geq c_1 \cdot d \cdot |l_1| \cdot \frac{\sigma}{10}$$

By Fact 4.4,  $|l_1| \geq \frac{3}{7} \cdot \left(\frac{4}{7}\right)^k \cdot |V|$ ,

$$\geq c_1 \cdot d \cdot \frac{3}{7} \cdot \left(\frac{4}{7}\right)^k \cdot |V| \cdot \frac{\sigma}{10}$$

As  $|S| = \sigma \cdot |V|$ ,

$$\geq c_2 \cdot d \cdot |S| \cdot \left(\frac{4}{7}\right)^k$$

for any  $c_2 \leq \frac{1}{10} \cdot \frac{3}{7} \cdot c_1$ .  $\square$

Let  $T_k$  correspond 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  $u$ , so if  $u$  is at level  $i$  of  $T_k$  then  $V_u \subseteq l_i$ . One can think of  $T_k$  as a quadtree partitioning of matrix  $C$  into blocks, where  $V_u$  is the largest level of the decoding subgraph of the  $C$  sub-block corresponding to  $u$ . Therefore  $|V_r| = 7^k$  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  $l$  corresponds to a set of size 1.

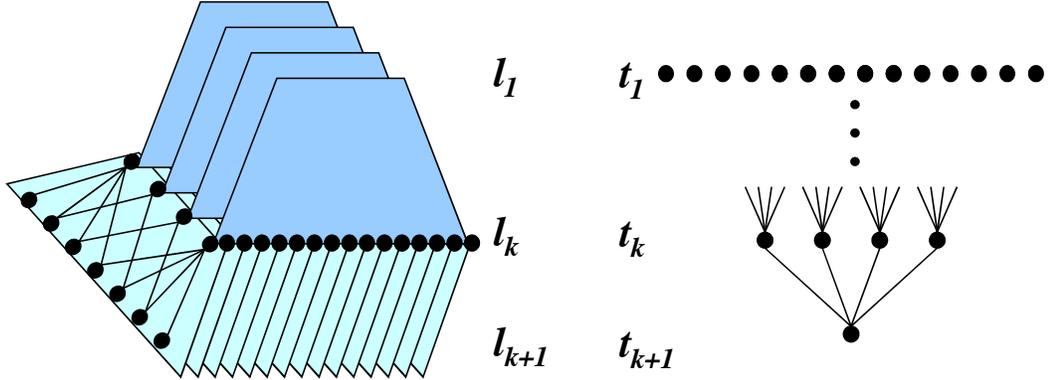


Fig. 3. The graph  $G_k$  and its corresponding tree  $T_k$ .

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  $i$ th level of  $T_k$ , counting from the bottom, so  $t_{k+1}$  is the root and  $t_1$  are the leaves.

**FACT 4.7.** *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 4.8. *Let  $u_0$  be an internal tree node, and let  $u_1, u_2, u_3, u_4$  be its four children. Then*

$$\sum_i |E(S, V \setminus S) \cap E(V_{u_i}, V_{u_0})| \geq c_1 \cdot d \cdot \sum_i |\rho_{u_i} - \rho_{u_0}| \cdot |V_{u_i}|$$

where  $c_1$  is a constant that depends on  $G_1$ .

PROOF. The proof follows that of Claim 4.5. 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  $\min\{\rho_{u_0}, \rho_{u_1}, \rho_{u_2}, \rho_{u_3}, \rho_{u_4}\} \cdot |V_{u_1}|$ . Therefore, there are at least  $\max_{i \in [4]} \{|\rho_{u_0} - \rho_{u_i}|\} \cdot |V_{u_1}| \geq \frac{1}{4} \cdot \sum_{i \in [4]} |\rho_{u_i} - \rho_{u_0}| \cdot |V_{u_i}|$   $G_1$  components with at least one vertex in  $S$  and one vertex that is not. The claim follows with  $c_1 = \frac{1}{4d}$ .  $\square$

#### 4.2. Proof of Theorem 1.1

Given the results of the previous section, we first state and prove our main lemma on the edge expansion of the decoding graph of Strassen's CDAG:

LEMMA 4.9. (MAIN LEMMA) *The edge expansion of  $Dec_k C$  is*

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

PROOF. Consider a subset  $S$  of the vertices of the decoding graph. Recall that  $G_k = Dec_k C$  is a layered graph (with layers corresponding to recursion steps), so all edges (excluding loops) connect between consecutive levels of vertices. By Claim 4.6, each level of  $G_k$  contains about the same fraction of  $S$  vertices, or else we have many edges leaving  $S$ . By Fact 4.7, the lowest level is composed of distinct parts that cannot have homogeneity (of the fraction of their  $S$  vertices), otherwise many edges leave  $S$ .

Let  $T_k$  and  $V_u$  be defined as in the previous section. We will show that the homogeneity between levels, combined with the heterogeneity of the lowest level, guarantees that there are many edges leaving  $S$ .

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 4.8, this is

$$\begin{aligned} &\geq \sum_{u \in T_k} c_1 \cdot d \cdot |\rho_u - \rho_{p(u)}| \cdot |V_u| \\ &= c_1 \cdot d \cdot \sum_{i \in [k]} \sum_{u \in t_i} |\rho_u - \rho_{p(u)}| \cdot 7^{i-1} \\ &\geq c_1 \cdot d \cdot \sum_{i \in [k]} \sum_{u \in t_i} |\rho_u - \rho_{p(u)}| \cdot 4^{i-1} \end{aligned}$$

As each internal node has four children, this is

$$= c_1 \cdot d \cdot \sum_{v \in t_1} \sum_{u \in v \sim r} |\rho_u - \rho_{p(u)}|$$

where  $v \sim r$  is the path from  $v$  to the root  $r$ . By the triangle inequality for the function  $|\cdot|$

$$\geq c_1 \cdot d \cdot \sum_{v \in t_1} |\rho_u - \rho_r|$$

By Fact 4.7,

$$\geq c_1 \cdot d \cdot |l_1| \cdot ((1 - \sigma_1) \cdot \rho_r + \sigma_1 \cdot (1 - \rho_r))$$

By Claim 4.6, w.l.o.g.,  $|\sigma_i - \sigma|/\sigma \leq \frac{1}{10}$  (otherwise  $|E(S, V \setminus S)| \geq c_2 \cdot d \cdot |S| \cdot \left(\frac{4}{7}\right)^k$ ), so  $\frac{9}{10}\sigma \leq \sigma_i \leq \frac{11}{10}\sigma$ . As  $\sigma \leq \frac{1}{2}$  and  $\rho_r = \sigma_{k+1}$ ,

$$\geq \frac{81}{100} \cdot c_1 \cdot d \cdot |l_1| \cdot \sigma$$

and by Fact 4.4,

$$\geq c_3 \cdot d \cdot |S| \cdot \left(\frac{4}{7}\right)^k$$

for any  $c_3 \leq \frac{3}{7} \cdot \frac{81}{100} \cdot c_1$ .

Thus, since  $d$  is constant (Fact 4.2), we have  $\frac{|E(S, V \setminus S)|}{|S|} = \Omega\left(\left(\frac{4}{7}\right)^k\right)$ , where the hidden constant is  $c_4 = d \cdot \min\{c_2, c_3\}$ .  $\square$

We now prove the Main Theorem.

**PROOF OF THEOREM 1.1.** Let  $k = \lg \sqrt{M} + c_5$  where  $c_5$  is a constant to be determined, and assume  $k$  divides  $\lg n$  evenly. Note that it is sufficient to prove the result for an infinite number of  $n$ 's, but the smallest  $n$  for which the proof holds is  $n = 2^{c_5} \sqrt{M}$  (so that  $k = \lg n$ ). This assumption implies that  $Dec_{\lg n} C$  is composed of edge-disjoint copies of  $Dec_k C$ , and we can apply Lemma 2.1 with  $G = Dec_{\lg n} C$ ,  $G' = Dec_k C$ , and  $s = |V(G')|/2$ . Since  $d$  and  $d'$  are the same, we have

$$h_s(Dec_{\lg n} C) \geq h(Dec_k C)$$

and by Lemma 4.9 this implies

$$h_s(Dec_{\lg n} C) \geq c_4 \cdot \left(\frac{4}{7}\right)^k.$$

Note that  $7^k/2 \leq s \leq 2 \cdot 7^k$ .

We now apply Lemma 3.2 with  $G$  as the entire CDAG of Strassen's algorithm of matrix dimension  $n$  and  $G' = Dec_{\lg n} C$ . Here  $\alpha = \frac{1}{3}$  and

$$h_s(Dec_{\lg n} C) \cdot \alpha s \geq \frac{c_2}{6} \cdot 4^{c_5} \cdot M \geq 3M$$

for  $c_5 \geq \lg \sqrt{18/c_4}$ , so

$$IO \geq \alpha \cdot \frac{|V|}{s} \cdot M = \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\lg 7} \cdot M \right).$$

The above inequality holds for  $M \leq c_6 \cdot n^2$ , where  $c_6 = 18/c_4 < 1$ . For  $c_6 \cdot n^2 \leq M \leq n^2$ , note that

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

as one has to read  $2n^2$  words of input data and at most  $n^2$  of them can be in the fast memory at the start of the computation.  $\square$

Next we prove the corollary for the parallel case.

**PROOF OF COROLLARY 1.2.** In the parallel case, we follow the reduction approach of [Irony et al. 2004] and consider the busiest processor. Due to averaging, it must do at least  $(1/p)^{\text{th}}$  of the work. We apply the same partitioning argument as in the proof of Theorem 1.1 to that processor's subset of computation. However, in order for the proof to work we must require  $M \leq c \frac{n^2}{p^{2/\lg 7}}$  for some constant  $c$  (rather than  $M \leq cn^2$  in the sequential case).  $\square$

## 5. EXTENSIONS

We now discuss extensions of our approach and the applicability to other algorithms, starting with other fast matrix multiplication algorithms.

### 5.1. Strassen-like Algorithms

A *Strassen-like* algorithm has a recursive structure that utilizes a base algorithm: multiplying two  $n_0$ -by- $n_0$  matrices using  $m_0$  scalar multiplications, where  $n_0$  and  $m_0$  are constants. Given two matrices of size  $n$ -by- $n$ , it splits them into  $n_0^2$  blocks (each of size  $\frac{n}{n_0}$ -by- $\frac{n}{n_0}$ ), and works block-wise, according to the base algorithm. Additions (and subtractions) in the base algorithm are interpreted as additions (and subtractions) of blocks. These are performed element-wise. Multiplications in the base algorithm are interpreted as multiplications of blocks. These are performed by recursively calling the algorithm. The arithmetic count of the algorithm is then  $T(n) = m_0 \cdot T\left(\frac{n}{n_0}\right) + O(n^2)$ , so  $T(n) = \Theta(n^{\omega_0})$  where  $\omega_0 = \log_{n_0} m_0$ .

This is the structure of all the fast matrix multiplication algorithms that were obtained since Strassen's [Pan 1980; Bini 1980; Schönhage 1981; Romani 1982; Coppersmith and Winograd 1982; Strassen 1987; Coppersmith and Winograd 1987], (see [Bürgisser et al. 1997] for discussion of these algorithms), as well as [Cohn et al. 2005], where the base algorithm utilizes a novel group-theoretic approach, and the recent breakthroughs [Stothers 2010; Vassilevska-Williams 2011].<sup>11</sup>

<sup>11</sup> In fact, any arithmetic circuit for multiplying fixed-size matrices can be converted into a bilinear circuit of the same or smaller size [Raz 2003]. This can be used to convert any  $O(n^{\omega_0})$  matrix multiplication algorithm into a bilinear matrix multiplication algorithm of running time  $O(n^{\omega_0 + \epsilon})$  for any  $\epsilon > 0$ . Furthermore,

*5.1.1. A critical technical assumption.* For our technique to work, we further demand that the  $Dec_1C$  part of the computation graph is a connected graph, in order to be Strassen-like (this is assumed in the proof of Claim 4.5). Thus the Strassen-like class includes Winograd's variant of Strassen's algorithm [Winograd 1971], which uses 15 additions rather than 18. It also contains the algorithm of Hopcroft and Kerr [Hopcroft and Kerr 1971] and some variants of Bini's algorithm [Bini 1980; Bini et al. 1979] (see [Ballard et al. 2012c] for further discussion<sup>12</sup>). The Strassen-like class does not contain the classical algorithm, where  $Dec_1C$  is composed of four disconnected graphs (corresponding to the four outputs). The assumption of connectivity of  $Dec_1C$  is not required by the geometric embedding approaches [Irony et al. 2004; Ballard et al. 2011d; Ballard et al. 2012a], that is applied to classical matrix multiplication, nor by the dominator approach [Hong and Kung 1981]. A disconnected  $Dec_1C$  graph can be handled when applying our expansion approach to some cases of fast matrix multiplication algorithms, at the cost of obtaining weaker lower bounds [Ballard et al. 2012c].

*5.1.2. The communication costs of Strassen-like algorithms.* To prove Theorem 1.4, which generalizes the I/O-complexity lower bound of Strassen's algorithm (Theorem 1.1) to all Strassen-like algorithms, we note the following: the entire proof of Theorem 1.1, and in particular, the computations in the proof of Lemma 4.9, hold for any Strassen-like algorithm, where we plug in  $n_0^2$  and  $m_0$ , instead of 4 and 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  $Dec_1C$  is connected (this critical technical assumption is used in the proof of Claim 4.5), and to know the sizes  $n_0$  and  $m_0$ . The only nontrivial adjustment is to show the equivalent of Fact 4.2: that the graph  $Dec_{\log n}C$  is of bounded degree. This is given in the following claim:

**CLAIM 5.1.** *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, i.e., the corresponding row-vector of  $A$  and column vector of  $B$ . 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  $B$ .  $\square$

## 5.2. Multiplying Rectangular Matrices

Many fast algorithms have been devised for multiplication of rectangular matrices (see [Ballard et al. 2012c] for detailed list). A fast algorithm for multiplying  $m_0 \times n_0$  and  $n_0 \times p_0$  matrices in  $q < m_0 n_0 p_0$  scalar multiplications can be applied recursively to multiply  $m_0^t \times n_0^t$  and  $n_0^t \times p_0^t$  matrices in  $O(q^t)$  flops. For such algorithms, the CDAG has very similar structure to Strassen and Strassen-like algorithms for square multiplication in that it is composed of two encoding graphs and one decoding graph. Assuming that the decoding graph is connected, the proofs of Theorem 1.1 and Lemma 4.9 apply where we plug in  $m_0 p_0$  and  $q$  for 4 and 7. In this case, we obtain a result analogous to Theorem 1.1 which states that the I/O-complexity of such an algorithm is given by

the algorithm can be made numerically stable while preserving this form [Demmel, Dumitriu, Holtz, and Kleinberg, 2007].

<sup>12</sup>In particular, note that the observation that some variants of Bini's algorithm have disconnected decoding graphs falsifies the conjecture made in the short version of this paper [Ballard et al. 2011d].

$$\Omega\left(\frac{q^t}{M^{\log_{m_0 p_0} q-1}}\right).$$

If the output matrix is the largest of the three matrices (*i.e.*,  $n_0 < m_0$  and  $n_0 < p_0$ ), then this lower bound is attained by the natural recursive algorithm and is therefore tight. The lower bound extends to the parallel case as well, analogous to Corollary 1.2, and can be attained using the algorithmic technique of [Ballard et al. 2012b].

However, in the case that the decoding graph is not connected, the proof does not apply, and in the case the output matrix is not the largest, the lower bound is not tight. In order to handle these technical challenges, we can employ modifications of the proof technique: we can deal with the decoding graph being disconnected by considering individual connected components, or we can consider one of the two encoding graphs, which may contain vertices of high degree. In either case, the proofs must be adapted, and we obtain slightly weaker results. We detail these approaches in [Ballard et al. 2012c] and discuss the application to rectangular matrix multiplication algorithms of [Bini et al. 1979] and [Hopcroft and Kerr 1971].

### 5.3. Memory-independent lower bounds

Some parallel algorithms require very little (up to a constant factor) extra memory beyond what is necessary to keep the input and output. These are sometimes called *linear space algorithms*. One class of such algorithms are the “2D” algorithms for classical matrix multiplication, that use a two-dimensional grid of processors. Here we allow  $M = \Theta\left(\frac{n^2}{p}\right)$  local memory use (recall that  $p$  is the number of processors, and  $n$  the dimension of the matrices), thus no more than a constant factor of replication of the input matrices is allowed (see [Cannon 1969] for an example of a 2D algorithm). In this case, the classical parallel lower bound reduces to  $\Omega\left(\frac{n^2}{\sqrt{p}}\right)$  [Irony et al. 2004], and this bound is attainable. Similarly, in the case of Strassen-like matrix multiplication, assuming  $M = \Theta\left(\frac{n^2}{p}\right)$ , Corollary 1.2 reduces to  $\Omega\left(\frac{n^2}{p^{2/\omega_0}}\right)$ .

However, as pointed out in [Irony et al. 2004], since  $M$  appears in the denominator, the lower bounds indicate that using more local memory in the parallel case can reduce communication costs. Indeed, so-called “3D” algorithms for classical matrix multiplication (see [Dekel et al. 1981; Aggarwal et al. 1990; Aggarwal et al. 1995]) reduce the communication cost by a factor of  $p^{1/6}$  using a factor of  $p^{1/3}$  more local memory, compared to 2D algorithms. Depending on the problem size and the physical local memory size, this extra memory may not be available, in which case 3D algorithms cannot be used. It is possible to interpolate between 2D and 3D algorithms with a parametrized algorithm which trades off memory for communication and obtain a communication optimal implementation for classical matrix multiplication, for local memory size  $M = \Theta\left(c \cdot \frac{n^2}{p}\right)$  for any  $1 \leq c \leq p^{1/3}$  [McColl and Tiskin 1999; Solomonik and Demmel 2011].

The lower bound arguments of [Irony et al. 2004; Ballard et al. 2011c] for classical algorithms and those proved in this paper for Strassen-like algorithms can be extended to prove memory-independent lower bounds [Ballard et al. 2012a]. As stated in Theorem 1.3, under certain assumptions, any parallel algorithm for classical matrix multiplication must move  $\Omega\left(\frac{n^2}{p^{2/3}}\right)$  words, and any parallel algorithm for Strassen-like matrix multiplication must move  $\Omega\left(\frac{n^2}{p^{2/\omega_0}}\right)$  words.

These bounds dominate the memory-dependent bounds once the local memory size is sufficiently large. In particular, the memory-dependent and memory independent

Table I. Communication-cost lower bounds for parallel matrix multiplication and perfect strong scaling ranges.  $n$  is matrix dimension,  $M$  is local memory size,  $p$  is the number of processors,  $\omega_0$  is the exponent of the arithmetic count.

	Classical	Strassen-like
Memory-dependent lower bound	$\Omega\left(\frac{n^3}{p\sqrt{M}}\right)$	$\Omega\left(\frac{n^{\omega_0}}{pM^{\omega_0/2-1}}\right)$
Memory-independent lower bound	$\Omega\left(\frac{n^2}{p^{2/3}}\right)$	$\Omega\left(\frac{n^2}{p^{2/\omega_0}}\right)$
Perfect strong scaling range	$p = O\left(\frac{n^3}{M^{3/2}}\right)$	$p = O\left(\frac{n^{\omega_0}}{M^{\omega_0/2}}\right)$
Attaining algorithm	[Solomonik and Demmel 2011]	[Ballard et al. 2012b]

bounds coincide when  $M = \Theta\left(\frac{n^2}{p^{2/\omega_0}}\right)$ . Further, the memory-independent bounds imply that there are strict limits on the perfect strong scaling range of matrix multiplication algorithms (both classical and Strassen-like). That is, within the *perfect strong-scaling range*, for a fixed problem size, by doubling the number of processors (and therefore doubling the total memory available) both the computational and communication costs are halved. Beyond the perfect strong-scaling range, the reduction in computational cost is linear, but the reduction in communication cost is sub-linear. These results are summarized in Table I.

The recent parallel algorithm in [Ballard et al. 2012b] attains the memory-dependent bound within the perfect strong-scaling range and attains the memory-independent bound outside the range and is therefore communication-optimal in all cases.

#### 5.4. Other Linear Algebra Problems

Fast matrix multiplication algorithms are basic building blocks in many fast algorithms in linear algebra, such as algorithms for LU, QR, and eigenvalue and singular value decompositions [Demmel et al. 2007]. Therefore, I/O-complexity lower bounds for these algorithms can be derived from our lower bounds for fast matrix multiplication algorithms [Ballard et al. 2012d]. For example, a lower bound on LU (or QR, etc.) follows when the fast matrix multiplication algorithm is called by the LU algorithm on sufficiently large submatrices. This is the case in the algorithms of [Demmel et al. 2007], and we can then deduce matching lower and upper bounds [Ballard et al. 2012d].

## 6. 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. Recently these bounds were attained by new parallel algorithms [Ballard et al. 2012b].

### 6.1. Recursive Implementations

In some cases, the simplest recursive implementation of an algorithm turns out to be communication-optimal in the sequential model (e.g., in the cases of matrix multiplication [Frigo et al. 1999] and Cholesky decomposition [Ahmed and Pingali 2000; Ballard et al. 2010], but not in the case of LU decomposition: the recursive algorithm of [Toledo 1997] is bandwidth optimal but not latency optimal).

In the context of parallel computation, recursive algorithms can again be used for communication-efficiency. See [Tiskin 2002] for LU decomposition and [Tiskin 2007]

for QR decomposition, in which cases a tradeoff is observed between the bandwidth and latency costs of the algorithm.

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.

## 6.2. Other Models

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 (see the MAGMA project [Baboulin et al. 2012], and also [Volkov and Demmel 2008] for using GPU assisted linear algebra computation). A first step in analyzing such systems has been recently introduced in [Ballard et al. 2011b], where the authors modeled heterogeneous shared memory architectures, such as mixed GPU/CPU architecture, and obtained tight lower and upper bounds for classical matrix multiplication.

Note that we can similarly generalize Corollaries 1.2 and 1.5 to other models, such as the heterogeneous model and shared memory model. The reduction is achieved by observing the communication of a single processor.

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.

## ACKNOWLEDGMENTS

We thank Eran Rom, Edgar Solomonik, Sivan Toledo, and Chris Umans for helpful discussions. We would also like to thank the anonymous reviewers for their helpful comments and in simplifying and improving the proofs.

## REFERENCES

- AGARWAL, R. C., BALLE, S. M., GUSTAVSON, F. G., JOSHI, M., AND PALKAR, P. 1995. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development* 39, 39–5.
- AGGARWAL, A., CHANDRA, A. K., AND SNIR, M. 1990. Communication complexity of PRAMs. *Theor. Comput. Sci.* 71, 3–28.
- AGGARWAL, A. AND VITTER, J. S. 1988. The input/output complexity of sorting and related problems. *Commun. ACM* 31, 9, 1116–1127.
- AHMED, N. AND PINGALI, K. 2000. Automatic generation of block-recursive codes. In *Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*. Springer-Verlag, London, UK, 368–378.
- ALON, N., SCHWARTZ, O., AND SHAPIRA, A. 2008. An elementary construction of constant-degree expanders. *Combinatorics, Probability & Computing* 17, 3, 319–327.
- ANDERSON, E., BAI, Z., BISCHOF, C., DEMMEL, J., DONGARRA, J., CROZ, J. D., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., OSTROUCHOV, S., AND SORENSEN, D. 1992. *LAPACK's user's guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. Also available from <http://www.netlib.org/lapack/>.

- BABOULIN, M., DEMMEL, J., DONG, T., DONGARRA, J., HORTON, M., JIA, Y., KABIR, K., LANGOU, J., LI, Y., LTAIEF, H., NATH, R., TOMOV, S., AND VOLKOV, V. 2008–2012. The MAGMA project. University of Tennessee, Department of Electrical Engineering and Computer Science. <http://icl.cs.utk.edu/magma/>.
- BALLARD, G., DEMMEL, J., AND DUMITRIU, I. 2011a. Communication-optimal parallel and sequential eigenvalue and singular value algorithms. EECSS Technical Report EECSS-2011-14, UC Berkeley. Feb.
- BALLARD, G., DEMMEL, J., AND GEARHART, A. 2011b. Brief announcement: communication bounds for heterogeneous architectures. In *SPAA '11: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, New York, NY, USA, 257–258.
- BALLARD, G., DEMMEL, J., HOLTZ, O., LIPSHITZ, B., AND SCHWARTZ, O. 2012a. Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '12. ACM, New York, NY, USA, 77–79.
- BALLARD, G., DEMMEL, J., HOLTZ, O., LIPSHITZ, B., AND SCHWARTZ, O. 2012b. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '12. ACM, New York, NY, USA, 193–204.
- BALLARD, G., DEMMEL, J., HOLTZ, O., LIPSHITZ, B., AND SCHWARTZ, O. 2012c. Graph expansion analysis for communication costs of fast rectangular matrix multiplication. Tech. Rep. EECSS-2012-194, UC Berkeley. March.
- BALLARD, G., DEMMEL, J., HOLTZ, O., AND SCHWARTZ, O. 2009. Communication-optimal Parallel and Sequential Cholesky Decomposition. In *SPAA '09: Proceedings of the 21st Annual ACM Symposium on Parallel Algorithms and Architectures*. ACM, New York, NY, USA, 245–252.
- BALLARD, G., DEMMEL, J., HOLTZ, O., AND SCHWARTZ, O. 2010. Communication-optimal parallel and sequential Cholesky decomposition. *SIAM Journal on Scientific Computing* 32, 6, 3495–3523.
- BALLARD, G., DEMMEL, J., HOLTZ, O., AND SCHWARTZ, O. 2011c. Graph Expansion and Communication Costs of Fast Matrix Multiplication. In *SPAA '11: Proceedings of the 23rd Annual ACM Symposium on Parallel Algorithms and Architectures*. ACM, New York, NY, USA, 1–12.
- BALLARD, G., DEMMEL, J., HOLTZ, O., AND SCHWARTZ, O. 2011d. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications* 32, 3, 866–901.
- BALLARD, G., DEMMEL, J., HOLTZ, O., AND SCHWARTZ, O. 2012d. Sequential communication bounds for fast linear algebra. Tech. Rep. EECSS-2012-36, UC Berkeley. March.
- BENDER, M. A., BRODAL, G. S., FAGERBERG, R., JACOB, R., AND VICARI, E. 2010. Optimal sparse matrix dense vector multiplication in the I/O-model. *Theory of Computing Systems, Special Issue of SPAA '07* 47, 4, 934–962.
- BILARDI, G., PIETRACAPRINA, A., AND D'ALBERTO, P. 2000. 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*. Springer-Verlag, London, UK, 47–58.
- BILARDI, G. AND PREPARATA, F. 1999. Processor-time tradeoffs under bounded-speed message propagation: Part II, lower bounds. *Theory of Computing Systems* 32, 5, 1432–1435.
- BINI, D. 1980. Relations between exact and approximate bilinear algorithms. applications. *Calcolo* 17, 87–97. 10.1007/BF02575865.
- BINI, D., CAPOVANI, M., ROMANI, F., AND LOTTI, G. 1979.  $O(n^{2.7799})$  complexity for  $n \times n$  approximate matrix multiplication. *Information Processing Letters* 8, 5, 234 – 235.
- BLACKFORD, L. S., CHOI, J., CLEARY, A., DAZEVEDO, E., DEMMEL, J., DHILLON, I., DONGARRA, J., HAMMARLING, S., HENRY, G., PETITET, A., STANLEY, K., WALKER, D., AND WHALEY, R. C. 1997. *ScalAPACK Users' Guide*. SIAM, Philadelphia, PA, USA. Also available from <http://www.netlib.org/scalapack/>.
- BLELLOCH, G. E., CHOWDHURY, R. A., GIBBONS, P. B., RAMACHANDRAN, V., CHEN, S., AND KOZUCH, M. 2008. Provably good multicore cache performance for divide-and-conquer algorithms. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 501–510.
- BURAGO, Y. D. AND ZALGALLER, V. A. 1988. *Geometric Inequalities*. Grundlehren der Mathematische Wissenschaften Series, vol. 285. Springer, Berlin.
- BÜRGISSER, P., CLAUSEN, M., AND SHOKROLLAHI, M. A. 1997. *Algebraic Complexity Theory*. Number 315 in Grundlehren der mathematischen Wissenschaften. Springer Verlag.
- CANNON, L. 1969. A cellular computer to implement the Kalman filter algorithm. Ph.D. thesis, Montana State University, Bozeman, MN.
- CHOWDHURY, R. A. AND RAMACHANDRAN, V. 2006. Cache-oblivious dynamic programming. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. ACM, New York, NY, USA, 591–600.

- COHN, H., KLEINBERG, R. D., SZEGEDY, B., AND UMANS, C. 2005. Group-theoretic algorithms for matrix multiplication. In *FOCS*. 379–388.
- COPPERSMITH, D. AND WINOGRAD, S. 1982. On the asymptotic complexity of matrix multiplication. *SIAM Journal on Computing* 11, 3, 472–492.
- COPPERSMITH, D. AND WINOGRAD, S. 1987. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. STOC '87. ACM, New York, NY, USA, 1–6.
- COPPERSMITH, D. AND WINOGRAD, S. 1990. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.* 9, 3, 251–280.
- DAVID, P.-Y., DEMMEL, J., GRIGORI, L., AND PEYRONNET, S. 2010. Brief announcement: Lower bounds on communication for sparse Cholesky factorization of a model problem. In *SPAA '10: Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures*.
- DEKEL, E., NASSIMI, D., AND SAHNI, S. 1981. Parallel matrix and graph algorithms. *SIAM J. Comput.*, 657–675.
- DEMMEL, J., DUMITRIU, I., AND HOLTZ, O. 2007. Fast linear algebra is stable. *Numerische Mathematik* 108, 1, 59–91.
- DEMMEL, J., GRIGORI, L., HOEMMEN, M., AND LANGOU, J. 2012. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM Journal on Scientific Computing* 34, 1, A206–A239.
- DESPREZ, F. AND SUTER, F. 2004. 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.
- DOUGLAS, C. C., HEROUX, M., SLISHMAN, G., AND SMITH, R. M. 1994. GEMMW: A portable level 3 BLAS Winograd variant of Strassen's matrix-matrix multiply algorithm. *Journal of Computational Physics* 110, 1, 1–10.
- ELMROTH, E. AND GUSTAVSON, F. 1998. New serial and parallel recursive QR factorization algorithms for SMP systems. In *Applied Parallel Computing. Large Scale Scientific and Industrial Problems.*, B. K. et al., Ed. Lecture Notes in Computer Science Series, vol. 1541. Springer, 120–128.
- FRENS, J. D. AND WISE, D. S. 2003. QR factorization with Morton-ordered quadtree matrices for memory re-use and parallelism. *SIGPLAN Not.* 38, 10, 144–154.
- FRIGO, M., LEISERSON, C. E., PROKOP, H., AND RAMACHANDRAN, S. 1999. Cache-oblivious algorithms. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Washington, DC, USA, 285.
- FULLER, S. H. AND MILLETT, L. I., Eds. 2011. *The Future of Computing Performance: Game Over or Next Level?* The National Academies Press, Washington, D.C. 200 pages, <http://www.nap.edu>.
- GRAHAM, S. L., SNIR, M., AND PATTERSON, C. A., Eds. 2004. *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. 289 pages, <http://www.nap.edu>.
- GRIGORI, L., DEMMEL, J., AND XIANG, H. 2008. Communication-avoiding Gaussian elimination. *Supercomputing 08*.
- GRIGORI, L., DEMMEL, J., AND XIANG, H. 2011. CALU: A communication optimal LU factorization algorithm. *SIAM Journal on Matrix Analysis and Applications* 32, 4, 1317–1350.
- GUSTAVSON, F. G. 1997. Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM J. Res. Dev.* 41, 6, 737–756.
- HONG, J. W. AND KUNG, H. T. 1981. I/O complexity: The red-blue pebble game. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*. ACM, New York, NY, USA, 326–333.
- HOPCROFT, J. E. AND KERR, L. R. 1971. On minimizing the number of multiplications necessary for matrix multiplication. *SIAM Journal on Applied Mathematics* 20, 1, pp. 30–36.
- HUSS-LEDERMAN, S., JACOBSON, E. M., JOHNSON, J. R., TSAO, A., AND TURNBULL, T. 1996. Implementation of Strassen's algorithm for matrix multiplication. In *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*. IEEE Computer Society, Washington, DC, USA, 32.
- IRONY, D., TOLEDO, S., AND TISKIN, A. 2004. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.* 64, 9, 1017–1026.
- KOUCKY, M., KABANETS, V., AND KOLOKOLOVA, A. 2010. Expanders made elementary. In preparation, Available from <http://www.cs.sfu.ca/~kabanets/papers/expanders.pdf>.
- LEISERSON, C. E. 2008. Personal communication with G. Ballard, J. Demmel, O. Holtz, and O. Schwartz.
- LEV, G. AND VALIANT, L. G. 1983. Size bounds for superconcentrators. *Theoretical Computer Science* 22, 3, 233–251.

- LOOMIS, L. H. AND WHITNEY, H. 1949. An inequality related to the isoperimetric inequality. *Bulletin of the AMS* 55, 961–962.
- MCCOLL, W. F. AND TISKIN, A. 1999. Memory-efficient matrix multiplication in the BSP model. *Algorithmica* 24, 287–297. 10.1007/PL00008264.
- MICHAEL, J. P., PENNER, M., AND PRASANNA, V. K. 2002. Optimizing graph algorithms for improved cache performance. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2002), Fort Lauderdale, FL*. 769–782.
- MIHAIL, M. 1989. Conductance and convergence of Markov chains: A combinatorial treatment of expanders. In *Proceedings of the Thirtieth Annual IEEE Symposium on Foundations of Computer Science*. 526–531.
- PAN, V. Y. 1980. New fast algorithms for matrix operations. *SIAM Journal on Computing* 9, 2, 321–342.
- RAZ, R. 2003. On the complexity of matrix product. *SIAM J. Comput.* 32, 5, 1356–1369 (electronic).
- REINGOLD, O., VADHAN, S., AND WIGDERSON, A. 2002. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics* 155, 1, 157–187.
- ROMANI, F. 1982. Some properties of disjoint sums of tensors related to matrix multiplication. *SIAM Journal on Computing* 11, 2, 263–267.
- SAVAGE, J. 1994. Space-time tradeoffs in memory hierarchies. Tech. rep., Brown University, Providence, RI, USA.
- SAVAGE, J. E. 1995. Extending the Hong-Kung model to memory hierarchies. In *COCOON*. 270–281.
- SCHÖNHAGE, A. 1981. Partial and total matrix multiplication. *SIAM Journal on Computing* 10, 3, 434–455.
- SOLOMONIK, E. AND DEMMEL, J. 2011. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *Euro-Par'11: Proceedings of the 17th International European Conference on Parallel and Distributed Computing*. Springer.
- STOTHERS, A. J. 2010. On the complexity of matrix multiplication. Ph.D. thesis, University of Edinburgh.
- STRASSEN, V. 1969. Gaussian elimination is not optimal. *Numer. Math.* 13, 354–356.
- STRASSEN, V. 1987. Relative bilinear complexity and matrix multiplication. *Journal für die reine und angewandte Mathematik (Crelles Journal)* 1987, 375–376, 406–443.
- TISKIN, A. 2002. Bulk-synchronous parallel Gaussian elimination. *Journal of Mathematical Sciences* 108, 977–991. 10.1023/A:1013588221172.
- TISKIN, A. 2007. Communication-efficient parallel generic pairwise elimination. *Future Generation Computer Systems* 23, 2, 179 – 188.
- TOLEDO, S. 1997. Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl.* 18, 4, 1065–1081.
- VASSILEVSKA-WILLIAMS, V. 2011. Breaking the Coppersmith-Winograd barrier. Manuscript. <http://www.cs.berkeley.edu/~virgi/matrixmult.pdf>.
- VOLKOV, V. AND DEMMEL, J. 2008. Benchmarking GPUs to tune dense linear algebra. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, Piscataway, NJ, USA, 1–11.
- WINOGRAD, S. 1971. On the multiplication of  $2 \times 2$  matrices. *Linear Algebra Appl.* 4, 4, 381–388.
- WISE, D. S. 2000. Ahnentafel indexing into Morton-ordered arrays, or matrix locality for free. In *Proceedings from the 6th International Euro-Par Conference on Parallel Processing*. Euro-Par '00. Springer-Verlag, London, UK, UK, 774–783.
- YANG, C.-Q. AND MILLER, B. 1988. Critical path analysis for the execution of parallel and distributed programs. In *Proceedings of the 8th International Conference on Distributed Computing Systems*. 366–373.

**A. STRASSEN'S FAST MATRIX MULTIPLICATION ALGORITHM**

Strassen's original algorithm follows [Strassen 1969]. See [Winograd 1971] for Winograd's variant, which reduces the number of additions. For actual uses of Strassen's algorithm, see [Douglas et al. 1994; Huss-Lederman et al. 1996; Desprez and Suter 2004].

**Algorithm 1** Matrix Multiplication: Strassen's Algorithm

---

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

```

1: if  $n = 1$  then
2:    $C_{11} = A_{11} \cdot B_{11}$ 
3: else
4:   {Decompose  $A$  into four equal square blocks  $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ 
      and the same for  $B$ .}
5:    $M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$ 
6:    $M_2 = (A_{21} + A_{22}) \cdot B_{11}$ 
7:    $M_3 = A_{11} \cdot (B_{12} - B_{22})$ 
8:    $M_4 = A_{22} \cdot (B_{21} - B_{11})$ 
9:    $M_5 = (A_{11} + A_{12}) \cdot B_{22}$ 
10:   $M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$ 
11:   $M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$ 
12:   $C_{11} = M_1 + M_4 - M_5 + M_7$ 
13:   $C_{12} = M_3 + M_5$ 
14:   $C_{21} = M_2 + M_4$ 
15:   $C_{22} = M_1 - M_2 + M_3 + M_6$ 
16: end if
17: return  $C$ 

```

---