Communication lower bounds and optimal algorithms for numerical linear algebra[*†]

G. Ballard[1], E. Carson[2], J. Demmel[2,3],
M. Hoemmen[4], N. Knight[2] and O. Schwartz[2]

[1] *Sandia National Laboratories,
Livermore, CA 94551, USA*
E-mail: gmballa@sandia.gov

[2] *EECS Department, UC Berkeley,
Berkeley, CA 94704, USA*
E-mail: ecc2z@cs.berkeley.edu, knight@cs.berkeley.edu,
odedsc@cs.berkeley.edu

[3] *Mathematics Department, UC Berkeley,
Berkeley, CA 94704, USA*
E-mail: demmel@cs.berkeley.edu

[4] *Sandia National Laboratories,
Albuquerque, NM 87185, USA*
E-mail: mhoemme@sandia.gov

[*] We acknowledge funding from Microsoft (award 024263) and Intel (award 024894), and matching funding by UC Discovery (award DIG07-10227). Additional support comes from ParLab affiliates National Instruments, Nokia, NVIDIA, Oracle and Samsung, as well as MathWorks. Research is also supported by DOE grants DE-SC0004938, DE-SC0005136, DE-SC0003959, DE-SC0008700, DE-SC0010200, DE-FC02-06-ER25786, AC02-05CH11231, and DARPA grant HR0011-12-2-0016. This research is supported by grant 3-10891 from the Ministry of Science and Technology, Israel, and grant 2010231 from the US–Israel Bi-National Science Foundation. This research was supported in part by an appointment to the Sandia National Laboratories Truman Fellowship in National Security Science and Engineering, sponsored by Sandia Corporation (a wholly owned subsidiary of Lockheed Martin Corporation) as Operator of Sandia National Laboratories under its US Department of Energy Contract DE-AC04-94AL85000.

[†] Colour online for monochrome figures available at journals.cambridge.org/anu.

The traditional metric for the efficiency of a numerical algorithm has been the number of arithmetic operations it performs. Technological trends have long been reducing the time to perform an arithmetic operation, so it is no longer the bottleneck in many algorithms; rather, *communication*, or moving data, is the bottleneck. This motivates us to seek algorithms that move as little data as possible, either between levels of a memory hierarchy or between parallel processors over a network. In this paper we summarize recent progress in three aspects of this problem. First we describe lower bounds on communication. Some of these generalize known lower bounds for dense classical ($O(n^3)$) matrix multiplication to all direct methods of linear algebra, to sequential and parallel algorithms, and to dense and sparse matrices. We also present lower bounds for Strassen-like algorithms, and for iterative methods, in particular Krylov subspace methods applied to sparse matrices. Second, we compare these lower bounds to widely used versions of these algorithms, and note that these widely used algorithms usually communicate asymptotically more than is necessary. Third, we identify or invent new algorithms for most linear algebra problems that do attain these lower bounds, and demonstrate large speed-ups in theory and practice.

CONTENTS

1. Introduction

1.1. Motivation

Linear algebra problems appear throughout computational science and engineering, as well as the analysis of large data sets (Committee on the Analysis of Massive Data; Committee on Applied and Theoretical Statistics; Board on Mathematical Sciences and Their Applications; Division on Engineering and Physical Sciences; National Research Council 2013), so it is important to solve them as efficiently as possible. This includes solving systems of linear equations, least-squares problems, eigenvalue problems, the singular value decomposition, and their many variations that can depend on the structure of the input data.

When numerical algorithms were first developed (not just for linear algebra), efficiency was measured by counting arithmetic operations. Over time, as technological trends such as Moore's law kept making operations faster, the bottleneck in many algorithms shifted from arithmetic to *communication*, that is, moving data, either between levels of the memory hierarchy such as DRAM and cache, or between parallel processors connected over a network. Communication is necessary because arithmetic can only be performed on two operands in the same memory at the same time, and (in the case of a memory hierarchy) in the smallest, fastest memory at the top of the hierarchy (*e.g.*, cache). Indeed, a sequence of recent reports (Graham, Snir and Patterson 2004, Fuller and Millett 2011) has documented this trend. Today the cost of moving a word of data (measured in time or energy) can exceed the cost of an arithmetic operation by orders of magnitude, and this gap is growing exponentially over time.

Motivated by this trend, the numerical linear algebra community has been revisiting all the standard algorithms, direct and iterative, for dense and sparse matrices, and asking three questions: Are there lower bounds on the amount of communication required by these algorithms? Do existing algorithms attain the lower bounds? If not, are there new algorithms that do?

The answers, which we will discuss in more detail in this paper, are briefly as follows. There are in fact communication lower bounds for most direct and iterative (*i.e.*, Krylov subspace) algorithms. These lower bounds apply to dense and sparse matrices, and to sequential, parallel and more complicated computer architectures. Existing algorithms in widely used libraries often do asymptotically more communication than these lower bounds require, even for heavily studied operations such as dense matrix multiplication (matmul for short). In many cases there are new algorithms that do attain the lower bounds, and show large speed-ups in theory and practice (even for matmul). These new algorithms do not just require 'loop transformations' but sometimes have different numerical properties, different ways to represent the answers, and different data structures.

Historically, the linear algebra community has been adapting to rising communication costs for a long time. For example, the level-1 Basic Linear Algebra Subroutines (BLAS1) (Lawson, Hanson, Kincaid and Krogh 1979) were replaced by BLAS2 (Dongarra, Croz, Hammarling and Hanson 1988*a*, 1988*b*) and then BLAS3 (Dongarra, Croz, Duff and Hammarling 1990*a*, 1990*b*), and EISPACK (Smith *et al.* 1976) and LINPACK (Dongarra, Moler, Bunch and Stewart 1979) were replaced by LAPACK (Anderson *et al.* 1992) and ScaLAPACK (Blackford *et al.* 1997), to mention a few projects. So it may be surprising that large speed-ups are still possible.

1.2. Modelling communication costs

More precisely, we will model the cost of communication as follows. There are two costs associated with communication. For example, when sending n words from one processor to another over a network, the words are first packed into a contiguous block of memory called a *message*, which is then sent to the destination processor. There is a fixed overhead time (called the *latency cost* or α) required for the packing and transmission over the network, and also time proportional to n needed to transmit the words (called the *bandwidth cost* or βn). In other words, we model the time to send one message of size n by $\alpha + \beta n$, and the time to send S messages containing a total of W words by $\alpha S + \beta W$.

Letting γ be the time to perform one arithmetic operation, and F the total number of arithmetic operations, our overall performance model becomes $\alpha S + \beta W + \gamma F$. The same technological trends cited above tell us that $\alpha \gg \beta \gg \gamma$. This is why it is important to count messages S and words W separately, because either one may be the bottleneck. Later we will present lower bounds on both S and W, because it is of interest to have algorithms that minimize both bandwidth and latency costs.

On a sequential computer with a memory hierarchy, the model $\alpha S + \beta W + \gamma F$ is enough to model two levels of memory, say DRAM and cache. When there are multiple levels of memory, there is a cost associated with moving data between each adjacent pair of levels, so there will be an $\alpha S + \beta W$ term associated with each level.

On a parallel computer, $\alpha S + \beta W + \gamma F$ will initially refer to the communication and arithmetic done by one processor only. A lower bound for one processor is (sometimes) enough for a lower bound on the overall algorithm, but to upper-bound the time required by an entire algorithm requires us to sum these terms along the *critical path*, that is, a sequence of processors that must execute in a linear order (because of data dependences), and that also maximizes the sum of the costs. Note that there may be different critical paths for latency costs, bandwidth costs and arithmetic costs.

We note that this simple model may be naturally extended to other kinds of architectures. First, when the architecture can overlap communication and computation (*i.e.*, perform them in parallel), we see that $\alpha S + \beta W + \gamma F$ may be replaced by $\max(\alpha S + \beta W, \gamma F)$ or $\max(\alpha S, \beta W, \gamma F)$; this can lower the cost by at most a factor of 2 or 3, and so does not affect our asymptotic analysis. Second, on a *heterogeneous* parallel computer, that is, with different processors with different values of α, β, γ, memory sizes, *etc.*, one can still use $\alpha_i S_i + \beta_i W_i + \gamma_i F_i$ as the cost of processor i, and take the maximum over i or sum over critical paths to get lower and upper bounds. Third, on a parallel machine with local memory hierarchies (the usual case), one can include both kinds of costs.

We call an algorithm *communication-optimal* if it asymptotically attains communication lower bounds for a particular architecture (we sometimes allow an algorithm to exceed the lower bound by factors that are polylogarithmic in the problem size or machine parameters). More informally, we call an algorithm *communication-avoiding* if it is communication-optimal, or if it communicates significantly less than a conventional algorithm.

Finally, we note that this timing model may be quite simply converted to an energy model. First, interpret α_E, β_E and γ_E as joules per message, per word and per flop, respectively, instead of seconds per message, per word and per flop. The same technological trends as before tell us that $\alpha_E \gg \beta_E \gg \gamma_E$ and are all improving, but growing apart exponentially over time. Second, for each memory unit we add a term $\delta_E M$, where M is the number of words of memory used and δ_E is the joules per word per second to store data in that memory. Third, we add another term $\epsilon_E T$, where T is the run time and ϵ_E is the number of joules per second spent in 'leakage', cooling and other activities. Thus a (single) processor may be modelled as using $\alpha_E S + \beta_E W + \gamma_E F + \delta_E M + \epsilon_E T$ joules to solve a problem. Lower and upper bounds on S, W, and T translate to energy lower and upper bounds.

1.3. Summary of results for direct linear algebra

We first summarize previous lower bounds for direct linear algebra, and then the new ones. Hong and Kung (1981) considered any matmul algorithm that has the following properties.

(1) It requires the usual $2n^3$ multiplications and additions to multiply two $n \times n$ matrices $C = A \cdot B$ on a sequential machine with a two-level memory hierarchy.
(2) The large (but slow) memory level initially contains A and B, and also C at the end of the algorithm.
(3) The small (but fast) memory level contains only M words, where $M < 3n^2$, so it is too small to contain A, B and C simultaneously.

Then Hong and Kung (1981) showed that any such algorithm must move at least $W = \Omega(n^3/M^{1/2})$ words between fast and slow memory. This is attained by well-known 'blocking' algorithms that partition A, B and C into square sub-blocks of dimension $(M/3)^{1/2}$ or a little less, so that one sub-block each of A, B and C fit in fast memory, and that multiply sub-block by sub-block.

This was generalized to the parallel case by Irony, Toledo and Tiskin (2004). When each of the P processors stores the minimal $M = O(n^2/P)$ words of data and does an equal fraction $2n^3/P$ of the arithmetic, their lower bound is $W = \Omega(\#\text{flops}/M^{1/2}) = \Omega(n^3/P/(n^2/P)^{1/2}) = \Omega(n^3/P^{1/2})$, which is attained by Cannon's algorithm (Cannon 1969) and SUMMA (van de Geijn and Watts 1997). The paper by Irony *et al.* (2004) also considers the so-called '3D' case, which does less communication by replicating the matrices $P^{1/3}$ times. This requires $M = n^2/P^{2/3}$ words of fast memory per processor. The lower bound becomes $W = \Omega(n^3/P/M^{1/2}) = \Omega(n^2/P^{2/3})$ and is a factor $P^{1/6}$ smaller than before, and it is attainable (Aggarwal, Chandra and Snir 1990, Johnsson 1992, Agarwal *et al.* 1995).

This was eventually generalized by Ballard, Demmel, Holtz and Schwartz (2011*d*) to *any* classical algorithm, that is, one that sufficiently resembles the three nested loops of matrix multiplication, to $W = \Omega(\#\text{flops}/M^{1/2})$ (this will be formalized below). This applies to (1) matmul, all the BLAS, Cholesky, LU decomposition, LDL^T factorization, algorithms that perform factorizations with orthogonal matrices (under certain technical conditions), and some graph-theoretic algorithms such as Floyd–Warshall (Floyd 1962, Warshall 1962), (2) dense or sparse matrices, where #flops may be much less than n^3, (3) some whole programs consisting of sequences of such operations, such as computing A^k by repeated matrix multiplication, no matter how the operations are interleaved, and (4) sequential, parallel and other architectures mentioned above. This lower bound applies for M larger than needed to store all the data once, up to a limit (Ballard *et al.* 2012*d*).

Furthermore, this lower bound on the bandwidth cost W yields a simple lower bound on the latency cost S. If the largest message allowed by the architecture is $m_{\max}$, then clearly $S \geq W/m_{\max}$. Since no message can be larger than than the memory, that is, $m_{\max} \leq M$, we get $S = \Omega(\#\text{flops}/M^{3/2})$. Combining these lower bounds on W and W with the number of arithmetic operations yields a lower bound on the overall run time, and this in turn yields a lower bound on the energy required to solve the problem (Demmel, Gearhart, Lipshitz and Schwartz 2013*b*).

Comparing these bounds to the costs of conventional algorithms, we see that they are frequently not attained, even for dense linear algebra. Many new algorithms have been invented that do attain these lower bounds, which will be summarized in Sections 3 (classical algorithms) and 5 (fast Strassen-like algorithms).

1.4. Summary of results for iterative linear algebra

Krylov subspace methods are widely used, such as GMRES (Saad and Schultz 1986) and conjugate gradient (CG) (Hestenes and Stiefel 1952) for linear systems, or Lanczos (Lanczos 1950) and Arnoldi (Arnoldi 1951) for eigenvalue problems. In their unpreconditioned variants, each iteration performs 1 (or a few) sparse matrix–vector multiplications (SpMV for short) with the input matrix A, as well as some dense linear algebra operations such as dot products. After s iterations, the resulting vectors span an $s+1$ dimensional Krylov subspace, and the 'best' solution (depending on the algorithm) is chosen from this space. This means that the communication costs grow in proportion to s. In the sequential case, when A is too large to fit in the small fast memory, it is read from the large slow memory s times. If A has nnz nonzero entries that are stored in an explicit data structure, this means that $W \geq s \cdot \text{nnz}$. In the parallel case, with A and the vectors distributed across processors, communication is required at each iteration. Unless A has a simple block diagonal structure with separate blocks (and corresponding subvectors) assigned to separate processors, at least one message per processor is required for the SpMV. And if a dot product is needed, at least $\log P$ messages are needed to compute the sum. Altogether, this means $S \geq s \log P$.

To avoid communication, we make certain assumptions on the matrix A: it must be 'well-partitioned' in a sense to be made more formal in Section 7, but for now think of a matrix resulting from a mesh or other spatial discretization, partitioned into roughly equally large submeshes with as few edges as possible connecting one submesh to another. In other words, the partitioning should be load-balanced and have a low 'surface-to-volume ratio'.

In this case, it is possible to take s steps of many Krylov subspace methods for the communication cost of one step. In the sequential case this means $W = O(\text{nnz})$ instead of $s \cdot \text{nnz}$; clearly reading the matrix once from slow memory for a cost of $W = \text{nnz}$ is a lower bound. In the parallel case this means $S = O(\log p)$ instead of $O(s \log p)$; clearly the latency cost of one dot product is also a lower bound.

The idea behind these new algorithms originally appeared in the literature as *s-step* methods (see Section 8 for references). One first computed a different basis of the same Krylov subspace, for example using the *matrix-powers kernel* $[b, Ab, A^2b, \dots, A^s b]$, and then reformulated the rest of the algorithm to compute the same 'best' solution in this subspace. The original motivation was exposing more parallelism, not avoiding communication, which requires different ways of implementing the matrix-powers kernel. But this research encountered a numerical stability obstacle: the matrix-powers kernel is basically running the power method, so that the vectors $A^i b$ are becoming more nearly parallel to the dominant eigenvector, resulting in a very ill-conditioned basis and failure to converge. Later research partly alleviated this by using different polynomial bases $[b, p_1(A)b, p_2(A)b, \dots, p_s(A)b]$ where $p_i(A)$ is a degree-i polynomial in A, chosen to make the vectors more linearly independent (*e.g.*, Philippe and Reichel 2012). But choosing a good polynomial basis (still a challenge to do automatically in general) was not enough to guarantee convergence in all cases, because two recurrences in the algorithm independently updating the approximate solution and residual could become 'decoupled', with the residual falsely indicating continued convergence of the approximate solution. This was finally overcome by a generalization (Carson and Demmel 2014) of the residual replacement technique introduced by Van der Vorst and Ye (1999). Reliable and communication-avoiding s-step methods have now been developed for many Krylov methods, which offer large speed-ups in theory and practice, though many open problems remain. These will be discussed in Section 8.

There are two directions in which this work has been extended. First, many but not all sparse matrices are stored with explicit nonzero entries and their explicit indices: the nonzero entries may be implicit (for example, -1 on the offdiagonal of a graph Laplacian matrix), or the indices may be implicit (common in matrices arising in computer vision applications, where the locations of nonzeros are determined by the associated pixel locations), or both may be implicit, in which case the matrix is commonly called a *stencil*. In these cases intrinsically less communication is necessary. In particular, for stencils, only the vectors need to be communicated; we discuss this further in Section 7.

Second, one often uses *preconditioned* Krylov methods, *e.g.*, $MAx = Mb$ instead of $Ax = b$, where M somehow approximates A^{-1}. It is possible to derive corresponding s-step methods for many such methods; see, for instance, Hoemmen (2010). If M has a similar sparsity structure to A (or is even sparser), then previous communication-avoiding techniques may be used. But since A^{-1} is generically dense, M would also often be dense if written out explicitly, even if it is applied using sparse techniques (*e.g.*, solving sparse triangular systems arising from an incomplete factorization). This means that many common preconditioners cannot be used in a straightforward way. One class that can be used is that of *hierarchically semiseparable matrices*, which are represented by low-rank blocks; all of these extensions are discussed in Section 8.

1.5. Outline of the rest of the paper

The rest of this paper is organized as follows. The first half (Sections 2–5) is devoted to direct (mostly dense, some sparse) linear algebra, and the second (Sections 6–8) to iterative linear algebra (mostly for sparse matrices).

We begin in Sections 2 and 3 with communication costs of classical direct algorithms. We present lower bounds for classical computations in Section 2, starting with the basic case of classical matrix multiplication (§ 2.1), extensions using reductions (§ 2.2), generalization to three-nested-loops computations (§§ 2.3, 2.4), orthogonal transformations (§ 2.5), and further extensions and impact of the lower bounds (§ 2.6).

In Section 3 we discuss communication costs of classical algorithms, both conventional and communication-optimal. We summarize sequential (§ 3.1) and parallel ones (§ 3.2), then provide some details (§ 3.3) and point to remaining gaps and future work (§ 3.4).

In Sections 4 and 5 we discuss communication costs of fast (Strassen-like) linear algebra. We present lower bounds for Strassen-like computations in Section 4, starting with the expansion analysis of computation graphs (§§ 4.1, 4.2) applied to Strassen's matrix multiplication (§ 4.3), and Strassen-like multiplication (§ 4.4), and other algorithms (§ 4.5). In Section 5 we discuss communication-optimal Strassen-like algorithms that attain the lower bounds, both sequential (§§ 5.1, 5.2), and parallel (§§ 5.3, 5.4).

In Sections 6, 7 and 8 we discuss communication costs of iterative linear algebra. We begin with sparse matrix–vector multiplication (SpMV) in Section 6, starting with sequential lower bounds and optimal algorithms (§ 6.1), followed by parallel lower bounds and optimal algorithms (§ 6.2). The main conclusion of Section 6 is that any conventional Krylov subspace (or similar) method that performs a sequence of SpMVs will most likely be communication-bound. This motivates Section 7, which presents communication-avoiding Krylov basis computations, starting with lower bounds on communication costs (§ 7.1), then Akx algorithms (§ 7.2), and blocking covers (§ 7.3). We then point to related work and future research (§ 7.4).

Based on the kernels introduced in Section 7, in Section 8 we present communication-avoiding Krylov subspace methods for eigenvalue problems (§ 8.2), and for linear systems (§ 8.3). We demonstrate speed-ups (§ 8.4), and discuss numerical issues with finite precision (§ 8.5), and how to apply preconditioning in communication-avoiding ways (§ 8.6). We conclude with remaining gaps and future research (§ 8.7).

2. Lower bounds for classical computations

In this section we consider lower bounds for *classical* direct linear algebra computations. These computations can be specified by algorithms that are basically composed of three nested loops; see further details in Section 2.3. For some special cases, such as dense matrix multiplication, 'classical' means that the algorithm performs all n^3 scalar multiplications in the definition of $n \times n$ matrix multiplication, though the order in which the scalar multiplications are performed is arbitrary. We thus exclude from the discussion in this section, for example, Strassen's fast matrix multiplication (Strassen 1969). See Sections 4 and 5 for lower bounds and upper bounds of Strassen-like methods.

2.1. Matrix multiplication

Hong and Kung (1981) proved a lower bound on the bandwidth cost required to perform dense matrix multiplication in the sequential two-level memory model using a classical algorithm, where the input matrices are too large to fit in fast memory. They obtained the following result, using what they called a 'red–blue pebble game' analysis of the computation graph of the algorithm.

Theorem 2.1 (Hong and Kung 1981, Corollary 6.2). For classical matrix multiplication of dense $m \times k$ and $k \times n$ matrices implemented on a machine with fast memory of size M, the number of words transferred between fast and slow memory is

$$W = \Omega\left(\frac{mkn}{M^{1/2}}\right).$$

This result was proved using a different technique by Irony *et al.* (2004) and generalized to the distributed-memory parallel case. They state the following parallel bandwidth cost lower bound using an argument based on the Loomis–Whitney inequality (Loomis and Whitney 1949), given as Lemma 2.5.

Theorem 2.2 (Irony *et al.* 2004, Theorem 3.1). For classical matrix multiplication of dense $m \times k$ and $k \times n$ matrices implemented on a distributed-memory machine with P processors, each with a local memory of size M, the number of words communicated by at least one processor is

$$W = \Omega\left(\frac{mkn}{PM^{1/2}} - M\right).$$

In the case where $m = k = n$ and each processor stores the minimal $M = O(n^2/P)$ words of data, the lower bound on bandwidth cost becomes $\Omega(n^2/P^{1/2})$. The authors also consider the case where the local memory size is much larger, $M = \Theta(n^2/P^{2/3})$, in which case $O(P^{1/3})$ times as much memory is used (compared to the minimum possible) and less communication is necessary. In this case the bandwidth cost lower bound becomes $\Omega(n^2/P^{2/3})$. See Section 2.6.2 for a discussion of limits on reducing communication by using extra memory, and Section 3.3.1 for further algorithmic discussion on utilizing extra memory for matrix multiplication.

For simplicity we will assume real matrices throughout the rest of this section; all the results generalize to the complex case.

2.2. Extending lower bounds with reductions

It is natural to try to extend the lower bounds for matrix multiplication to other linear algebra operation by means of reductions. Given a lower bound for one algorithm, we can make a reduction argument to extend that bound to another algorithm. In our case, given the matrix multiplication bounds, if we can show how to perform matrix multiplication using another algorithm (assuming the transformation requires no extra communication in an asymptotic sense), then the same bound must apply to the other algorithm, under the same assumptions.