Communication Lower Bounds for Distributed-Memory Matrix Multiplication[1]

Dror Irony and Sivan Toledo

School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel.
Email: {irony,stoledo}@tau.ac.il
URL: http://www.tau.ac.il/~stoledo

We present lower bounds on the amount of communication that matrix-multiplication algorithms must perform on a distributed-memory parallel computer. We denote the number of processors by P and the dimension of square matrices by n. We show that the most widely-used class of algorithms, the so-called 2-dimensional (2D) algorithms, are optimal, in the sense that in any algorithm that uses only $O(n^2/P)$ words of memory per processor, at least one processor must send or receive $\Omega(n^2/\sqrt{P})$ words. We also show that algorithms from another class, the so-called 3-dimensional (3D) algorithms, are also optimal. These algorithm use replication to reduce communication. We show that in any algorithm that uses $O(n^2/P^{2/3})$ words of memory per processor, at least one processor must send or receive $\Omega(n^2/P^{2/3})$ words. Furthermore, we show a continuous tradeoff between the size of local memories and the amount of communication that must be performed. The 2D and 3D bounds are essentially instantiations of this tradeoff. We also show that if the input is distributed across memory-processor nodes and if the input is not replicated, then $\Omega(n^2)$ words must cross any bisection cut of the machine. Our bounds apply only to conventional $\Theta(n^3)$ algorithms. They do not apply to Strassen's algorithm or other $o(n^3)$ algorithms.

Key Words: communication, lower bounds, distributed memory, matrix multiplication

1. INTRODUCTION

Although communication is a bottleneck in many computations running on distributed-memory parallel computers and on clusters of workstations or servers, few communication lower bounds have been proved. We know a great deal about the amount

[1]This research was supported by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (grant number 572/00 and grant number 9060/99) and by the University Research Fund of Tel-Aviv University.

of communication that specific algorithms perform, but we know little about how much communication they must perform.

We present lower bounds on the amount of communication that is required to multiply matrices using a conventional algorithm on a distributed-memory parallel computer. The analysis uses a unified framework that also applies to the analysis of capacity cache misses in sequential matrix-multiplication algorithm.

We use a simple yet realistic computational model to prove the lower bounds. We model a parallel computer as a collection of P processor-memory nodes connected by a communication network. That is, all the memory is distributed among the nodes, which can communicate with each other over a network. Our analysis bounds the number of words that must be sent and received by at least one of the nodes. The bounds apply even if each processor-memory node include several processors, which is a fairly common today in machines ranging from clusters of dual-processor workstations to SGI Origin 2000s and IBM SPs.

Aggarwal, Chandra, and Snir [3] presented lower bounds on the amount of communication in matrix multiplication and a number of other computations. Their bounds, however, assume a *shared-memory* computational model that does not model well existing computers. Their LPRAM model assumes that P processors, each with a private cache, are connected to a large shared memory. Furthermore, they assume that when a computation begins, the caches are empty, and that when the computation ends, the output must be returned to the main memory. Their analyses bound the amount of data that must be transferred between the shared main memory and the private caches. Their bound for matrix multiplication essentially quantifies the number of compulsory and capacity cache misses in a shared-memory multiprocessor. The LPRAM model does not model well systems in which all the memory is distributed among processing nodes. In particular, LPRAMs do not model well clusters of workstations and parallel computers with memory which is physically distributed, such as SGI Origin 2000s and IBM SPs.

This distinction between the LPRAM model and our model is highly relevant to matrix-multiplication lower bounds. Matrix multiplication is nearly always a subroutine in a larger computation. In a distributed-memory machine, the multiplicands are already distributed in some manner when the matrix-multiplication subroutine is called, and the product must be left distributed in the processors' local memories when it ends. Thus, the LPRAM bound, which essentially shows that each processor must access $\Omega(n^2/P^{2/3})$ input elements, is irrelevant to distributed-memory machines, since these elements may already reside in the processor's memory. The LPRAM lower bound does not depend on the amount of local memory; if data is allowed to be stored and perhaps replicated in local memories when the computation begins and ends, no communication may be necessary at all.

In contrast, our lower bounds allow for any initial data distribution of the input matrices, including data distributions that replicate input elements, as 3D algorithms do. That is, *our lower bounds do not even count the communication necessary to perform the replication of input elements!* (Except in Section 6 where we explicitly forbid replication to lower bound the communication across the bisection of the machine.) Our bounds also allow any distribution of the output matrix C. The only constraint that we place on the algorithm is that every element of C must reside in some processor's local memory (possibly several).

We state and prove lower bounds in this paper using concrete constants rather than asymptotic notation. We do so since in most of the bounds the amount of communication depends on three parameters: the size of the matrices n, the number of processors P, and the size of the local memory, M. Asymptotic notation makes it unclear which of the parameters must grow in order for the function to reach its asymptotic behavior. Also, the use of concrete constants clarifies the dependence of the amount of communication on each of the three parameters. The constants that appear in the statement of lemmas and theorems, however, were chosen to make the proofs as simple as possible; they were not chosen to be as tight as possible.

Some of our bounds assume that the matrices involved are square, whereas others apply to matrices of any shape. We restrict matrices to a square shape where we felt that bounds for rectangular matrices would complicate the statement of the results or the proofs too much.

Our analysis applies only to conventional matrix-multiplication algorithms. It does not apply to Strassen's algorithm [23] or other $o(n^3)$ algorithms. Lower bounds on the communication complexity of Strassen's and other unconventional algorithms are beyond the scope of this paper.

The paper is organized as follows. The next section, Section 2 presents a technical tool that underlies our unified approach to communication and cache-traffic lower bounds. Section 3 presents the basic memory-communication tradeoff that shows that lack of memory increases communication. Section 4 uses this provable tradeoff to analyze so-called 2-dimensional (2D) matrix-multiplication algorithms [6, 8, 9, 25]. These algorithms use only $\Theta(n^2/P)$ words of memory per processor, just a constant factor more than required to store the input and output (some of these algorithms use only $3n^2/P+o(n^2/P)$ words per processor). We show that the amount of communication that they perform per processor, $O(n^2/P^{1/2})$, is asymptotically optimal for this amount of memory. Section 5 uses a more sophisticated argument to show that so-called 3-dimensional (3D) algorithms are also optimal. 3D algorithms [3, 4, 8, 10, 13] replicate the input matrices $\Theta(P^{1/3})$ times, so they need $\Theta(n^2/P^{2/3})$ words of memory per processor. But this allows them to reduce the amount of communication to only $\Theta(n^2/P^{2/3})$ per processor. We show that this amount of communication is optimal for the amount of memory that is used. The argument in this case is somewhat more complex since the continuous tradeoff that we prove in Section 3 does not apply to $M = \Omega(n^2/P^{2/3})$. Section 6 proves that if no replication of input elements is allowed, then $\Omega(n^2)$ words must cross the bisection of the machine. Finally, Section 7 uses the basic lemma that underlies all of our results to prove a well-known lower bound on the number of cache misses (sometimes referred to as page faults or I/O's in the lower-bounds literature). The main point of Section 7 is to show how other kinds of lower bounds can be derived using our unified approach and how of I/O and communication lower bounds are related.

2. THE BASIC LEMMA

This section presents the technical lemma that underlies all the lower bounds in this paper. The lemma shows that a processor that accesses at most N elements of A, at most N elements of B, and contributes to the computation of at most N

elements of the product $C = AB$ can perform at most $O(N\sqrt{N})$ useful arithmetic operations.

Hong and Kung [12] proved a weaker form of this lemma. Their lemma only considers access to elements of A and B, not to contributions to elements of C, so it is too weak to be used in the proofs of our distributed-memory lower bounds. Also, Hong and Kung stated the lemma using asymptotic notation, whereas we state and prove the lemma using concrete constants.

But before we state and prove the lemma, we must define precisely the kinds of algorithms that it applies to and the way that we count elements of the product that are computed.

DEFINITION 2.1. A *conventional* matrix-multiplication algorithm is one that computes each element c_{ij} of C as a sum of products $a_{ik}b_{kj}$ for all k.

We need one more definition before we state and prove the lemma.

DEFINITION 2.2. A *useful multiplication* in a conventional matrix-multiplication algorithm is a scalar multiplication $a_{ik}b_{kj}$ that eventually contributes to the sum that forms c_{ij}.

We are now ready to prove the basic lemma. Note that it holds for matrices of any shape, as long as the shapes allow multiplication.

LEMMA 2.1. *Consider the conventional matrix multiplication $C = AB$, where A is m-by-k, B is k-by-n, and C is m-by-n. A processor that contributes to N_C elements of C and accesses N_A elements of A and N_B elements of B can perform at most*

$$\min\left((N_B + N_C)\sqrt{N_A}, (N_A + N_C)\sqrt{N_B}, (N_A + N_B)\sqrt{N_C}\right)$$

useful scalar multiplications.

Proof. We denote by S_A the set of elements (index pairs) of A that the processor accesses, by S_B the set of elements of B that the processor accesses, and by S_C the set of elements of C that the processor contributes to.

We first show that $(N_B + N_C)\sqrt{N_A}$ bounds the number of useful multiplications. We partition the rows of A into two sets: the set M_A contains the rows of A with at least $\sqrt{N_A}$ elements in S_A and F_A contains the rest of the rows. There are at most $\sqrt{N_A}$ rows in M_A.

Since each row of C is a product of the corresponding row in A and all of B, there can be at most $N_B\sqrt{N_A}$ useful multiplications involving rows in M_A. Since each element of C is a product of a row of A and a column of B and since each row in F_A has less than $\sqrt{N_A}$ elements in S_A, there can be at most $N_C\sqrt{N_A}$ useful multiplications involving rows in F_A.

A similar argument shows that $(N_A + N_C)\sqrt{N_B}$ bounds the number of useful multiplications. We partition the columns of B into a set M_B, consisting of columns with at least $\sqrt{N_B}$ elements in S_B, and a set F_B containing the rest of the columns.

Since each column of C is a product of A and the corresponding column of B, there can be at most $N_A\sqrt{N_B}$ useful multiplications involving rows in M_B. Since each element of C is a product of a row of A and a column of B, there can be at most $N_C\sqrt{N_B}$ useful multiplications involving rows in F_B.

Finally, we show that $(N_A + N_B)\sqrt{N_C}$ bounds the number of useful multiplications. We partition the rows of C into a sets M_C and F_C as we did for A.

Each row of C is a product of a row of A and all of B, so there can be at most $N_B\sqrt{N_C}$ useful multiplications involving rows in M_C . An element of A is used in the computation of elements from one row of C. If that row of C contains less than $\sqrt{N_C}$ elements in S_C, then the element of A us used in less than $\sqrt{N_C}$ multiplications. Hence, the number of useful multiplications involving rows of C in F_C is less than $N_A\sqrt{N_C}$. ■

3. THE MEMORY-COMMUNICATION TRADEOFF

This section proves a tradeoff between memory and communication in matrix-multiplication algorithms. The analysis shows that reducing the amount of memory forces an algorithm to perform more communication. We shall use this provable tradeoff in the next section to prove that 2D matrix-multiplication algorithms are asymptotically optimal for the amount of memory that they use. These algorithms use little extra memory beyond the storage required to store the matrices, and hence, they lie at extreme end of the memory-communication tradeoff. In Section 5 we shall extend the tradeoff to deal with larger memory sizes, which will enable us to prove that 3D algorithms are also asymptotically optimal for the amount of memory that they use.

Lemma 3.1. *Consider the conventional matrix multiplication $C = AB$, where A is m-by-k, B is k-by-n, and C is m-by-n, on a P-processor distributed-memory parallel computer. Consider a processor that has M words of local memory and which performs at least $\epsilon mnk/P$ scalar multiplications. The processor must send or receive at least*

$$\left\lfloor \frac{\epsilon}{4\sqrt{2}} \frac{mnk}{PM\sqrt{M}} \right\rfloor M$$

words. The amount of communication is also bounded by

$$\frac{\epsilon}{4\sqrt{2}} \frac{mnk}{P\sqrt{M}} - M \ .$$

Proof. We decompose the schedule of the computation on the processor into phases. Phase ℓ begins when total number of words sent and received so far by the processor is exactly ℓM. Thus, in each phase, except perhaps the last phase, the processor send and receives exactly M words.

The number N_A of elements of A that the processor may access during a phase is at most $2M$, since each one of these elements must reside in the processor's memory when the phase begins, or else it must have been received from another processor during the phase. The same argument shows that $N_B \le 2M$.

We define an element c_{ij} of the product C as *live* during a phase if: (1) the processor computes $a_{ik}b_{kj}$ for some k during the phase, and (2) a partial sum containing $a_{ik}b_{kj}$ either resides in the processor's memory at the end of the phase or is sent to another processor during the phase.

The number N_C of live elements of C during a phase is at most $2M$, since each live element either uses one word of memory at the end of the phase or it is sent to another processor.

Lemma 2.1 shows that the number of useful multiplications during a phase is at most

$$\begin{aligned} N_B\sqrt{N_A} + N_C\sqrt{N_A} &\leq 2M\sqrt{2M} + 2M\sqrt{2M} \\ &= 4\sqrt{2}M\sqrt{M} \ . \end{aligned}$$

The total number of scalar multiplications in the algorithm is mnk. Therefore, the number of full phases (that is phases in which exactly M words are sent and received) is at least

$$\left\lfloor \frac{\epsilon}{4\sqrt{2}} \frac{mnk}{PM\sqrt{M}} \right\rfloor ,$$

so the total amount of communication is at least

$$\left\lfloor \frac{\epsilon}{4\sqrt{2}} \frac{mnk}{PM\sqrt{M}} \right\rfloor M \ .$$

Since $\lfloor x \rfloor \geq x - 1$ for any positive x, we also have

$$\begin{aligned} \frac{\epsilon}{4\sqrt{2}} \frac{mnk}{PM\sqrt{M}} M &\geq \left(\frac{\epsilon}{4\sqrt{2}} \frac{mnk}{PM\sqrt{M}} - 1 \right) M \\ &= \frac{\epsilon}{4\sqrt{2}} \frac{mnk}{P\sqrt{M}} - M \ , \end{aligned}$$

which concludes the proof. ■

The lemma that we have just proved analyzes a parallel computation from the point of view of one processor. The next theorem takes a more global view. The running time is typically determined by the most heavily loaded processor. Therefore, by showing that at least one processor must perform a lot of communication, we essentially lower bound the amount of time that the algorithm must spend on communication.

Theorem 3.1. (Memory-Communication Tradeoff) *Consider the conventional matrix multiplication $C = AB$, where A is m-by-k, B is k-by-n, and C is m-by-n, on a P-processor distributed-memory parallel computer with M words of local memory per processor. At least one of the processors must send or receive at least*

$$\left\lfloor \frac{1}{4\sqrt{2}} \frac{mnk}{PM\sqrt{M}} \right\rfloor M$$

words. The amount of communication is also bounded by

$$\frac{mnk}{4\sqrt{2}P\sqrt{M}} - M \ .$$

Proof. At least one of the processors must perform mnk/P multiplications. The result follows from applying Lemma 3.1 with $\epsilon = 1$. ∎

4. A COMMUNICATION LOWER BOUND FOR ALMOST-IN-PLACE ALGORITHMS

We are not aware of algorithms whose performance can match asymptotically the lower bound in the memory-communication tradeoff for any value of M. That is, we are not aware of adaptive algorithms that can utilize any amount of available memory to reduce communication. But two classes of algorithms do match the lower bounds for specific values of M. At one end of the spectrum we have 2D algorithms that use little extra memory beyond that required to store the matrices. At the other hand of the spectrum, we have so-called 3D algorithms that use extra memory to replicate the input matrices and reduce communication. We now specialize Theorem 3.1 to 2D algorithms; Section 5 analyzes algorithms that use extra memory.

The first distributed-memory parallel matrix-multiplication algorithm is probably the one due to Cannon [6]. Cannon originally proposed the algorithm for the case $P = n^2$ and for a parallel computer whose processors are connected as a two-dimensional mesh. The generalization of Cannon's algorithm to larger block-distributed matrices is due to Dekel, Nassimi, and Sahni [8]. The algorithm has also been generalized to hypercube and other interconnection topologies. Fox, Otto and Hey [9] describe a different algorithm which, unlike Cannon's algorithm, uses broadcasts. Another 2D algorithm was proposed by van de Geijn and Watts [25]. This algorithm, called SUMMA, uses many broadcasts of small pieces of a matrix (smaller than the block stored on a processor), which allows the broadcasts to be pipelined and to occur concurrently with computation.

The storage required by 2D is proportional to the size of the matrices, so for square matrices the amount M of memory per processor should only be proportional to n^2/P. Clearly, $3n^2/P$ words per processor are necessary just to store the multiplicands and the product. Most 2D algorithms only need $o(n^2/P)$ temporary storage beyond the storage required to store the matrices (e.g., SUMMA). Simpler 2D algorithms, such as blocked implementations of Cannon's algorithm, require $2n^2/P$ additional words per processor, in order to store 2 blocks of A and 2 blocks of B. (This can be reduced to $o(n^2/P)$ by breaking each communication phase into many small messages, possibly at an increased overhead due to many small messages.) The next theorem shows that 2D algorithms are asymptotically optimal in terms of the amount of communication per processor: any algorithm that uses only $O(n^2/P)$ words of memory per processor must perform $\Omega(n^2\sqrt{P})$ communication per processor. In order to keep the theorem simple, we only state and prove it for square matrices.

Theorem 4.1. (2D Communication Lower-Bound) *Consider the conventional multiplication of two n-by-n matrices on a P-processor distributed-memory parallel computer, where each processor has $\mu n^2/P$ words of local memory. If*

$P \geq (8\sqrt{2})^2\mu^3$, then at least one of the processors must send or receive

$$\frac{1}{8\sqrt{2\mu}} n^2 \sqrt{P}$$

or more words.

Proof. By Theorem 3.1, the amount of communication in at least one of the processors is bounded from below by

$$\begin{aligned} \frac{n^3}{4\sqrt{2}P\sqrt{M}} - M &= \frac{n^3}{4\sqrt{2}P\sqrt{\mu n^2/P}} - \frac{\mu n^2}{P} \\ &= \frac{n^2}{4\sqrt{2\mu P}} - \frac{\mu n^2}{P} \\ &\geq \frac{n^2}{4\sqrt{2\mu P}} - \frac{\mu n^2}{8\sqrt{2}\mu\sqrt{\mu}\sqrt{P}} \\ &= \frac{n^2}{4\sqrt{2\mu P}} - \frac{n^2}{8\sqrt{2\mu P}} \\ &= \frac{n^2}{8\sqrt{2\mu P}} \,. \end{aligned}$$

The inequality relies on the fact that $\sqrt{P} \geq 8\sqrt{2}\mu^{3/2}$. ■

5. EXTENDING THE TRADEOFF BEYOND $M = \Theta(N^2/P^{2/3})$

The tradeoff that we analyzed in Section 3 fails to provide meaningful bounds when $M = \Theta(n^2/P^{2/3})$. In this regime, there may not even be a single full phase, in the sense of the proof of Lemma 3.1. Since the proof of Lemma 3.1 does not analyze the amount of communication in the last phase, it provides no useful bound in this case.

This section lower bounds the amount of communication that must be performed when $M = \Theta(n^2/P^{2/3})$. We show that in this regime the amount of communication per processor is $\Omega(n^2/P^{2/3})$.

This bound matches, asymptotically, the upper bounds that 3D algorithms achieve. 3D algorithms reduce communication over the more conventional 2D algorithms using replication. Such algorithms were first proposed by Berntsen [4] and by Aggarwal, Chandra, and Snir [3] at about the same time (Berntsen's paper was submitted to publication in 1988; The paper by Aggarwal, Chandra and Snir was presented at a conference in 1988). Essentially the same algorithm that was proposed in [3] was proposed again later, independently, by Gupta and Kumar [10] and by Johnsson [13]. Berntsen described a somewhat more complex algorithm. Although Dekel, Nassimi and Sahni [8] were perhaps the first to propose 3D algorithms, their algorithms are not communication efficient: the total number of words that are communicated is $\Theta(n^3)$.

Aggarwal, Chandra, and Snir also prove an $\Omega(n^2/P^{2/3})$ bound on the amount of communication per processor that matrix-multiplication algorithms must perform on a *shared-memory* parallel computer. As explained in the Introduction, their bound is irrelevant to matrix-multiplication on a distributed memory machine. Their bound relies on the private caches of the processors being empty when the computation begins. But on a distributed-memory machine, the matrices are typically already distributed when a matrix-multiplication subroutine is invoked.

The main result of this section hinges on the next lemma.

LEMMA 5.1. *Consider the conventional multiplication of two n-by-n matrices on a P-processor distributed-memory parallel computer. Consider a processor that has $\mu n^2/P^{2/3}$ words of local memory and which performs at least $\epsilon n^3/P$ scalar multiplications. Let*

$$\delta = \min\left(\mu, \frac{1}{32}\cdot\frac{\epsilon^2}{\mu^2}\right) .$$

For any

$$P \geq 1024\sqrt{2}\frac{\mu^6}{\epsilon^3} ,$$

the processor must send or receive at least

$$\delta\frac{n^2}{P^{2/3}}$$

words.

Proof. Let π be the number of multiplications that the processor performs. Using Lemma 2.1 and its notation, we have

$$N_B\sqrt{N_A} + N_C\sqrt{N_A} \geq \pi \geq \epsilon\frac{n^3}{P} \tag{1}$$

$$N_A\sqrt{N_B} + N_C\sqrt{N_B} \geq \pi \geq \epsilon\frac{n^3}{P} \tag{2}$$

$$N_A\sqrt{N_C} + N_B\sqrt{N_C} \geq \pi \geq \epsilon\frac{n^3}{P} \tag{3}$$

If $N_A \geq (\mu+\delta)n^2/P^{2/3}$, then the theorem holds, since only $\mu n^2/P^{2/3}$ elements of A can reside in the processor's local memory when the algorithm begins, so the rest must be received from other processors. The same argument holds when $N_B \geq (\mu+\delta)n^2/P^{2/3}$. If $N_C \geq (\mu+\delta)n^2/P^{2/3}$, then the theorem holds again, since the processor must send contributions to at least $\delta n^2/P^{2/3}$ elements of C to other processors.

If both N_A, N_B, and N_C are smaller than $(\mu+\delta)n^2/P^{2/3}$, then we claim that all three quantities must be larger than

$$\frac{1}{4}\left(\frac{\epsilon}{\mu+\delta}\right)^2\frac{n^2}{P^{2/3}} .$$

From (1) and the fact that both N_A and N_B are smaller than $(\mu+\delta)n^2/P^{2/3}$, we have

$$2\sqrt{N_C}(\mu+\delta)\frac{n^2}{P^{2/3}} > N_A\sqrt{N_C} + N_B\sqrt{N_C} \geq \epsilon\frac{n^3}{P} ,$$

so

$$2\sqrt{N_C} > \frac{\epsilon}{\mu+\delta}\frac{n}{P^{1/3}} ,$$

and finally

$$N_C > \frac{1}{4}\left(\frac{\epsilon}{\mu+\delta}\right)^2 \frac{n^2}{P^{2/3}} . \tag{4}$$

An identical argument shows that the expression in (4) also lower bounds N_A and N_B.

The number of elements of C that the processor can compute on its own without any contributions from other processors is small. For each such c_{ij}, the processor must access the entire ith row of A and the entire jth column of B. If N_A and N_B are smaller than $(\mu+\delta)n^2/P^{2/3}$, then the processor can compute at most

$$\left((\mu+\delta)\frac{n^2}{P^{2/3}}/n\right) \times \left((\mu+\delta)\frac{n^2}{P^{2/3}}/n\right) = (\mu+\delta)^2\frac{n^2}{P^{4/3}} \tag{5}$$

elements of C on its own.

Suppose that the processor participates in the computation of c_{ij} but does not compute it on its own. If c_{ij} resides on our processor at the end of the computation, then our processor must have received a contribution to c_{ij} from at least one other processor. If c_{ij} resides on another processor, our processor must have sent its own contribution to some other processor. Either way, one word of data must either be received or sent for each such c_{ij}.

Therefore, the processor must send or receive at least

$$\frac{1}{4}\left(\frac{\epsilon}{\mu+\delta}\right)^2 \frac{n^2}{P^{2/3}} - (\mu+\delta)^2\frac{n^2}{P^{4/3}} = \left(\frac{1}{4}\left(\frac{\epsilon}{\mu+\delta}\right)^2 - \frac{(\mu+\delta)^2}{P^{2/3}}\right)\frac{n^2}{P^{2/3}}$$

words to participate in the computation of the elements of C (subtracting (5) from (4)). By the hypotheses on P and δ, we have

$$\begin{aligned}
\frac{1}{4}\left(\frac{\epsilon}{\mu+\delta}\right)^2 - \frac{(\mu+\delta)^2}{P^{2/3}} &\geq \frac{1}{4}\left(\frac{\epsilon}{2\mu}\right)^2 - \frac{(\mu+\delta)^2}{P^{2/3}} \\
&= \frac{1}{16}\left(\frac{\epsilon}{\mu}\right)^2 - \frac{(\mu+\delta)^2}{P^{2/3}} \\
&\geq \frac{1}{16}\left(\frac{\epsilon}{\mu}\right)^2 - \frac{(2\mu)^2}{P^{2/3}} \\
&\geq \frac{1}{32}\left(\frac{\epsilon}{\mu}\right)^2 \\
&\geq \delta .
\end{aligned}$$

The first two inequalities are true since $\delta < \mu$. The third holds since $P \geq 1024\sqrt{2}\mu^6/\epsilon^3$, which implies that $P^{2/3} \geq 128\mu^4/\epsilon^2$, which in turn implies that

$$\frac{(2\mu)^2}{P^{2/3}} \leq \frac{1}{32}\left(\frac{\epsilon}{\mu}\right)^2 .$$

This shows that the number of words that must be sent or received is at least $\delta n^2/P^{2/3}$, as claimed. ■

We can now prove the main result of this section. Its proof is essentially the same as that of Theorem 3.1 and is omitted.

THEOREM 5.1. (3D COMMUNICATION LOWER BOUND) *Consider the conventional multiplication of two n-by-n matrices on a P-processor distributed-memory parallel computer, where each processor has $\mu n^2/P^{2/3}$ words of local memory. For any*

$$P \geq 1024\sqrt{2}\mu^6 \ ,$$

at least one processor must send or receive

$$\min\left(\mu, \frac{1}{32\mu^2}\right)\frac{n^2}{P^{2/3}}$$

words or more.

6. BISECTION-BANDWIDTH BOUNDS

This section analyzes the amount of data that must cross the bisection of a distributed-memory parallel computer. That is, we split the memory-processor nodes into two subsets and lower bound the amount of data that must cross any cut that separates the subsets in the communication network of the computer. We assume that each element of the input matrices is stored exactly once in the distributed memory of the machine and that the input is evenly split between the subsets. Indeed, if we allow replication of the input matrices, the output can be computed without any communication across the bisection. For example, most 3D algorithms perform no communication across some bisection cuts in the network following the initial data replication phase.

Another way to derive lower bounds on the communication across cuts is to apply the lower bounds in Section 4 and 5 to groups of processors. These bounds also bound the amount of communication that a group of p processors must perform with the other $P-p$ processors in the machine. This communication must be transmitted on any edge cut in the communication network between the two processor groups. Hence, we can bound the amount of communication that must traverse cuts in the network.

This technique, however, is unlikely to provide useful bounds for large p (say $p = P/2$). First, some of our results (Theorems 4.1 and 5.1) only hold for large P. Second, other results (Theorem 3.1) provides useless bounds for $P = 2$, since they only guarantee that the amount of communication is greater than 0 or a negative number.

Therefore, we need a specific bound on the communication across bisection cuts. The theorem below assumes that A and B are evenly distributed. The proof can be easily modified to show that same asymptotic behavior also applies when each of the two processor subsets initially stores at most μn^2 elements of A and μn^2

elements of B, for any constant $\mu < 1/\sqrt{2}$. For $\mu \geq 1/\sqrt{2}$, a more complex proof would be required.

THEOREM 6.1. *Consider the conventional multiplication of two n-by-n matrices on a P-processor distributed-memory parallel computer, where each input element is initially stored in the local memory of exactly one processor. At least*

$$\frac{\sqrt{13}-3}{4} n^2 \approx 0.1514 n^2$$

words must be transferred across any cut that splits the input distribution of the multiplicands A and B evenly.

Proof. Let

$$\delta = \frac{\sqrt{13}-3}{4} \ .$$

Consider a cut in the communication network that splits the nodes into two subsets, each holding exactly $n^2/2$ elements of A and $n^2/2$ elements of B.

If more than δn^2 elements of A or more than δn^2 elements of B are transferred across the cut, The theorem holds.

Otherwise, we claim that each part of the machine can compute at most

$$\left(\frac{1}{2}+\delta\right)^2 n^2$$

elements of C on its own (that is, by summing products $a_{ik}b_{kj}$ that are all computed locally). Computing each such element requires access an entire row of A and an entire column of B. If at most δn^2 elements of A and B cross the cut, each part has access to at most $\left(\frac{n^2}{2}+\delta n^2\right)/n$ rows of A and columns of B, so it can compute at most

$$\left(\frac{1}{2}+\delta\right)^2 n^2$$

elements of C on its own.

Hence, the other

$$n^2 - 2\left(\frac{1}{2}+\delta\right)^2 n^2 = \left(-2\delta^2 - 2\delta + \frac{1}{2}\right) n^2$$

elements of C must be computed by the two parts of the machine together, which means that at least that many words must cross the cut. Since $\delta = (\sqrt{13}-3)/4$, we have $-2\delta^2 - 2\delta + 1/2 = \delta$, so the theorem holds. ■

7. THE I/O LOWER BOUND

The next theorem and the corollary that follows it show how to use the Lemma 2.1 to bound the number of compulsory and capacity cache misses in matrix multiplication. These results lower bound the number of words that must be transferred between a slow memory and a fast cache when arithmetic operations can only access data in the cache.

The results themselves are not new, but they show how the proof technique that we use for the parallel communication bounds can be applied to the analysis of cache misses. Specifically, the bounds that we prove here are asymptotically the same as those proved by Hong and Kung [12] and again by Toledo [24]. Our bounds, however, specify constants, unlike Hong and Kung's result which is stated using asymptotic notation. The constants here are slightly stronger than those given in [24], but the proof technique is similar. In effect, we are using Toledo's proof technique but the use of Lemma 2.1 simplifies the structure of the proof.

The lower bounds use a lax memory-system model that is equivalent to Hong and Kung's red-blue pebble game. Therefore, they apply to any cache organization. The lower bounds also match, asymptotically, the performance of recursive matrix multiplication and of blocked matrix multiplication, assuming that the block size is chosen appropriately and when the cache is fully associative and uses the LRU replacement policy.

Matrix-multiplication algorithms whose asymptotic performance matches the lower bound are as old as computers. Rutledge and Rubinstein [22, 21] described the library of blocked matrix subroutines that they designed (together with Herbert F. Mitchell) and implemented for the UNIVAC, a first-generation computer that became operational in 1952. McKeller and Coffman [20] provided the first rigorous analysis of data reuse in matrix computations, including matrix multiplications. They showed that blocked algorithms transferred fewer words between fast and slow memory than algorithms that operated by row or by column. High-quality implementations of I/O-efficient matrix-multiplication algorithms are widely available and used [2, 1, 5, 7, 11, 14, 17, 15, 16, 18, 19, 26]

The proof of the next theorem is very similar to the proof of Lemma 3.1.

THEOREM 7.1. *Consider the conventional multiplication of two n-by-n matrices on a computer with a large slow memory and a fast cache that can contain M words. Arithmetic operations can only be performed on words that are in the cache. The number of words that are moved between the slow memory and the fast cache is at least*

$$\left\lfloor \frac{n^3}{4\sqrt{2}M\sqrt{M}} \right\rfloor M$$

words. The number of words that are moved is also bounded by

$$\frac{n^3}{4\sqrt{2}\sqrt{M}} - M \, .$$

Proof. We decompose the schedule of the computation into phases. Phase ℓ begins when total number of words moved between the memory and the cache is exactly ℓM. Thus, in each phase, except perhaps the last phase, exactly M words are transferred between memory and the cache.

The number N_A of elements of A that the processor may operate upon during a phase is at most $2M$, since each one of these elements must either reside in the cache when the phase begins, or it must be read into the cache during the phase. The same argument shows that $N_B \leq 2M$.

We define an element c_{ij} of the product C as *live* during a phase if the processor computes $a_{ik}b_{kj}$ for some k during the phase and if a partial sum containing $a_{ik}b_{kj}$ either resides in the cache at the end of the phase or is written to slow memory during the phase.

The number N_C of live elements of C during a phase is at most $2M$, since each live element either uses one word of cache at the end of the phase or it is written to slow memory.

Lemma 2.1 shows that the number of useful multiplications during a phase is at most

$$\begin{aligned} N_B\sqrt{N_A} + N_C\sqrt{N_A} &\leq 2M\sqrt{2M} + 2M\sqrt{2M} \\ &= 4\sqrt{2}M\sqrt{M} \ . \end{aligned}$$

Therefore, the number of full phases (that is, phases in which exactly M words are transferred) is at least

$$\left\lfloor \frac{n^3}{4\sqrt{2}M\sqrt{M}} \right\rfloor ,$$

so the total number of words that are transferred is at least

$$\left\lfloor \frac{n^3}{4\sqrt{2}M\sqrt{M}} \right\rfloor M \ .$$

Since $\lfloor x \rfloor \geq x - 1$ for any positive x, we also have

$$\begin{aligned} \frac{n^3}{4\sqrt{2}M\sqrt{M}} M &\geq \left(\frac{n^3}{4\sqrt{2}M\sqrt{M}} - 1 \right) M \\ &= \frac{n^3}{4\sqrt{2}\sqrt{M}} - M \ . \end{aligned}$$

which concludes the proof. ■

The next corollary shows that when the matrices are several times larger than the cache, the number of cache misses is proportional to $n^3/\sqrt{M}$. The constants in the corollary are essentially an example; by picking a stronger bound on the size of the matrices relative to the cache we could have proved a stronger bound on the number of cache misses.

COROLLARY 7.1. (THE I/O LOWER BOUND [12]) *Under the conditions of Theorem 7.1, and assuming that $M \leq n^2/128^{2/3} < n^2/5.039$, the number of words that must be transferred to and from the cache is at least*

$$\frac{n^3}{8\sqrt{2}\sqrt{M}} = \Theta\left(\frac{n^3}{\sqrt{M}} \right) \ .$$

Proof. If $M \le n^2/128^{1/3}$ then $\sqrt{M} \le n/128^{1/6}$ the number of words that must be transferred is at least

$$\begin{aligned}
\frac{n^3}{4\sqrt{2}\sqrt{M}} - M &\ge \frac{n^3}{4\sqrt{2}\sqrt{M}} - \frac{n^2}{128^{1/3}} \\
&= \frac{n^3}{4\sqrt{2}\sqrt{M}} - \frac{n^3}{128^{1/3}n} \\
&\ge \frac{n^3}{4\sqrt{2}\sqrt{M}} - \frac{n^3}{128^{1/3} \cdot 128^{1/6}\sqrt{M}} \\
&= \frac{n^3}{4\sqrt{2}\sqrt{M}} - \frac{n^3}{8\sqrt{2}\sqrt{M}} \\
&= \frac{n^3}{8\sqrt{2}\sqrt{M}} \, .
\end{aligned}$$

■

REFERENCES

1. R. C. Agarwal, F. G. Gustavson, and M. Zubair. Exploiting functional parallelism of POWER2 to design high-performance numerical algorithms. *IBM Journal of Research and Development*, 38(5):563–576, 1994.

2. R. C. Agarwal, F. G. Gustavson, and M. Zubair. Improving performance of linear algebra algorithms for dense matrices using algorithmic prefetch. *IBM Journal of Research and Development*, 38(3):265–275, 1994.

3. Alok Aggarwal, Ashok Chandra, and Marc Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71:3–28, 1990.

4. Jarle Bernsten. Communication efficient matrix multiplication on hypercubes. *Parallel Computing*, 12:335–342, 1989.

5. J. Bilmes, K. Asanovic, C. W. Chin, and J. Demmel. Optimizing matrix multiply using PHIPAC: a portable, high-performance, ANSI C coding methodology. In *Proceedings of the International Conference on Supercomputing*, Vienna, Austria, 1997.

6. L. E. Cannon. *A Cellular Computer to Implement the Kalman filter algorithm*. PhD thesis, Montana State University, 1969.

7. M. J. Dayde and I. S. Duff. A blocked implementation of level 3 BLAS for RISC processors. Technical Report RT/APO/96/1, ENSEEIHT-IRIT, France, 1996.

8. Eliezer Dekel, David Nassimi, and Sartaj Sahni. Parallel matrix and graph algorithms. *SIAM Journal on Computing*, 10:657–675, 1981.

9. G. C. Fox, S. W. Otto, and A. J. G. Hey. Matrix algorithms on a hypercube. i: Matrix multiplication. *Parallel Computing*, 4:17–31, 1987.

10. Anshul Gupta and Vipin Kumar. The scalability of matrix multiplication algorithms on parallel computers. Technical Report TR 91-54, Department of Computer Science, University of Minnesota, 1991. Available online from `ftp://ftp.cs.umn.edu/users/kumar/matrix.ps`. A short version appeared in *Proceedings of 1993 International Conference on Parallel Processing*, pages III-115–III-119, 1993.

11. Greg Henry, Pat Fay, Ben Cole, and Timothy G. Mattson. The performance of the Intel TFLOPS supercomputer. *Intel Technical Journal*, 98(1), 1998. available online at `http://developer.intel.com/technology/itj/`.

12. J.-W. Hong and H. T. Kung. I/O complexity: the red-blue pebble game. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 326–333, 1981.

13. S. Lennart Johnsson. Minimizing the communication time for matrix multiplication on multiprocessors. *Parallel Computing*, 19:1235–1257, 1993.

14. S. Lennart Johnsson and Luis F. Ortiz. Local basic linear algebra subroutines (LBLAS) for the Connection Machine system CM-200. *International Journal of Supercomputing Applications*, 7:322–350, 1993.

15. B. Kågström, P. Ling, and C. Van Loan. High performance GEMM-based level-3 BLAS: Sample routines for double precision real data. In M. Durand and F. El Dabaghi, editors, *High Performance Computing II*, pages 269–281, Amsterdam, 1991. North-Holland.

16. B. Kågström, P. Ling, and C. Van Loan. Portable high performance GEMM-based level 3 BLAS. In *Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pages 339–346, Philadelphia, 1993.

17. B. Kågström and C. Van Loan. GEMM-based level-3 BLAS. Technical Report CTC-91-TR47, Department of Computer Science, Cornell University, 1989.

18. Chandrika Kamath, Roy Ho, and Dwight P. Manley. DXML: a high-performance scientific subroutine library. *Digital Technical Journal*, 6(3):44–56, 1994.

19. David Kramer, S. Lennart Johnsson, and Yu Hu. Local basic linear algebra subroutines (LBLAS) for the CM-5/5E. *International Journal of Supercomputing Applications*, 10:300–335, 1996.

20. A. C. McKeller and E. G. Coffman, Jr. Organizing matrices and matrix operations for paged memory systems. *Communications of the ACM*, 12(3):153–165, 1969.

21. J. Rutledge and H. Rubinstein. High order matrix computation on the UNIVAC. Presented at the meeting of the Association for Computing Machinery; Copies available from Sivan Toledo, May 1952.

22. Joseph Rutledge and Harvey Rubinstein. Matrix algebra programs for the UNIVAC. Presented at the Wayne Conference on Automatic Computing Machinery and Applications; Copies available from Sivan Toledo, March 1951.

23. V. Strassen. Gaussian elimination is not optimal. *Numererische Mathematik*, 13:354–355, 1969.

24. Sivan Toledo. A survey of out-of-core algorithms in numerical linear algebra. In James M. Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 161–179. American Mathematical Society, 1999.

25. Robert van de Geijn and Jerrell Watts. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 9:255–274, 1997.

26. R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. Technical report, Computer Science Department, University Of Tennessee, 1998. available online at www.netlib.org/atlas.