MIT/LCS/TM-102

LOWER BOUNDS ON INFORMATION TRANSFER IN DISTRIBUTED COMPUTATIONS

Harold Abelson

April 1978

MIT/LCS/TM-102

Lower Bounds on Information Transfer in Distributed Computations

by

Harold Abelson

April 1978

Massachusetts Institute of Technology
Laboratory for Computer Science

Cambridge Massachusetts 02139

Lower Bounds on Information Transfer in Distributed Computations

Harold Abelson

Department of Electrical Engineering and Computer Science

Massachusetts Institute of Technology

Abstract

We derive a lower bound on the interprocessor information transfer required for computing a function in a distributed network. The bound is expressed in terms of the function's derivatives, and we use it to exhibit functions whose computation requires a great deal of interprocess communication. As a sample application, we give lower bounds on information transfer in the distributed computation of some typical matrix operations.

Keywords: Computational complexity, distributed processing.

This report was prepared with the support of the National Science Foundation under NSF grant no. MCS77-19754.

Lower Bounds on Information Transfer in Distributed Computations

Harold Abelson

It is evident that traditional notions of computational complexity, such as the number of primitive operations or memory cells required to compute functions, do not form an adequate framework for assessing the complexity of computations carried out in distributed networks. Even in the relatively straightforward situation of memoryless processors arranged in highly structured configurations, Gentleman [3] has shown that data movement, rather than arithmetic operations, is often the significant factor in the performance of parallel computations. And for more general kinds of distributed processing, involving arbitrary network configurations and distributed data bases, the situation is correspondingly more complex.

This paper addresses the problem of measuring computational complexity in terms of the interprocess communication required when a computation is distributed among a number of processors. More precisely, we model the distributed computation of functions which depend upon large amounts of data by assuming that the data is partitioned into disjoint subsets, and that a processor is assigned to each subset. Each processor (which we can think of as a node in a computational network) computes some values based on its own data, and transmits these values to other processors, which are able to use them in subsequent local computations. This "compute locally and share information" procedure is repeated over and over until finally some (predetermined) processor outputs the value of the desired function. In measuring the "complexity" of such computations we will be concerned, not with the complexity of the individual local computations, but rather with the total information transfer, i.e., the total number of values which must be transmitted between processors.

We derive a lower bound on the total information transfer required for computing a function in a distributed network. The bound is expressed in terms of the function's derivatives, and we use it to exhibit functions whose computation requires a great deal of

interprocess communication. As a sample application, we give lower bounds on information transfer in the distributed computation of some typical matrix operations.

1. Information transfer with one-way communication

We begin by reviewing a result of [1] which gives bounds on information transfer in networks which allow only one-way communication. Suppose $X=(x_1,x_2,\ldots,x_a)$ and $Y=(y_1,y_2,\ldots,y_b)$ are collections of real variables, and that $\Phi: X\times Y\to\mathbb{R}$ is a continuously differentiable real-valued function. (Throughout this section we assume that all functions are real-valued and continuously differentiable, i.e., with continuous first derivatives. Such functions are standardly referred to as "C^1 functions.") From the distributed computation perspective outlined above, we can regard X as the data accessed by some subset of the processors in a network, and Y as the rest of the data, including that accessed by the predetermined processor which is to output the value of Φ. Then if we allow communication only from X to Y and not from Y to X, computing Φ with total information transfer N, i.e., transmitting N values from X to Y is equivalent to representing Φ in the form

$$\Phi(X,Y)=g(f_1(X),f_2(X),\ldots,f_N(X),Y)$$

where the $f_i(X)$ are functions of X alone. We can give a necessary and sufficient condition for the existence of such a representation. For any particular values X_0 for the variables X let $\frac{\partial\Phi}{\partial x_j}\Big|_{X_0}$ denote the real-valued function of Y defined by

$$\frac{\partial\Phi}{\partial x_j}\Big|_{X_0}(Y)=\frac{\partial\Phi}{\partial x_j}(X_0,Y)$$

Then we have:

Theorem 1 A C^1 function $\Phi: X \times Y \to \mathbb{R}$ can be represented in the form

$$\Phi(X,Y) = g(f_1(X), f_2(X), \ldots, f_N(X), Y)$$

where $f_i: X \to \mathbb{R}$ and $g: \mathbb{R}^{N+b} \to \mathbb{R}$ are C^1 functions, if and only if, given any value X_0 for X, at most N of the functions $\frac{\partial \Phi}{\partial x_j}\Big|_{X_0}$ are linearly independent.

(For the proof, see [1], [2].)

We can use Theorem 1 to demonstrate that allowing two-way communication between the processors in a network can drastically reduce the information transfer required to compute certain functions. For example, let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ and let Φ be

$$\Phi(X, Y) = \sum_{k=1}^{n} y_k x_1^k + \sum_{k=1}^{n} x_k y_1^k$$

Then
$$\frac{\partial f}{\partial x_1} = y_1 + \sum_{k=1}^{n} k y_k x_1^{k-1}$$

and
$$\frac{\partial f}{\partial x_k} = y_1^k \qquad (k \neq 1)$$

give n linearly independent functions of Y when we substitute any particular values for X. Hence any network which permits only one-way communication from X to Y cannot compute Φ with total information transfer less than n. Similarly, allowing only one-way communication from Y to X will also require information transfer n.

With two-way communication, however, we can compute Φ with information transfer 3 as follows: X sends to Y the value of x_1 and Y sends X the value of y_1. Once X knows y_1, it can compute the second term in the expression for Φ, and transmit this to Y. Then Y, knowing both x_1 and the second term, can compute Φ. Thus, by letting n grow large, we can exhibit functions whose computations require arbitrarily large information transfer when only one-way

communication (either from X to Y or from Y to X) is permitted, but which can be computed with information transfer 3 using two-way communication.

2. Two-way communication

We consider a general model of distributed computation with two-way communication. Suppose as before that there are two sets of real variables X and Y. (As above, we can think of X as the data accessed by some collection of the processors in a network and Y as the rest of the data.) We formulate the following description of a multi-stage distributed computation of a function $\Phi(X,Y)$: At the first stage, the X processors compute some number, say a_1, real-valued functions f_i^1 of the variables x_j, and transmit these values to Y. (We will henceforth omit the subscripts on the f's and refer to the vector of functions $f^1(X)$.) Simultaneously, the Y processors transmit to X the values of b_1 functions $g^1(Y)$. At the second stage, X transmits to Y the values of a_2 functions $f^2(X,g^1)$, which depend on both X and the values g^1 received from Y at the previous stage. Likewise, Y transmits to X the values of b_2 functions $g^2(Y,f^1)$. In general, at the kth stage, X sends to Y the values of a_k functions $f^k(X,g^1,g^2,\ldots,g^{k-1})$ which depend on X and the values received from Y in previous stages; and Y transmits to X the values of b_k similar functions g^k.

We wish to characterize those functions which can be computed by Y, say, at the rth stage. Notice first of all that an r-stage function g_i^r depends in general on all previous f^k ($k = 1,\ldots,r-1$) and on g^k ($k = 1,\ldots,r-2$), but not on g^{r-1} (since the value of g^{r-1} is not used in computing f^{r-1}). So in this terminology, a 1-stage function is simply a function of Y alone, and a 2-stage function is a function computed with 1-way communication from X to Y. We also see that the total information transfer used in computing an r-stage function g^r is

$$N = \sum_{i=1}^{r-1} a_i + \sum_{i=1}^{r-2} b_i$$

We will assume from here on that all functions computed at each stage have continuous

1st and 2nd derivatives. (Such functions are commonly called "C^2 functions.") Under this assumption, we obtain the following lower bound on the information transfer required in a multi-stage distributed computation.

Theorem 2 Let $\Phi: X \times Y \to \mathbb{R}$ be a C^2 function and let R be the rank of the matrix of second order partial derivatives

$$\Delta_{ij} = \frac{\partial^2 \Phi}{\partial x_i \partial y_j}$$

Then any multi-stage distributed computation of Φ must have total information transfer at least R between X and Y (assuming that the functions computed at each stage are all C^2).

The proof of Theorem 2 is given below in Section 4.

The "inner product" provides a simple example of a function whose distributed computation requires maximal information transfer. That is, let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ and take

$$\Phi(X,Y) = \sum_{i=1}^{n} x_i y_i$$

The corresponding matrix Δ of 2nd derivatives is the $n \times n$ identity matrix, which has rank n. Hence, by Theorem 2, any multi-stage distributed computation for Φ must have information transfer at least n between X and Y. Notice that this is maximal, in the sense that any function can be computed with information transfer equal to the number of variables in X.

Another example of a function which requires maximal information transfer is the determinant of an $n \times n$ matrix, when the different columns are accessed by different local processors. More precisely, if $M = [m_{ij}]$, let X be the variables in, say, the 1st column of M, i.e., $x_k = m_{1k}$, let Y be the rest of the variables and take $\Phi(X,Y) = \det(M)$. We leave it to the reader to verify that the corresponding $n \times (n^2-n)$ matrix Δ does indeed have rank n. (Here is an outline of a proof: Use the fact that the derivative of the determinant function with respect to any

element m_{ij} is equal to $(-1)^{i+j}$ times the cofactor of m_{ij}. Then, by choosing matrices M for which the cofactors of elements in the 1st and 2nd columns have appropriate values, one can demonstrate that, for n even, there are choices of M for which the $n \times n$ submatrix consisting of the first n columns of Δ already has rank n. Similarly, for n odd, one can construct matrices for which the first $2n$ columns of Δ form an $n \times 2n$ matrix of rank n.)

Finally, consider the solution of systems of linear equations when the columns of the coefficient matrix are accessed at different processors. Let X be the variables in the kth column of an $n \times n$ matrix M and let Y be the rest of the elements of M. For any non-zero vector b let $\Phi_k(b)$ be the function (of X and Y) which is equal to the kth component of the solution to the system of equations $Mz = b$. Then we have:

Lemma 2.1 For any non-zero vector b, the matrix $\dfrac{\partial^2 \Phi_k(b)}{\partial x_i \partial y_j}$ has rank $n-1$.

Therefore, computing the kth component of the solution to a system of linear equations requires information transfer at least $n-1$ between the kth column and the rest of the matrix. (The proof of Lemma 2.1 is given in Section 5 below.)

3. Computations in Networks

In applying Theorem 2 to the analysis of specific network computations, it is important to realize that the bounds on data transfer are valid for any partition of the network into two pieces. To illustrate this point we derive lower bounds on total data transfer in distributed implementations of three typical matrix computations.

As a first example, suppose that we have a computational network consisting of n processors $X_1, \ldots, X_n$ and one additional processor Y, and that each processor is directly connected to every other processor. We wish to use this network to compute the determinant of an $n \times n$ matrix by letting each X_i access a single column of the matrix and having the

determinant output at Y. Observe that n^2 is an upper bound for the information transfer in such a computation, since we can always first transmit all the data to Y and do all the computation there. For a lower bound on information transfer, we get:

Corollary 1 For any determinant computation using the above network configuration, the total information transfer over the network must be greater than $\frac{n^2}{2}$. (Assuming that the local computations performed by each processor are C^2 functions.)

Proof: Let N_{ij} be the direct information transfer between X_i and X_j, i.e., the number of values sent directly from X_i to X_j plus the number of values sent directly from X_j to X_i during the course of the computation. (Set $N_{ii}=0$ for convenience.) Let N_{iY} be the information transfer between X_i and Y. Then the total information transfer over the network is given by

$$N = \sum_{i=1}^{n} N_{iY} + \sum_{i<j} N_{ij} \qquad (1)$$

Let Net_i be the given network, only viewed as partitioned into two pieces: X_i and the rest of the network. Then for each i, Net_i performs the determinant computation discussed in Section 2 above, where it was explained that the total information transfer must be at least n. Hence we have

$$\text{Information transfer for } \text{Net}_i = N_{iY} + \sum_{j=1}^{n} N_{ij} \geq n$$

So summing this inequality over all i gives

$$\sum_{i=1}^{n} N_{iY} + 2\sum_{i<j} N_{ij} \geq n^2$$

and combining this with Equation (1) gives the desired result.

As a second illustration, consider the multiplication of $n \times n$ matrices in the following distributed configuration: There are n processors X_i, all interconnected. At the outset of the

computation, each processor has access to one row of a matrix M and one column of a matrix N. At the conclusion of the computation, X_i is to output the ith row of the product MN.

Corollary 2 Computing a matrix product in the above configuration requires total information transfer at least $\frac{n^2}{2}$. (Assuming that the local computations are C^2.)

Proof: Let N_{ij} and Net_i be as in the proof of Corollary 1. Each X_i must compute elements of the matrix product which are inner products of data local to X_i and data accessed by the rest of the network. Hence, as shown in Section 2, the information transfer between X_i and the rest of the network, i.e., the information transfer for the network Net_i, must be at least n:

$$\text{Information transfer for } \text{Net}_i = \sum_{j=1}^{n} N_{ij} \geq n$$

The proof is completed by summing this inequality over all i and using the fact that the total information transfer for the entire network is

$$N = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} N_{ij}$$

Finally, consider the distributed solution to a system of linear equations $Mz = b$ in the same configuration as above. Each processor X_i is given access to the ith column of M and is to output the ith entry of z. (We assume that every processor has access to all entries of b.)

Corollary 3: For any non-zero vector b, solving the system of linear equations $Mz = b$ in the above distributed configuration requires total information transfer at least $\frac{n(n-1)}{2}$.

The proof is essentially identical to that of Corollary 2, with the bound on the information transfer for Net_i given by Lemma 2.1.

4. Proof of Theorem 2

The proof of Theorem 2 is based on the following result:

Lemma 4.1 If Φ is computed by an r-stage computation, then for all i and j

$$\frac{\partial^2 \Phi}{\partial x_i \partial y_j} = \sum_{p=1}^{r-1} \sum_{q=1}^{a_p} A_j^{pq} \frac{\partial f_q^p}{\partial x_i} + \sum_{s=1}^{r-2} \sum_{t=1}^{b_s} B_i^{st} \frac{\partial g_t^s}{\partial y_j}$$

where each A_j^{pq} is a function of X and Y which depends only on j, p and q, and each B_i^{st} is a function of X and Y which depends only on i, s and t.

Proof that Lemma 4.1 implies Theorem 2:

Let N be the total information transfer in the computation for Φ. As remarked in Section 2 we have

$$N = \sum_{i=1}^{r-1} a_i + \sum_{i=1}^{r-2} b_i$$

which is equal to the number of terms in the large sum given in the conclusion of the Lemma. Consolidating these terms according to their dependence on i and j yields

$$\frac{\partial^2 \Phi}{\partial x_i \partial y_j} = \sum_{m=1}^{N} \Gamma_{im} \Delta_{mj}$$

where Γ_{im} is some expression which depends on i and m, and Λ_{mj} depends on j and m. In other words, if there are $|X|$ variables x_k and $|Y|$ variables y_k, then the $|X| \times |Y|$ matrix M_{ij} is the product of the $|X| \times N$ matrix Γ and the $N \times |Y|$ matrix Λ. Hence the rank of Δ can be at most N.

Proof of Lemma 4.1: The proof proceeds by induction. We first check the case of 2-stage functions, r=2. (The case of 1-stage functions is trivial since there is no information transfer.) A 2-stage function Φ has the form $\Phi(X,Y)=(f^1(X),Y)$ where $f^1=(f^1_1,\ldots,f^1_{a_1})$ is a vector of functions of X. Differentiating with respect to x_i yields

$$\frac{\partial \Phi}{\partial x_i} = \sum_{q=1}^{a_1} \frac{\partial \Phi}{\partial f^1_q} \frac{\partial f^1_q}{\partial x_i}$$

Since the $\frac{\partial f^1_q}{\partial x_i}$ are all functions of X alone, we can differentiate this equation with respect to y_j to obtain

$$\frac{\partial \Phi}{\partial x_i \partial y_j} = \sum_{q=1}^{a_1} \frac{\partial^2 \Phi}{\partial y_j \partial f^1_q} \frac{\partial f^1_q}{\partial x_i}$$

which establishes the lemma for r=2. (The fact that all functions involved are C^2 allows us to interchange the order of the x_i and y_j differentiations.)

Assume now that the lemma holds for functions of stages less than r. Then if

$$\Phi(X, Y) = \Phi(f^1, \ldots, f^{r-1}, Y)$$

is an r-stage function, we have

$$\frac{\partial \Phi}{\partial x_i} = \sum_{p=1}^{r-1} \sum_{q=1}^{a_p} \frac{\partial \Phi}{\partial f^p_q} \frac{\partial f^p_q}{\partial x_i}$$

and differentiating this with respect to y_j gives

$$\frac{\partial^2 \Phi}{\partial x_i \partial y_j} = \sum_{p=1}^{r-1} \sum_{q=1}^{a_p} \frac{\partial^2 \Phi}{\partial y_j \partial f^p_q} \frac{\partial f^p_q}{\partial x_i} + \sum_{p=1}^{r-1} \sum_{q=1}^{a_p} \frac{\partial \Phi}{\partial f^p_q} \frac{\partial^2 f^p_q}{\partial x_i \partial y_j} \qquad (2)$$

But notice that each function f^p_q is itself a p-stage function computed by X (using information received from Y via the g's). So by induction we may apply the Lemma to f^p_q (in which must interchange the roles of X and Y) to obtain

$$\frac{\partial^2 f^p_q}{\partial x_i \partial y_j} = \sum_{u=1}^{p-1}\sum_{v=1}^{b_u}\beta(p,q)^{uv}_j \frac{\partial g^u_v}{\partial y_j} + \sum_{u=1}^{p-2}\sum_{v=1}^{a_u}\alpha(p,q)^{uv}_i \frac{\partial f^u_v}{\partial x_i}$$

Multiplying this expression by $\frac{\partial \Phi}{\partial f^p_q}$, summing over p and q and consolidating the coefficients of $\frac{\partial f^p_q}{\partial x_i}$ and $\frac{\partial g^s_t}{\partial y_j}$ shows that the second term in Equation (2) can be rewritten as

$$\sum_{p=1}^{r-3}\sum_{q=1}^{a_p}\gamma^{pq}_j \frac{\partial f^p_q}{\partial x_i} + \sum_{s=1}^{r-2}\sum_{t=1}^{b_s}\delta^{st}_i \frac{\partial g^s_t}{\partial y_j}$$

Finally, combining these terms with the first term of Equation (2) proves the Lemma.

5. Proof of Lemma 2.1

Let $D(p,q,r,s)$ equal $\frac{\partial^2 \Phi_k(b)}{\partial m_{pq} \partial m_{rs}}$. We wish to compute the rank of the $n \times (n^2-n)$ matrix Δ which is generated by the $D(p,q,r,s)$ as m_{pq} ranges over the kth column of M, and m_{rs} ranges over all elements of M not in the kth column. Each element m_{rs} $(s \neq k)$ gives rise to a column of Δ which we will denote by $D(*,k,r,s)$.

Claim 1: For any r, $D(*,k,r,s)$ is a scalar multiple of $D(*,k,1,s)$.

This implies, first of all, that the rank of Δ can be at most n-1 (corresponding to the number of distinct choices for s), and secondly, that the rank of Δ is the same as the rank of the $n \times (n-1)$ matrix $\bar{\Delta}$, where $\bar{\Delta}_{ps} = D(p,k,1,s)$. The proof of 2.1 is then completed by

Claim 2: For any non-zero vector b, the corresponding matrix $\bar{\Delta}$ has rank n-1.

To prove these claims, begin by differentiating the equation $Mz = b$ with respect to m_{pq} and note that $\frac{\partial M}{\partial m_{pq}}$ is equal to E_{pq}, the elementary matrix with 1 in the (p,q)-th position and 0 elsewhere. This gives

$$E_{pq} z + M \frac{\partial z}{\partial m_{pq}} = 0$$

or, setting $N = M^{-1}$ to simplify the notation,

$$\frac{\partial z}{\partial m_{pq}} = -NE_{pq}z = -NE_{pq}Nb$$

Differentiating with respect to m_{rs} leads to the equation

$$\frac{\partial^2 z}{\partial m_{pq}\partial m_{rs}} = [NE_{pq}NE_{rs}N + NE_{rs}NE_{pq}N]\,b = [n_{qr}NE_{ps}N + n_{sp}NE_{rq}N]\,b$$

(The second equality comes from the fact that, for any matrix A, $E_{pq}AE_{rs} = a_{qr}E_{ps}$.) Then $D(p,q,r,s)$, which is the kth component of this expression, is equal to the inner product

$$\langle n_{qr}(NE_{ps}N)^{<k>} + n_{sp}(NE_{rq}N)^{<k>},\, b\rangle$$

(We use the symbol $A^{<k>}$ to denote the kth row of the matrix A.) Now using the fact that, for any matrix A, $(AE_{ps}A)^{<k>} = a_{kp}A^{<s>}$, and setting $q = k$ yields

$$D(p,k,r,s) = n_{kr}\langle n_{kp}N^{<s>} + n_{sp}N^{<k>},\, b\rangle \qquad (3)$$

which shows at once that

$$D(p,k,r,s) = \frac{n_{kr}}{n_{k1}}\,D(p,k,1,s)$$

and establishes Claim 1.

Proceeding with Claim 2, we see from (3) that

$$\bar{\Delta}_{ps} = D(p,k,1,s) = n_{k1}\langle n_{kp}N^{<s>} + n_{sp}N^{<k>},\, b\rangle \qquad (4)$$

Since the entries of $\bar{\Delta}$ are symbolic expressions in the m_{ij}, it suffices to show that, for any non-zero b, we can choose particular values for the m_{ij} which lead to corresponding $\bar{\Delta}$'s of rank n-1. Let P_k be the $n \times n$ identity matrix with the 1st and kth rows interchanged. Using (4), the reader can verify that taking $N = P_k$ gives a corresponding $\bar{\Delta}$ of rank n-1, so long as the first component b_1 of b is non-zero. If b_1 does equal zero, then some other component, say b_u is non-zero, in which case taking N equal to $P_k + E_{ku}$ gives a $\bar{\Delta}$ of rank n-1.

6. Remarks

In conclusion, we mention some questions which are left open by this work. First, note that while Theorem 2 deals with computational networks which are much more general than those handled by Theorem 1, it gives only a lower bound on information transfer. It would be useful to derive an upper bound on the information transfer required in two-way networks, analogous to the one for one-way networks.

The network analysis in Section 3 is valid in the most general circumstance, *viz.*, where every node can communicate directly with every other node. By postulating more restrictive configurations, e.g., as in [3], the above results could be strengthened accordingly.

Expressing "information transfer" in terms of number of function values transmitted, rather than in terms of bits, and placing no restrictions other than differentiability on the local functions, broadens applicability of the above theorems beyond purely computational settings to include other kinds of systems in which interactions among "local" processes are a major consideration, for example, biological or social systems. On the other hand, it would be useful to investigate ramifications of the above theorems in constraining the number of bits which must be transmitted in a distributed computation.

One important class of computations in which the differentiability hypothesis is not satisfied is those which make important use of conditional expressions, such as maximizing an expression over the items in a data base. It should be possible to obtain results analogous to the

above theorems for dealing with these situations as well.

Acknowledgements

I would like to thank Michael Rabin for suggesting the problem of extending the results of [1] to networks with two-way communication, and Peter Elias for comments on an earlier draft of this paper.

References

1. Abelson, Harold, "Towards a Theory of Local and Global in Computation," *Theoretical Computer Science*, vol. 6 (1978), 41-67.

2. Abelson, Harold, "Correction to: Towards a Theory of Local and Global in Computation," *Theoretical Computer Science*, in press.

3. Gentleman, W. Morven, "Some Complexity Results for Matrix Computations on Parallel Processors," *JACM*, 25, January 1978, 112-115.