Strong Scaling of Matrix Multiplication Algorithms and Memory-Independent Communication Lower Bounds

(Brief Announcement)

Grey Ballard[*]
UC Berkeley
ballard@eecs.berkeley.edu

James Demmel[*†]
UC Berkeley
demmel@cs.berkeley.edu

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

Benjamin Lipshitz[*]
UC Berkeley
lipshitz@berkeley.edu

Oded Schwartz[§]
UC Berkeley
odedsc@eecs.berkeley.edu

ABSTRACT

A parallel algorithm has perfect strong scaling if its running time on P processors is linear in $1/P$, including all communication costs. Distributed-memory parallel algorithms for matrix multiplication with perfect strong scaling have only recently been found. One is based on classical matrix multiplication (Solomonik and Demmel, 2011), and one is based on Strassen's fast matrix multiplication (Ballard, Demmel, Holtz, Lipshitz, and Schwartz, 2012). Both algorithms scale perfectly, but only up to some number of processors where the inter-processor communication no longer scales.

We obtain a memory-independent communication cost lower bound on classical and Strassen-based distributed-memory matrix multiplication algorithms. These bounds imply that no classical or Strassen-based parallel matrix multiplication algorithm can strongly scale perfectly beyond the ranges already attained by the two parallel algorithms mentioned above. The memory-independent bounds and the strong scaling bounds generalize to other algorithms.

ACM Classification Keywords: F.2.1

ACM General Terms: Algorithms, Design, Performance.

Keywords: Communication-avoiding algorithms, Strong scaling, Fast matrix multiplication

[*]Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung.

[†]Research is also supported by DOE grants DE-SC0003959, DE- SC0004938, and 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, while visiting the Institute for Advanced Study.

[§]Research supported by U.S. Department of Energy grants under Grant Numbers DE-SC0003959.

1. INTRODUCTION

In evaluating the recently proposed parallel algorithm based on Strassen's matrix multiplication [2] and comparing the communication costs to the known lower bounds [3], we found a gap between the upper and lower bounds for certain problem sizes. The main motivation of this work is to close this gap by tightening the lower bound for this case, proving that the algorithm is optimal in all cases, up to $O(\log P)$ factors. A similar scenario exists in the case of classical matrix multiplication; in this work we provide the analogous tightening of the existing lower bound [5] to show optimality of another recently proposed algorithm [7].

In addition to proving optimality of algorithms, the lower bounds in this paper yield another interesting conclusion regarding strong scaling. We say that an algorithm strongly scales perfectly if it attains running time on P processors which is linear in $1/P$, including all communication costs. While it is possible for classical and Strassen-based matrix multiplication algorithms to strongly scale perfectly, the communication costs restrict the strong scaling ranges much more than do the computation costs. These ranges depend on the problem size relative to the local memory size, and on the computational complexity of the algorithm.

Interestingly, in both cases the dominance of a memory-independent bound arises, and the strong scaling range ends, exactly when the memory-dependent latency lower bound becomes constant. This observation may provide a hint as to where to look for strong scaling ranges in other algorithms. Of course, since the latency cost cannot possibly drop below a constant, it is an immediate result of the memory-dependent bounds that the latency cost cannot continue to strongly scale perfectly. However the bandwidth cost typically dominates the cost, and it is the memory-independent bandwidth scaling bounds that limit the strong scaling of matrix multiplication in practice. For simplicity we omit discussions of latency cost, since the number of messages is always a factor of M below the bandwidth cost in the strong scaling range, and is always constant outside the strong scaling range.

While the main arguments in this work focus on matrix multiplication, we present results in such a way that they can be generalized to other algorithms, including other $O(n^3)$-based dense and sparse algorithms as in [4] and other fast matrix multiplication algorithms as in [3].

Our paper is organized as follows. In Section 2.1 we prove a memory-independent communication lower bound for Strassen-based matrix multiplication algorithms, and we prove an analogous bound for classical matrix multiplication in Section 2.2. We discuss the implications of these bounds on strong scaling in Section 3 and compare the communication costs of Strassen and classical matrix multiplication as the number of processors increases. In Section 4 we discuss generalization of our bounds to other algorithms. The main results of this paper are summarized in Table 1.

2. COMMUNICATION LOWER BOUNDS

We use the distributed-memory communication model (see, *e.g.*, [4]), where the bandwidth-cost of an algorithm is proportional to the number of words communicated and the latency-cost is proportional to the number of messages communicated along the critical path. We will use the notation that n is the size of the matrices, P is the number of processors, M is the local memory size of each processor, and $\omega_0 = \log_2 7 \approx 2.81$ is the exponent of Strassen's matrix multiplication.

2.1 Strassen's Matrix Multiplication

In this section, we prove a memory-independent lower bound for Strassen's matrix multiplication of $\Omega(n^2/P^{2/\omega_0})$ words, where $\omega_0 = \log_2 7$. We reuse notation and proof techniques from [4]. By prohibiting redundant computations we mean that each arithmetic operation is computed by exactly one processor. This is necessary for interpreting edge expansion as communication cost.

THEOREM 2.1. *Suppose a parallel algorithm performing Strassen's matrix multiplication minimizes computational costs in an asymptotic sense and performs no redundant computation. Then, for sufficiently large P,*[1] *some processor must send or receive at least $\Omega\left(\frac{n^2}{P^{2/w_0}}\right)$ words.*

PROOF. The computation DAG (see e.g., [4] for formal definition) of Strassen's algorithm multiplying square matrices $A \cdot B = C$ can be partitioned into three subgraphs: an encoding of the elements of A, an encoding of the elements of B, and a decoding of the scalar multiplication results to compute the elements of C. These three subgraphs are connected by edges that correspond to scalar multiplications. Call the third subgraph $Dec_{\lg n}C$, where $\lg n = \log_2 n$ is the number of levels of recursion for matrices of dimension n.

In order to minimize computational costs asymptotically, the running time for Strassen's matrix multiplication must be $O(n^{\omega_0}/P)$. Since a constant fraction of the flops correspond to vertices in $Dec_{\lg n}C$, this is possible only if some processor performs $\Theta\left(\frac{n^{\omega_0}}{P}\right)$ flops corresponding to vertices in $Dec_{\lg n}C$.

By Lemma 10 of [3], the edge expansion of Dec_kC is given by $h(Dec_kC) = \Omega((4/7)^k)$. Using Claim 5 there (decomposition into edge disjoint small subgraphs), we deduce that

$$h_s(Dec_{\lg n}C) = \Omega\left(\left(\frac{4}{7}\right)^{\log_7 s}\right), \tag{1}$$

where h_s is the edge expansion for sets of size at most s.

Let S be the set of vertices of $Dec_{\lg n}C$ that correspond to computations performed by the given processor. Set $s = |S| = \Theta\left(\frac{n^{\omega_0}}{P}\right)$. By equation (1), the number of edges between S and $\overline{S}$ is

$$|E(S,\overline{S})| = \Omega\left(s \cdot h_s(Dec_{\lg n}C)\right) = \Omega\left(\frac{n^2}{P^{2/\omega_0}}\right),$$

and because $Dec_{\lg n}C$ is of bounded degree (Fact 9 there) and each vertex is computed by only one processor, the number of words moved is $\Theta(|E(S,\overline{S})|)$ and the result follows. □

2.2 Classical Matrix Multiplication

In this section, we prove a memory-independent lower bound for classical matrix multiplication of $\Omega(n^2/P^{2/3})$ words. The same result appears elsewhere in the literature, under slightly different assumptions: in the LPRAM model

[1]The theorem applies to any $P \geq 2$ with a strict enough assumption on the load balance among vertices in $Dec_{\lg n}C$ as defined in the proof.

[1], where no data exists in the (unbounded) local memories at the start of the algorithm; in the distributed-memory model [5], where the local memory size is assumed to be $M = \Theta(n^2/P^{2/3})$; and in the distributed-memory model [7], where the algorithm is assumed to perform a certain amount of input replication. Our bound is for the distributed memory model, holds for any M, and assumes no specific communication pattern.

Recall the following special case of the Loomis-Whitney geometric bound:

LEMMA 2.2. [6] *Let V be a finite set of lattice points in $\mathbf{R}^3$, i.e., points (x, y, z) with integer coordinates. Let V_x be the projection of V in the x-direction, i.e., all points (y, z) such that there exists an x so that $(x, y, z) \in V$. Define V_y and V_z similarly. Let $|\cdot|$ denote the cardinality of a set. Then $|V| \leq \sqrt{|V_x| \cdot |V_y| \cdot |V_z|}$.*

Using Lemma 2.2 (in a similar way to [4, 5]), we can describe the ratio between the number of scalar multiplications a processor performs and the amount of data it must access.

LEMMA 2.3. *Suppose a processor has I words of initial data at the start of an algorithm, performs $\Theta(n^3/P)$ scalar multiplications within classical matrix multiplication, and then stores O words of output data at the end of the algorithm. Then the processor must send or receive at least $\Omega(n^2/P^{2/3}) - I - O$ words during the execution of the algorithm.*

PROOF. We follow the proofs in [4, 5]. Consider a discrete $n \times n \times n$ cube where the lattice points correspond to the scalar multiplications within the matrix multiplication $A \cdot B$ (*i.e.*, lattice point (i, j, k) corresponds to the scalar multiplication $a_{ik} \cdot b_{kj}$). Then the three pairs of faces of the cube correspond to the two input and one output matrices.

The projections on the three faces correspond to the input/output elements the processor has to access (and must communicate if they are not in its local memory). By Lemma 2.2, and the fact that $\sqrt{|V_x| \cdot |V_y| \cdot |V_z|} \leq \sqrt{\frac{1}{6}(|V_x| + |V_y| + |V_z|)^3}$, the number of words the processor must access is at least $\sqrt[3]{6}\,|V|^{2/3} = \Omega(n^2/P^{2/3})$. Since the processor starts with I words and ends with O words, the result follows. □

THEOREM 2.4. *Suppose a parallel algorithm performing classical dense matrix multiplication begins with one copy of the input matrices and minimizes computational costs in an asymptotic sense. Then, for sufficiently large P,[2] some processor must send or receive at least $\Omega\left(\frac{n^2}{P^{2/3}}\right)$.*

PROOF. At the end of the algorithm, every element of the output matrix must be fully computed and exist in some processor's local memory (though multiples copies of the element may exist in multiple memories). For each output element, we designate one memory location as the output and disregard all other copies. For each of the n^2 designated memory locations, we consider the n scalar multiplications whose results were used to compute its value and disregard all other redundantly computed scalar multiplications.

In order to minimize computational costs asymptotically, the running time for classical dense matrix multiplication must be $O(n^3/P)$. This is possible only if at least a constant fraction of the processors perform $\Theta\left(\frac{n^3}{P}\right)$ of the scalar multiplications corresponding to designated outputs.

Since there exists only one copy of the input matrices and designated output–$O(n^2)$ words of data–some processor which performs $\Theta(n^3/P)$ multiplications must start and end with no more than $I + O = O(n^2/P)$ words of data. Thus, by Lemma 2.3, some processor must read or write $\Omega(n^2/P^{2/3}) - O(n^2/P) = \Omega(n^2/P^{2/3})$ words of data. □

[2]The theorem applies to any $P \geq 2$ with a strict enough assumption on the load balance.

	Classical	Strassen
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	[7]	[2]

Table 1: Bandwidth-cost lower bounds for matrix multiplication and perfect strong scaling ranges. The classical memory dependent bound is due to [5], and the Strassen memory dependent bound is due to [3]. The memory-independent bounds are proved here, though variants of the classical bound appear in [1, 5, 7].

3. LIMITS OF STRONG SCALING

In this section we present limits of strong scaling of matrix multiplication algorithms. These are immediate implications of the memory independent communication lower bounds proved in Section 2. Roughly speaking, the memory-dependent communication-cost lower-bound is of the form $\Omega\left(f(n, M)/P\right)$ for both classical and Strassen matrix multiplication algorithms. However, the memory independent lower bounds are of the form $\Omega\left(f(n, M)/P^c\right)$ where $c < 1$ (see Table 1). This implies that strong scaling is not possible when the memory-independent bound dominates. We make this formal below.

COROLLARY 3.1. *Suppose a parallel algorithm performing Strassen's matrix multiplication minimizes bandwidth and computational costs in an asymptotic sense and performs no redundant computation. Then the algorithm can achieve perfect strong scaling only for $P = O\left(\frac{n^{\omega_0}}{M^{\omega_0/2}}\right)$.*

PROOF. By [3], any parallel algorithm performing matrix multiplication based on Strassen moves at least $\Omega\left(\frac{n^{\omega_0}}{PM^{\omega_0/2-1}}\right)$ words. By Theorem 2.1, a parallel algorithm that minimizes computational costs and performs no redundant computation moves at least $\Omega\left(\frac{n^2}{P^{2/\omega_0}}\right)$ words. This latter bound dominates in the case $P = \Omega\left(\frac{n^{\omega_0}}{M^{\omega_0/2}}\right)$. Thus, while a communication-optimal algorithm will strongly scale perfectly up to this threshold, after the threshold the communication cost will scale as $1/P^{2/\omega_0}$ rather than $1/P$. □

COROLLARY 3.2. *Suppose a parallel algorithm performing classical dense matrix multiplication starts and ends with one copy of the data and minimizes bandwidth and computational costs in an asymptotic sense. Then the algorithm can achieve perfect strong scaling only for $P = O\left(\frac{n^3}{M^{3/2}}\right)$.*

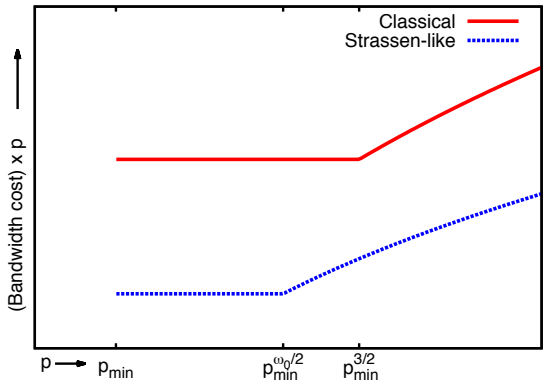

Figure 1: Bandwidth costs and strong scaling of matrix multiplication: classical vs. Strassen-based. Horizontal lines correspond to perfect strong scaling. $\mathbf{P_{min}}$ is the minimum number of processors required to store the input and output matrices.

PROOF. By [5], any parallel algorithm performing matrix multiplication moves at least $\Omega\left(\frac{n^3}{P\sqrt{M}}\right)$ words. By Theorem 2.4, a parallel algorithm that starts and ends with one copy of the data and minimizes computational costs moves at least $\Omega\left(\frac{n^2}{P^{2/3}}\right)$ words. This latter bound dominates in the case $P = \Omega\left(\frac{n^3}{M^{3/2}}\right)$. Thus, while a communication-optimal algorithm will strongly scale perfectly up to this threshold, after the threshold the communication cost will scale as $1/P^{2/3}$ rather than $1/P$. □

In Figure 1 we present the asymptotic communication costs of classical and Strassen-based algorithms for a fixed problem size as the number of processors increases. Both of the perfectly strong scaling algorithms stop scaling perfectly above some number of processors, which depends on the matrix size and the available local memory size.

Let $P_{\min} = \Theta\left(\frac{n^2}{M}\right)$ be the minimum number of processors required to store the input and output matrices. By Corollaries 3.1 and 3.2 the perfect strong scaling range is $P_{\min} \leq P \leq P_{\max}$ where $P_{\max} = \Theta(P_{\min}^{3/2})$ in the classical case and $P_{\max} = \Theta(P_{\min}^{\omega_0/2})$ in the Strassen case.

Note that the perfect strong scaling range is larger for the classical case, though the communication costs are higher.

4. EXTENSIONS AND OPEN PROBLEMS

The memory-independent bound and perfect strong scaling bound of Strassen's matrix multiplication (Theorem 2.1 and Corollary 3.1) apply to other Strassen-like algorithms, as defined in [4], with ω_0 being the exponent of the total arithmetic count, provided that $Dec_{\lg n}C$ is connected. The proof follows that of Theorem 2.1 and of Corollary 3.1, but uses Claim 18 of [3] instead of Fact 9 there, and replaces Lemma 10 there with its extension.

The memory-dependent bound of classical matrix multiplication of [5] was generalized in [4] to algorithms which perform computations of the form

$$\text{Mem}(c(i,j)) = f_{ij}(g_{ijk}(\text{Mem}(a(i,k)), \text{Mem}(b(k,j)))), \quad (2)$$

where $\text{Mem}(i)$ denotes the argument in memory location i and f_{ij} and g_{ijk} are functions which depend non-trivially on their arguments (see [4] for more detailed definitions).

The memory-independent bound of classical matrix multiplication (Theorem 2.4) applies to these other algorithms as well. If the algorithm begins with one copy of the input data and minimizes computational costs in an asymptotic sense, then, for sufficiently large P, some processor must send or receive at least $\Omega\left(\left(\frac{G}{P}\right)^{2/3} - \frac{D}{P}\right)$ words, where G is the total number of g_{ijk} computations and D is the number of non-zeros in the input and output. The proof follows that of Lemma 2.3 and Theorem 2.4, setting $|V| = G$ (instead of n^3), replacing n^3/P with G/P , and setting $I+O = O(D/P)$ (instead of $O(n^2/P)$).

Algorithms which fit the form of equation (2) include LU and Cholesky decompositions, sparse matrix-matrix multiplication, as well as algorithms for solving the all-pairs-shortest-paths problem. Only a few of these have parallel algorithms which attain the lower bounds in all cases. In several cases, it seems likely that one can prove better bounds than those presented here, thus obtaining a stricter bound on perfect strong scaling.

We also believe that our bounds can be generalized to QR decomposition and other orthogonal transformations, fast linear algebra, fast Fourier transform, and other recursive algorithms.

5. REFERENCES

[1] AGGARWAL, A., CHANDRA, A. K., AND SNIR, M. Communication complexity of PRAMs. *Theoretical Computer Science 71*, 1 (1990), 3 – 28.

[2] BALLARD, G., DEMMEL, J., HOLTZ, O., LIPSHITZ, B., AND SCHWARTZ, O. Communication-optimal parallel algorithm for Strassen's matrix multiplication, 2012. Submitted to SPAA.

[3] BALLARD, G., DEMMEL, J., HOLTZ, O., AND SCHWARTZ, O. Graph expansion and communication costs of fast matrix multiplication. In *SPAA '11: Proceedings of the 23rd Annual Symposium on Parallelism in Algorithms and Architectures* (New York, NY, USA, 2011), ACM, pp. 1–12.

[4] BALLARD, G., DEMMEL, J., HOLTZ, O., AND SCHWARTZ, O. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Analysis Applications 32*, 3 (2011), 866–901.

[5] IRONY, D., TOLEDO, S., AND TISKIN, A. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput. 64*, 9 (2004), 1017–1026.

[6] LOOMIS, L. H., AND WHITNEY, H. An inequality related to the isoperimetric inequality. *Bulletin of the AMS 55* (1949), 961–962.

[7] SOLOMONIK, E., AND DEMMEL, J. 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* (2011), Springer.