Communication Costs of Strassen's Matrix Multiplication

Grey Ballard
UC Berkeley
ballard@cs.berkeley.edu

James Demmel
UC Berkeley
demmel@cs.berkeley.edu

Olga Holtz
UC Berkeley and TU Berlin
holtz@math.berkeley.edu

Oded Schwartz
UC Berkeley
odedsc@cs.berkeley.edu

ABSTRACT

Algorithms have historically been evaluated in terms of the number of arithmetic operations they performed. This analysis is no longer sufficient for predicting running times on today's machines. Moving data through memory hierarchies and among processors requires much more time (and energy) than performing computations. Hardware trends suggest the relative costs of this communication will only increase. Proving lower bounds on the communication of algorithms and finding algorithms that attain these bounds are therefore fundamental goals. We show that the communication cost of an algorithm is closely related to the graph expansion properties of its corresponding computation graph.

Matrix multiplication is one of the most fundamental problems in scientific computing and in parallel computing. Applying expansion analysis to Strassen's and other fast matrix multiplication algorithms, we obtain the first lower bounds on their communication costs. These bounds show that the current sequential algorithms are optimal but that previous parallel algorithms communicate more than necessary. Our new parallelization of Strassen's algorithm is communication-optimal and outperforms all previous matrix multiplication algorithms.

1. INTRODUCTION

Communication (*i.e.*, moving data) can greatly dominate the cost of an algorithm, whether the cost is measured in running time or in total energy. This holds for moving data between levels of a memory hierarchy or between processors over a network. 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. In fact, technological trends [16, 17] are making communication costs grow exponentially over time compared to arithmetic costs. Moore's Law is making arithmetic on a chip improve at about 60% per year, but memory and network bandwidth is improving at only 26% and 23% per year [16]. So even in cases where communication is not the bottleneck today, it may be in the future.

Ideally, we would be able to determine lower bounds on the amount of required communication for important problems and design algorithms that attain them, namely algorithms that are communication-optimal. These dual problems have long attracted researchers, with one example being classical $\Theta(n^3)$ matrix multiplication (see further details below), with lower bounds in [18, 20] and many optimal sequential and parallel algorithms, e.g., [1, 11].

These lower bounds have recently been extended to a large class of other classical linear algebra problems, including linear system solving, least squares, and eigenvalue problems, for dense and sparse matrices, and for sequential and parallel machines [9]. Surprisingly, the highly optimized algorithms in widely implemented libraries like LAPACK and ScaLAPACK [3] often do not attain these lower bounds, even in the asymptotic sense. This has led to much recent work inventing new, faster algorithms that do; see the citations in [9, 10] for references.

In this paper we describe a novel approach to prove the first communication lower bounds for Strassen's $\Theta(n^{\log_2 7})$ matrix multiplication algorithm, as well as many similar fast algorithms. Specifically, we introduce expansion analysis of the computational graphs of the algorithms and show that the expansion helps determine the communication cost. These communication cost bounds are *lower* than those of classical matrix multiplication: this means that not only does Strassen's algorithm reduce computation, it also creates an opportunity for reducing communication. In addition, the lower bound decreases as the amount of available memory grows, suggesting that using extra memory may also allow for faster algorithms.

In fact there is an optimal parallel algorithm that attains our lower bounds for varying amounts of memory, whose performance exceeds all other known matrix multiplication implementations, classical or Strassen-based, on a large parallel machine [6], see Figure 1. In the rest of this paper we focus on explaining our new lower bounds for Strassen's algorithm and their implications.

1.1 Communication Models

In order to analyze the communication costs of algorithms we consider idealized memory and communication models. In the sequential case (see Figure 2), we consider a machine with two levels of memory hierarchy: a fast memory

The original version of this paper is entitled "Graph Expansion and Communication Costs of Fast Matrix Multiplication" and was first published in the *Proceedings of the 2011 ACM Symposium on Parallelism in Algorithms and Architectures* and also appeared in the December 2012 issue of the *Journal of the ACM.*

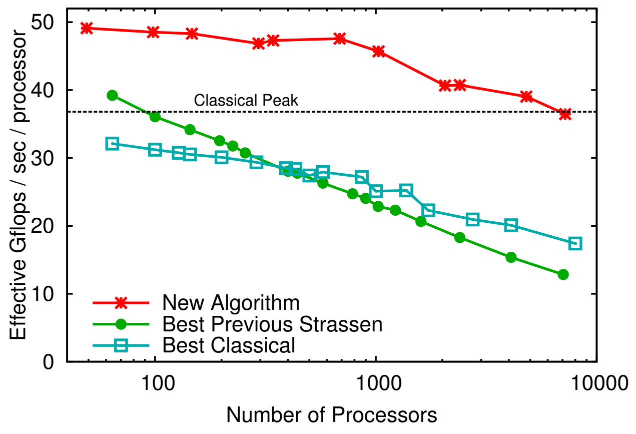

Figure 1: Strong-scaling performance comparison of parallel matrix multiplication algorithms on a Cray XT4. All data corresponds to a fixed dimension $n = 94080$. The x-axis represents the number of processors p on a log scale, and the y-axis measures effective performance, or $2n^3/(p \cdot \text{time})$. The new algorithm outperforms all other known algorithms and exceeds the peak performance of the machine with respect to the classical flop count. The new algorithm runs 24-184% faster than the best previous Strassen-based algorithm and 51-84% faster than the best classical algorithm for this problem size.

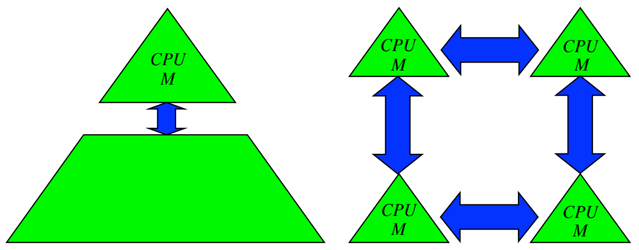

Figure 2: Sequential two-level (left) and parallel distributed-memory (right) models.

of size M words (where computation is performed) and a slow memory of infinite size. We assume that the input initially resides in slow memory and is too large to fit in fast memory. We define the *communication cost* of a sequential algorithm to be the total number of words transferred between the slow and fast memories.

In the parallel case (see Figure 2), we consider p processors, each with a local memory of size M, connected over a network. In this case, the communication cost is the number of words transferred between processors, counted along the critical path of the algorithm. That is, two words that are communicated simultaneously between separate pairs of processors are counted only once.

1.2 Classical Matrix Multiplication

To illustrate the effects of arithmetic reordering on communication and running time of a sequential computation, consider the problem of computing matrix multiplication $C = A \cdot B$, where the $(i, j)^{\text{th}}$ output element is computed by the classical formula $C_{ij} = \sum_k A_{ik} \cdot B_{kj}$. One "naive" ordering of the computation of the classical algorithm can be specified simply by three nested loops (see Algorithm 1). For matrices that are too large to fit in fast memory, this ordering requires the communication of at least one operand for each scalar multiplication, resulting in a total communication cost of $\Theta(n^3)$. A natural question to ask is: can we do better?

Algorithm 1 Naive Classical Matrix Multiplication

```
1: for $i = 1$ to $n$ do
2:   for $j = 1$ to $n$ do
3:     for $k = 1$ to $n$ do
4:       $C_{ij} = C_{ij} + A_{ik} \cdot B_{kj}$
```

The answer is yes. We can reduce communication by using a "blocked" algorithm (see Algorithm 2). The idea is to partition A, B, and C into square blocks of size $b \times b$ so that three blocks can simultaneously fit in the fast memory. We use the notation $C[I, J]$ to refer to the $(I, J)^{\text{th}}$ $b \times b$ block of the C matrix. When $C[I, J]$, $A[I, K]$, and $B[K, J]$ are all in fast memory, then the inner loop of the algorithm (corresponding to $\Theta(b^3)$ arithmetic operations) can be performed with no more communication.

Algorithm 2 Blocked Classical Matrix Multiplication

```
1: for $I = 0$ to $n/b$ do
2:   for $J = 0$ to $n/b$ do
3:     for $K = 0$ to $n/b$ do
4:       $C[I, J] = C[I, J] + A[I, K] \cdot B[K, J]$
```

If we pick the maximum block size of $b = \sqrt{M/3}$, this results in a total of $\Theta((n/\sqrt{M})^3)$ block operations, each requiring $\Theta(M)$ words to be communicated. Hence the total communication cost is $\Theta(n^3/\sqrt{M})$, a factor of $\Theta(\sqrt{M})$ better than that of the naive algorithm.

The typical performance difference of the naive and blocked algorithms on a sequential machine is an order of magnitude. With the blocked algorithm, attained performance is close to the peak capabilities of the machine. Again, the question arises: can we do better? Can we further reorder these computations to communicate less?

If we insist on performing the $\Theta(n^3)$ arithmetic operations given by the classical formulation, the answer is no. Hong and Kung [18] proved a communication cost lower bound of $\Omega(n^3/\sqrt{M})$ for any reordering, showing that the blocked algorithm is communication-optimal. But this is not the end of the story: this communication optimality of the blocked algorithm assumes $\Theta(n^3)$ arithmetic operations.

1.3 Strassen's Matrix Multiplication

While the classical algorithms for matrix multiplication are already optimized for reducing communication cost to the minimum possible, a completely different algorithmic approach for this problem is possible. Let us recall Strassen's algorithm [24] (see Algorithm 3).

Strassen's key idea is to multiply 2×2 matrices using 7 scalar multiplies instead of 8. Because $n \times n$ matrices can be divided into quadrants, Strassen's idea applies recursively. Each of the seven quadrant multiplications is computed recursively, and the computational cost of additions and subtractions of quadrants is $\Theta(n^2)$. Thus, the recurrence for the flop count is $F(n) = 7F(n/2) + \Theta(n^2)$ with base case $F(1) = 1$, which yields $F(n) = \Theta(n^{\log_2 7})$, which is asymptotically less computation than the classical algorithm.

The main results presented in the following section expose a wonderful fact: not only does Strassen's algorithm require less computation than the classical algorithm, but it also requires less communication!

Algorithm 3 Strassen's Matrix Multiplication Algorithm

```
Input: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ and $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \in \mathbb{R}^{n \times n}$
 1: if $n = 1$ then
 2:     $C = A \cdot B$
 3: else
 4:     $M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$
 5:     $M_2 = (A_{21} + A_{22}) \cdot B_{11}$
 6:     $M_3 = A_{11} \cdot (B_{12} - B_{22})$
 7:     $M_4 = A_{22} \cdot (B_{21} - B_{11})$
 8:     $M_5 = (A_{11} + A_{12}) \cdot B_{22}$
 9:     $M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$
10:     $M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$
11:     $C_{11} = M_1 + M_4 - M_5 + M_7$
12:     $C_{12} = M_3 + M_5$
13:     $C_{21} = M_2 + M_4$
14:     $C_{22} = M_1 - M_2 + M_3 + M_6$
Output: $A \cdot B = C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \in \mathbb{R}^{n \times n}$
```

2. COMMUNICATION LOWER BOUNDS

In this section we state our main results: communication lower bounds for Strassen's matrix multiplication. The proof technique described in Section 3 allows us to state bounds in both sequential and parallel cases. As mentioned in the Introduction, the lower bounds are lower than the bounds for the classical algorithm [18, 20]. In both sequential and parallel cases, there now exist communication-optimal algorithms that achieve the lower bounds.

2.1 Sequential Case

We obtain the following lower bound:

Theorem 1 ([10]). *Consider Strassen's algorithm implemented on a sequential machine with fast memory of size M. Then for $M \le n^2$, the communication cost of Strassen's algorithm is*

$$IO(n, M) = \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \cdot M\right).$$

It holds for any implementation and any known variant of Strassen's algorithm that is based on performing 2×2 matrix multiplication with 7 scalar multiplications. This includes Winograd's $O(n^{\log_2 7})$ variant that uses 15 additions instead of 18, which is the most commonly used fast matrix multiplication algorithm in practice.

This lower bound is tight, in that it is attained by the standard recursive sequential implementation of Strassen's algorithm. The recursive algorithm's communication cost is given by the recurrence $IO(n, M) \le 7 \cdot IO\left(\frac{n}{2}, M\right) + O(n^2)$. The base case occurs when the input and output

	Classical	Strassen
Sequential lower bound [18, 10]	$\left(\frac{n}{\sqrt{M}}\right)^3 M$	$\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M$

Table 1: Asymptotic communication cost lower bounds for sequential matrix multiplication, where n is the matrix dimension and M is the fast memory size. Note that although the expressions for classical and Strassen are similar, the proof techniques are quite different.

sub-matrices fit in the fast memory and the matrix multiplication can be performed with no further communication. This yields

$$IO(n, M) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \cdot M\right)$$

for $M \le n^2$, matching the lower bound stated in Theorem 1.

2.2 Parallel Case

The proof technique of Theorem 1 extends to parallel machines, yielding:

Corollary 2 ([10]). *Consider Strassen's algorithm implemented on a parallel machine with p processors, each with a local memory of size M. Then for $M = O\left(\frac{n^2}{p^{2/\log_2 7}}\right)$, the communication cost of Strassen's algorithm is*

$$IO(n, p, M) = \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \cdot \frac{M}{p}\right).$$

While Corollary 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 [5] and holds for all local memory sizes, though it requires separate assumptions.

Theorem 3 ([5]). *Suppose a parallel algorithm performing Strassen's matrix multiplication load balances the computation. Then, the communication cost is*

$$IO(n, p) = \Omega\left(\frac{n^2}{p^{2/\log_2 7}}\right).$$

Note that the bound in Corollary 2 dominates the one in Theorem 3 for $M = O(n^2/p^{2/\log_2 7})$. Thus, the tightest lower bound for parallel implementations of Strassen is the maximum of these two bounds. Table 2.2 and Figure 3, both adapted from [5], illustrate the relationship between the two functions. Figure 3 in particular shows bounds on strong scaling: for a fixed dimension n, increasing the number of processors (each with local memory size M) within a limited range does not increase the total volume of communication. Thus the communication cost along the critical path decreases linearly with p. This is because in this "perfect strong scaling range" the dominant lower bound includes a p in the denominator; however, when the second bound begins to dominate, the denominator includes a $p^{2/3}$ rather than p, and increasing p leads to more communication volume. As shown in the figure, a similar phenomenon occurs for the classical algorithm, though with slightly different parameters [5, 23].

	Classical	Strassen
Memory-dependent lower bound [20, 10]	$\left(\frac{n}{\sqrt{M}}\right)^3 \frac{M}{p}$	$\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{p}$
Memory-independent lower bound [5]	$\frac{n^2}{p^{2/3}}$	$\frac{n^2}{p^{2/\log_2 7}}$

Table 2: Asymptotic communication cost lower bounds for parallel matrix multiplication, where n is matrix dimension, M is local memory size, and p is the number of processors.

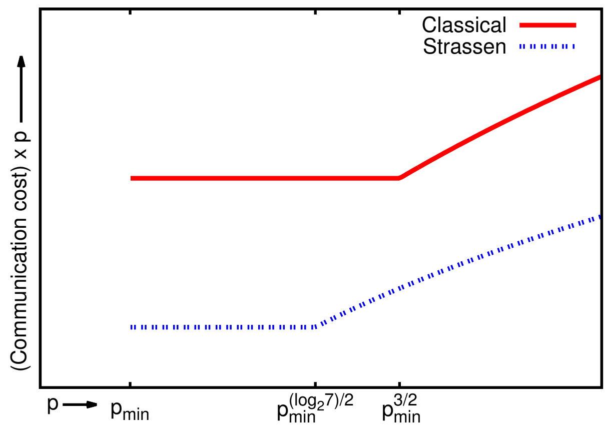

Figure 3: Communication costs and strong scaling of matrix multiplication: classical vs. Strassen. The vertical axis corresponds to p times the communication cost, so horizontal lines correspond to perfect strong scaling. The quantity $p_{\min}$ is the minimum number of processors required to store the input and output matrices (*i.e.*, $p_{\min} = 3n^2/M$ where n is the matrix dimension and M is the local memory size).

The recent parallel algorithm for Strassen's matrix multiplication [6] has communication cost

$$IO(n,p,M) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \cdot \frac{M}{p} + \frac{n^2}{p^{2/\log_2 7}}\right)$$

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 2 and Theorem 3 above. A similar algorithm for Strassen's matrix multiplication in the BSP model is presented in [22].

3. PROOF HIGHLIGHTS

The crux of the proof of Theorem 1 is based on estimating the edge expansion of the computation graph of Strassen's algorithm. We describe below how communication cost is closely related to the edge expansion properties of this graph. The graph has a recursive structure, and we use a combinatorial analysis of the expansion. The high-level argument is based on partitioning the computation in segments, which we explain in Section 3.3. Let us first define two key concepts: computation graphs and edge expansion. See [10] for the full proof.

3.1 Computation Graphs

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, and output argument, and edges according to direct dependencies (e.g., for the binary arithmetic operation $x := y + z$ we have directed edges from vertices corresponding to operands y and z to the vertex corresponding to x).

In the sequential case, an implementation (or scheduling) determines the order of execution of the arithmetic operations, which respects the partial ordering of the CDAG. In the parallel case, an implementation determines which arithmetic operations are performed by which of the p processors as well as the ordering of local operations. 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 and therefore relate to communication.

3.2 Edge Expansion

Expansion is a graph-theoretic concept [19] that relates a given subset of a graph to its boundary. If a graph has large expansion, then subsets of vertices will have relatively large boundaries. For example, a 2D grid where each vertex has north, south, east, and west neighbors has small expansion, whereas a complete graph has large expansion. While there are several variants of expansion metrics, we are interested in edge expansion of regular graphs, defined as follows: the edge expansion $h(G)$ of a d-regular undirected graph $G = (V,E)$ is:

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

where $E_G(A,B)$ is the set of edges connecting the disjoint vertex sets A and B.

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'.

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

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

For many interesting graph families (including Strassen's CDAG), $h_s(G)$ does not depend on $|V(G)|$ when s is fixed, although it may decrease when s increases.

3.3 The Partition Argument

The high-level lower bound argument is based on partitioning the execution of an algorithm's implementation into segments. 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

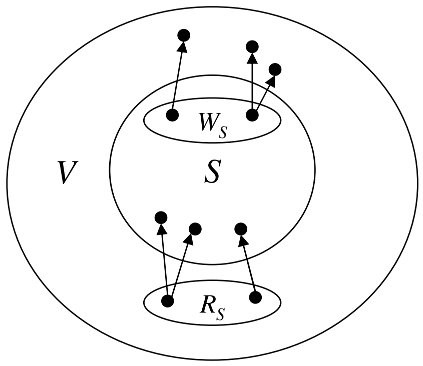

Figure 4: A subset (segment) S and its corresponding read operands R_S, and write operands W_S.

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 S be some segment, and define R_S and W_S to be the set of read and write operands, respectively (see Figure 4). 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. Recall that M is the size of the fast memory. Then the total communication cost due to reads of operands 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 segment 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 segment ends. The total communication cost is therefore bounded below by

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

3.4 Edge Expansion and Communication

Consider a segment S and its read and write operands R_S and W_S (see Figure 4). If the graph G containing S has $h(G)$ edge expansion, maximum degree d and at least $2|S|$ vertices, then (using the definition of $h(G)$), we have

Claim 4. $|R_S| + |W_S| \geq h(G) \cdot |S|$.

Combining this with (3) and choosing to partition V into $|V|/s$ segments of equal size s, we obtain: $IO \geq \max_s(|V|/s) \cdot (h(G) \cdot s - 2M) = \Omega(|V| \cdot h(G))$. In many cases $h(G)$ is too small to attain the desired communication cost lower bound. Typically, $h(G)$ is a decreasing function of $|V(G)|$; that is, the edge expansion deteriorates with the increase of the input size and number of arithmetic operations of the corresponding algorithm (this is the case with Strassen's algorithm). In such cases, it is better to consider the expansion of G on small sets only: $IO \geq \max_s(|V|/s) \cdot (h_s(G) \cdot s - 2M)$. Choosing the minimal s so that

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

we obtain

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

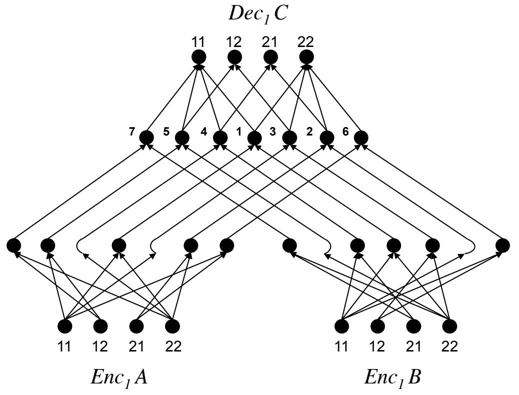

Figure 5: Computation graph of Strassen's algorithm for multiplying 2×2 matrices (H_1). The encodings of A and B correspond to the additions and subtractions in lines 4-10 of Algorithm 3, and the decoding of the seven multiplications to compute C correspond to lines 11-14. A vertex labeled with two indices ij corresponds to the $(i,j)^{\text{th}}$ entry of a matrix and a vertex labeled with one index k corresponds to the k^{th} intermediate multiplication.

The existence of a value $s \leq |V|/2$ that satisfies condition (4) is not always guaranteed. In [10] we confirm the existence of such s for Strassen's CDAG for sufficiently large $|V|$.

4. STRASSEN'S CDAG

Recall Strassen's algorithm for matrix multiplication and consider its computation graph. If we let H_i be the computation graph of Strassen's algorithm for recursion of depth i, then $H_{\log_2 n}$ corresponds to the computation for input matrices of size $n \times n$. Let us first consider H_1 as shown in Figure 5, which corresponds to multiplying 2×2 matrices. Each of A and B are "encoded" into seven pairs of multiplication inputs, and vertices corresponding to the outputs of the multiplications are then "decoded" to compute the output matrix C.

The general computation graph $H_{\log_2 n}$ has similar structure:

- Encode A: generate weighted sums of elements of A
- Encode B: generate weighted sums of elements of B
- Multiply the encodings of A and B element-wise
- Decode C: take weighted sums of the products

Denote by $Enc_{\log_2 n}A$ the part of $H_{\log_2 n}$ that corresponds to the encoding of matrix A. Similarly, $Enc_{\log_2 n}B$, and $Dec_{\log_2 n}C$ correspond to the parts of $H_{\log_2 n}$ that compute the encoding of B and the decoding of C, respectively. Figure 6 shows a high level picture of $H_{\log_2 n}$. In the next section we provide a more detailed description of the CDAG.

4.1 Recursive Construction

We construct the computation graph H_{i+1} by constructing $Dec_{i+1}C$ from Dec_iC and Dec_1C, similarly constructing $Enc_{i+1}A$ and $Enc_{i+1}B$, and then composing the three parts

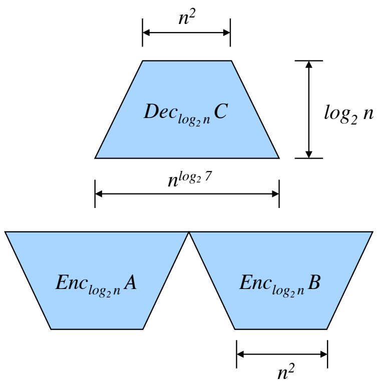

Figure 6: High-level view of Strassen's CDAG for $n \times n$ matrices. The graph is composed of two encoding subgraphs and one decoding subgraph; connections between the subgraphs are not shown.

together. Here is the main idea for recursively constructing $Dec_{i+1}C$ which is illustrated by Figure 7.

- Replicate Dec_1C 7^i times.
- Replicate Dec_iC 4 times.
- 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 set of each first output vertex of the 7^i Dec_1C graphs is identified with the set of 7^i input vertices of the first copy of Dec_iC.
  - The set of each second output vertex of the 7^i Dec_1C graphs is identified with the set of 7^i input vertices of the second copy of Dec_iC. And so on.
  - We make sure that the j^{th} input vertex of a copy of Dec_iC is identified with an output vertex of the j^{th} copy of Dec_1C.

After constructing $Enc_{i+1}A$ and $Enc_{i+1}B$ in a similar manner, we obtain H_{i+1} by connecting edges from the k^{th} output vertices of $Enc_{i+1}A$ and $Enc_{i+1}B$ to the k^{th} input vertex of $Dec_{i+1}C$, which corresponds to the element-wise scalar multiplications.

4.2 Strassen's Edge Expansion

Given the construction of the CDAG for Strassen's algorithm, we now state our main lemma on the edge expansion of the decoding graph. The proof technique resembles the expander analysis in [2]. For the complete proof, see [10].

Lemma 5. (Main lemma) *The edge expansion of Dec_kC is*

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

By another argument (proof in [10]) we obtain that

$$h_s(Dec_{\log_2 n}C) \geq h(Dec_kC),$$

where $s = \Theta(7^k)$. Choosing $s = \Theta(M^{(\log_2 7)/2})$, we satisfy Inequality 4 and obtain Inequality 5 (for sufficiently large $|V|$). This gives Theorem 1.

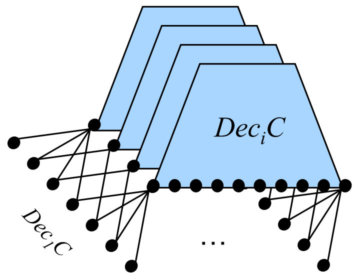

Figure 7: Illustration of the recursive construction of the decoding subgraph. To construct $Dec_{i+1}C$, Dec_iC is replicated 4 times and Dec_1C is replicated 7^i times, and appropriate vertices are identified.

5. EXTENSIONS

In this paper, we focus on lower bounds for Strassen's matrix multiplication algorithm on two machine models. However, the design space of improving fundamental algorithms via communication minimization is much larger. It includes proving lower bounds and developing optimal algorithms; using classical methods as well as fast algorithms like Strassen's; performing matrix multiplication, other matrix algorithms, and more general computations; minimizing time and/or energy; using minimal memory or trading off extra memory for less communication; and using hierarchical, homogeneous, or heterogeneous sequential and parallel models. In this section we discuss a subset of these extensions; see [9, 10] and the references therein for more details.

5.1 Lower Bounds

The proof technique described in Section 3 is not specific to Strassen's algorithm and can be applied more widely. The partition argument is used for classical algorithms in numerical linear algebra [8, 20] where a geometric inequality specifies the per-segment communication cost rather than edge expansion. Further, the edge expansion technique applies to *Strassen-like* algorithms that also multiply square matrices with $o(n^3)$ arithmetic operations, to other fast algorithms for rectangular matrix multiplication, and to other matrix computations.

5.1.1 Strassen-like Algorithms

Strassen-like algorithms are recursive matrix multiplication algorithms based on a scheme for multiplying $k \times k$ matrices using q scalar multiplications for some k and $q < k^3$ (so that the algorithm performs $O(n^{\omega_0})$ flops where $\omega_0 = \log_k q$.) For the latest bounds on the arithmetic complexity of matrix multiplication and references to previous bounds, see [25]. For our lower bound proof to apply, we require another technical criterion for Strassen-like algorithms: the decoding graph must be connected. This class of algorithms includes many (but not all) fast matrix multiplications. For details and examples, see [7, 10].

For Strassen-like algorithms, the statements of the communication lower bounds have the same form as Theorem 1, Corollary 2, and Theorem 3: replace $\log_2 7$ with ω_0 everywhere it appears! The proof technique follows that for Strassen's algorithm. While the bounds for the classical algorithm have the same form, replacing $\log_2 7$ with 3, the proof techniques are quite different [18, 20].

5.1.2 Fast Rectangular Matrix Multiplication

Many fast algorithms have been devised for multiplication of rectangular matrices (see [7] for detailed list). A fast algorithm for multiplying $m \times k$ and $k \times r$ matrices in $q < mkr$ scalar multiplications can be applied recursively to multiply $m^t \times k^t$ and $k^t \times r^t$ 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 and Lemma 5 apply where we plug in mr and q for 4 and 7. In this case, we obtain a result analogous to Theorem 1 which states that the communication cost of such an algorithm is given by $\Omega(q^t/M^{\log_{mr} q-1})$. If the output matrix is the largest of the three matrices (*i.e.*, $k < m$ and $k < r$), 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 2, and can be attained using the algorithmic technique of [6].

5.1.3 The Rest of Numerical Linear Algebra

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 [13]. Therefore, communication cost lower bounds for these algorithms can be derived from our lower bounds for fast matrix multiplication algorithms. 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 [13], and we can then deduce matching lower and upper bounds [10].

5.1.4 Nested Loops Computation

Nearly all of the arguments for proving communication lower bounds are based on establishing a relationship between a given set of data and the amount of useful computation that can be done with that data, a so-called "surface-to-volume" ratio. For example, Hong and Kung [18] use an analysis of dominator sets and minimal sets of CDAGs to establish such ratios. The Loomis-Whitney geometric inequality is applied for this purpose to matrix computations specified by three nested loops in [8, 20]. Recently, Christ et al. [12] have extended this analysis using a generalization of the Loomis-Whitney inequality, known as the Hölder-Brascamp-Lieb inequality, to prove lower bounds for computations that are specified by an arbitrary set of nested loops that linearly access arrays and meet certain other criteria.

5.2 Algorithms

The main motivation for pursuing communication lower bounds is to provide targets for algorithmic performance. Indeed, the conjecture and proof of Theorem 1 and Corollary 2, as well as the existence of an optimal algorithm in the sequential case, were the main motivations for improving the parallel implementations of Strassen's algorithm. Not only were we able to devise an optimal algorithm, but we were able to show with an implementation for distributed-memory machines that it performs much faster in practice [6, 21].

5.2.1 Communication Avoiding Parallel Strassen

In Section 2.2 we stated the communication cost of a new parallel algorithm for Strassen's matrix multiplication, matching the asymptotic lower bound. The details of the algorithm appear in [6], and more extensive implementation details and performance data are given in [21]. We show that the new algorithm is more efficient than any other parallel matrix multiplication algorithm of which we are aware, including those that are based on the classical algorithm and those that are based on previous parallelizations of Strassen's algorithm.

Figure 1 shows performance on a Cray XT4. For results on other machines, see [21]. For example, running on a Cray XE6 with up to 10,000 cores, for a problem of dimension $n = 131712$, our new algorithm attains performance as high as 30% above the peak for classical matrix multiplication, 83% above the best classical implementation, and 75% above the best previous implementation of Strassen's algorithm. Even for a small problem of dimension $n = 4704$, it attains performance 66% higher than the best classical implementation.

5.2.2 Further Applications

The key algorithmic idea in our parallel implementation of Strassen's algorithm is a careful parallel traversal of the recursion tree. This idea works for many other recursive algorithms where the subproblems do not have interdependencies (and it also works in some cases where dependencies exist). For example, classical rectangular matrix multiplication [14] and sparse matrix-matrix multiplication [4] can be parallelized in this way to obtain communication optimality.

The same techniques can be utilized to save energy at the algorithmic level (since communication consumes more energy than computation) as well as to obtain lower bounds on energy requirements [15].

In summary, we believe this work flow of theoretical lower bounds to algorithmic development to efficient implementations is very effective: by considering fundamental computations at an algorithmic level, significant improvements in many applications are possible.

6. ACKNOWLEDGMENTS

We would like to thank Benjamin Lipshitz for his work on many of these ideas and for useful discussions during the writing of this paper.

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, DE-SC0005136, DE-SC0008700, AC02-05CH11231, and DE-FC02-06-ER25786, and DARPA grant HR0011-12-2-0016. 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.

7. REFERENCES

[1] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39:39–5, 1995.

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

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK's user's guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992. Also available from http://www.netlib.org/lapack/.

[4] G. Ballard, A. Buluç, J. Demmel, L. Grigori, B. Lipshitz, O. Schwartz, and S. Toledo. Communication optimal parallel multiplication of sparse random matrices. Technical Report UCB/EECS-2013-13, University of California, Berkeley, Feb 2013.

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

[6] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 193–204, New York, NY, USA, 2012. ACM.

[7] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Graph expansion analysis for communication costs of fast rectangular matrix multiplication. In G. Even and D. Rawitz, editors, *Design and Analysis of Algorithms*, volume 7659 of *Lecture Notes in Computer Science*, pages 13–36. Springer Berlin Heidelberg, 2012.

[8] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. In *SPAA '11: Proceedings of the 23rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–12, New York, NY, USA, 2011. ACM.

[9] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32(3):866–901, 2011.

[10] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. *J. ACM*, 59(6):32:1–32:23, Dec. 2012.

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

[12] M. Christ, J. Demmel, N. Knight, T. Scanlon, and K. Yelick. Communication lower bounds and optimal algorithms for programs that reference arrays – Part I. Manuscript, 2013.

[13] J. Demmel, I. Dumitriu, and O. Holtz. Fast linear algebra is stable. *Numerische Mathematik*, 108(1):59–91, 2007.

[14] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger. Communication-optimal parallel recursive rectangular matrix multiplication. In *Proc. 27th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE, 2013. To appear.

[15] J. Demmel, A. Gearhart, B. Lipshitz, and O. Schwartz. Perfect strong scaling using no additional energy. In *Proc. 27th IEEE International Parallel & Distributed Processing Symposium*, IPDPS '13. IEEE, 2013. To appear.

[16] S. H. Fuller and L. I. Millett, editors. *The Future of Computing Performance: Game Over or Next Level?* The National Academies Press, Washington, D.C., 2011. 200 pages, http://www.nap.edu.

[17] 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.

[18] 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.

[19] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the AMS*, 43(4):439–561, 2006.

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

[21] B. Lipshitz, G. Ballard, J. Demmel, and O. Schwartz. Communication-avoiding parallel Strassen: Implementation and performance. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 101:1–101:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[22] W. F. McColl and A. Tiskin. Memory-efficient matrix multiplication in the BSP model. *Algorithmica*, 24:287–297, 1999. 10.1007/PL00008264.

[23] E. Solomonik and J. Demmel. 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, 2011.

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

[25] V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing*, STOC '12, pages 887–898, New York, NY, USA, 2012. ACM.