Trade-offs between Communication and Space*

TAK LAM

Department of Computer Science, FR-35, University of Washington, Seattle, Washington 98195

AND

PRASOON TIWARI AND MARTIN TOMPAC

IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598

Received October 26, 1988; revised September 5, 1990

This paper initiates the study of communication complexity when the processors have limited work space. The following trade-offs between the number \( C \) of communications steps and space \( S \) are proved:

1. For multiplying two \( n \times n \) matrices in the arithmetic model with two-way communication, \( CS = \Theta(n^3) \).
2. For convolution of two degree \( n \) polynomials in the arithmetic model with two-way communication, \( CS = \Theta(n^2) \).
3. For multiplying an \( n \times n \) matrix by an \( n \)-vector in the Boolean model with one-way communication, \( CS = \Theta(n^2) \).

In contrast, the discrete Fourier transform and sorting can be accomplished in \( O(n) \) communication steps and \( O(\log n) \) space simultaneously, and the search problems of Karchmer and Wigderson associated with any language in \( NC^k \) can be solved in \( O(\log^k n) \) communication steps and \( O(\log^k n) \) space simultaneously.


1. COMMUNICATION AND SPACE

The minimum communication required in order to solve problems in the two-processor model has been studied extensively. (See, for example, [28, 1, 3, 19].) This paper initiates the study of communication complexity when the processors have limited work space. As is customary, the systems we study consist of two com-

1 Supported by an IBM Graduate Fellowship. Author's current address: Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong. The work was performed while the first author was a visitor at the Thomas J. Watson Research Center, IBM Research Division.
2 Current address: Department of Computer Science, 1210 West Dayton Street, University of Wisconsin–Madison, Madison, WI 53706.
3 Current address: Department of Computer Science, FR-35, University of Washington, Seattle, WA 98195.
municating processors that are given private inputs \(x\) and \(y\), respectively, and that are to output some function \(f(x, y)\). We restrict our attention to the case in which these processors execute straight-line protocols (defined in Section 2) and measure both the number \(S\) of work registers and the number \(C\) of communication steps used.

With no restriction on \(S\) it is impossible to prove superlinear lower bounds on \(C\), since one processor can send its entire input to the other, which then computes and outputs \(f(x, y)\). In contrast, we prove the following trade-offs when space is restricted to \(S\):

1. For multiplying two \(n \times n\) matrices in the arithmetic model with two-way communication, \(CS = \Theta(n^3)\).

2. For convolution of two degree \(n\) polynomials in the arithmetic model with two-way communication, \(CS = \Theta(n^3)\).

3. For multiplying an \(n \times n\) matrix by an \(n\)-vector in the Boolean model with one-way communication, \(CS = \Theta(n^3)\).

The proof technique used in the arithmetic lower bounds (Section 3) is quite different from that used in the Boolean lower bound (Section 4). The lower bounds are based on a new pebble game that models the space and communication requirements of straight-line protocols.

If a single processor can compute \(f(x, y)\) in time \(C\) and space \(S\), then a system of two processors can compute \(f(x, y)\) in communication \(O(C)\) and space \(O(S)\), simply by communicating every intermediate value computed by either. Viewed another way, almost all the known time-space trade-offs are special cases of communication-space trade-offs, in which one processor receives all the inputs but is incapable of computation, being allowed only to communicate the inputs to the other processor. Thus, the new lower bounds outlined above imply previous time-space trade-offs of Ja'Ja' [15], Tompa [26], and Grigoryev [13]. (They do not imply the results of Yesha [30] and Abrahamson [2].)

The converse, however, is false. Whereas the time \(T\) and space \(S\) must satisfy \(TS = \Omega(n^3)\) when computing the discrete Fourier transform [26] or sorting [7, 26], in Section 6 we demonstrate that both of these functions can be computed in linear communication steps and \(O(\log n)\) space simultaneously. As further motivation for studying space-bounded communication complexity, in Section 5 we show that the search problems of Karchmer and Wigderson [16] associated with any language in \(NC^k\) can be solved in \(O(\log^k n)\) communication steps and \(O(\log^k n)\) space simultaneously.

Hong and Kung [14] introduced the “red-blue pebble game” to study the space and I/O requirements of straight-line programs. This too can be viewed as a special case of our new pebble game, in which one processor (the memory) has no space bound but is incapable of computing.

The closest previous work is that of Papadimitriou and Ullman [20], in which they proved a communication-time trade-off. Both their work and the aforementioned work of Hong and Kung [14] studied lower bounds for straight-line
implementations of a single circuit, whereas our lower bounds apply to any circuit that solves the problem. For instance, Hong and Kung proved that the space $S$ and I/O $Q$ required to implement the standard straight-line matrix multiplication algorithm satisfy $Q = O(n^3)$, whereas we prove that $CS = \Theta(n')$ for any straight-line matrix multiplication algorithm.

2. A Pebble Game That Models Communication and Space

In this section, we examine what it means for communicating processors to execute "straight-line protocols." Each processor is assumed to have a set of private registers $R_1, R_2, \ldots$ and private inputs. A straight-line protocol consists of two sequences of instructions, one executing on each processor. There are four kinds of instructions:

1. $R_i \leftarrow a$, where $a$ is an input or constant.
2. $R_i \leftarrow f(R_j, R_{j2}, \ldots, R_{jz})$, where $f$ is a primitive operator.
3. send $R_i$.
4. $R_i \leftarrow$ receive.

If $x$ and $y$ denote the respective inputs to the two processors, the processors themselves will be referred to as the $x$-processor and the $y$-processor. We first consider the case in which there is a one-way communication channel from the $x$-processor to the $y$-processor. In this case the $y$-processor cannot execute the send instruction and the $x$-processor cannot execute the receive instruction. A send causes a copy of the value stored in the specified register to be loaded into the communication channel, and the sending processor pauses until the value is received by the other processor. In executing a receive, a processor waits until there is some value on the channel, then copies that value into the designated register. A two-way communication channel can be viewed as two one-way communication channels, one in each direction.

A protocol is said to compute a set of values if each value in the set is stored in some private register at some time during the execution.

The space used by a straight-line protocol is the maximum number of registers used by either processor. The communication is the total number of send instructions executed by both processors.

There is a natural way in which a straight-line protocol gives rise to an equivalent circuit. However, a circuit can be realized by several straight-line protocols. To determine the space and communications required to implement a circuit, we introduce a new pebble game. The idea is analogous to the use of other pebble games in studying time and space requirements of straight-line programs [21]. This new game is implicit in the work of Papadimitriou and Ullman [20].

There are two sets of pebbles, referred to as $x$-pebbles and $y$-pebbles. In each move, one can remove zero or more pebbles from the circuit and then, according to the following rules, choose a vertex $v$ of the circuit and pebble it.
1. If $v$ is an input vertex from $x$ ($y$), then an $x$-pebble (resp. $y$-pebble) can be placed on $v$.

2. If, at the beginning of the move, all immediate predecessors of $v$ were $x$-pebbled ($y$-pebbled), then $v$ can be $x$-pebbled (resp. $y$-pebbled).

3. If, at the beginning of the move, there was an $x$-pebble on $v$, then $v$ can be $y$-pebbled. In this case we say a communication (from the $x$-processor to the $y$-processor) has occurred at $v$.

The game as described models one-way communication from the $x$-processor to the $y$-processor. If two-way communication is allowed, then rule 3 is duplicated with $x$ and $y$ interchanged.

The goal of the game is to have pebbled each output vertex of the circuit at least once. Each pebbling strategy corresponds uniquely to a straight-line protocol as follows. Each pebble corresponds to a register of one of the processors. Pebbling a vertex $v$ corresponds to loading the value computed at $v$ into the corresponding register. Therefore, the maximum number of pebbles of either type used by a pebbling strategy measures the space used by the equivalent straight-line protocol, and the number of applications of rule 3 above measures the number of communication steps used.

3. Trade-offs for Arithmetic Straight-line Protocols

This section is divided into two subsections. In Subsection 3.1, we observe that, without loss of generality, we can restrict our attention to bilinear arithmetic circuits while proving the communication-space trade-offs for general arithmetic straight-line programs that compute bilinear forms. This is reminiscent of a similar scenario when we count only the number of nonscalar multiplications performed by arithmetic straight-line programs for computing bilinear forms [S]. Then, in Subsection 3.2, we prove the communication-space trade-offs claimed in Section 1 for arithmetic straight-line protocols.

3.1. Preliminary Remarks

Lemmas 1 and 2 stated below follow immediately from a result of Winograd, and its generalization due to Strassen and Unger, respectively [S]. These lemmas will enable us to restrict our attention to bilinear straight-line programs in Subsection 3.2. For a definition of bilinear (linear) programs or circuits, see either of [8, 26, 27].

**Lemma 1.** Let $G$ be an arithmetic circuit with $\{+,-,\ast\}$ gates, computing bilinear forms, that can be pebbled using $S$ pebbles in $C$ communication steps. Then there is a bilinear arithmetic circuit $X$ with $\{+,-,\ast\}$ gates that can be pebbled using $O(S)$ pebbles in $O(C)$ communication steps.
Proof. This lemma can be proved using arguments similar to those used in the proof of Theorem 2.5.3 of [8].

Lemma 2. Let \( G \) be an arithmetic circuit with \{ +, −, *, / \} gates, computing bilinear forms, that can be pebbled using \( S \) pebbles in \( C \) communication steps. Then there is an arithmetic circuit \( H \) with gates only from the set \{ +, −, * \} that can be pebbled using \( O(S) \) pebbles in \( O(C) \) communication steps.

Proof: This lemma can be proved using arguments similar to those used in the proof of Theorem 2.5.5 of [8].

In our proofs, we will need the following two results from [11, 27] (see also [26, Lemma 2]), respectively.

Let \([n] = \{1, 2, \ldots, n\}\) and let \( A_{m \times n} \) be an \( m \times n \) matrix. Let \( A(X, Y) \) be the submatrix of \( A \) consisting of all the rows and all the columns whose indices are contained in the respective subsets \( X \subseteq [m] \) and \( Y \subseteq [n] \).

Theorem 3 (Binet–Cauchy). Let \( A_{m \times m} = B_{m \times k} C_{k \times m} \), where \( k > m \). Then

\[
\det A = \sum_{Z \subseteq [k], |Z| = m} s(Z) \det B([m], Z) \det C(Z, [m]),
\]

where \( s(Z) \in \{1, -1\} \).

Define \( \text{diag}(\alpha_1, \alpha_2, \ldots, \alpha_k) \) to be the \( k \times k \) diagonal matrix that has \( \alpha_1, \alpha_2, \ldots, \alpha_k \) on the main diagonal, in that order. We use the Binet–Cauchy theorem in order to prove the following corollary, which is required in Section 3.2.

Corollary 4. Let \( A_{m \times n} = B_{m \times k} \Delta_{k \times k} C_{k \times n} \), where

\( \Delta = \text{diag}(\alpha_1, \alpha_2, \ldots, \alpha_k) \).

If \( \text{rank}(A) \geq r \), then there exist \( X \subseteq [m], Y \subseteq [n], \) and \( Z \subseteq [k] \), such that \( |X| = |Y| = |Z| = r \), and \( B(X, Z) \) and \( C(Z, Y) \) are both nonsingular. Furthermore, the rows and the columns of these matrices can be rearranged so that \( X = Y = [r] \).

Proof. Since \( \text{rank}(A) \geq r \), we can select subsets \( X \subseteq [m] \) and \( Y \subseteq [n] \) such that \( |X| = |Y| = r \), and \( A(X, Y) \) is nonsingular. Observe that \( A(X, Y) = B(X, [k]) \Delta C([k], Y) \). By two applications of the Binet–Cauchy theorem on determinants,

\[
\det A(X, Y) = \sum_{|Z| = r, Z \subseteq [k]} \pm \det B(X, Z) \det \Delta(Z, Z) \det C(Z, Y).
\]

Observe that since \( \Delta \) is diagonal, the determinant would be zero if the set of row indices and the set of column indices are unequal. Since \( \det A(X, Y) \neq 0 \), there is some subset \( Z \) such that all of \( B(X, Z), \Delta(Z, Z), \text{ and } C(Z, Y) \) are nonsingular. 

1 See [11] for a definition of \( s(Z) \).
LEMMA 5 (Valiant). Let \( A \) be an \( m \times n \) matrix with a nonsingular \( r \times r \) minor \( A' \) on the rows \( i_1, \ldots, i_r \) and columns \( j_1, \ldots, j_r \). Let \( G \) be any linear circuit computing \( y = Ax \) on input \( x \), where \( x = [x_1, \ldots, x_n]^T \) and \( y = [y_1, \ldots, y_m]^T \). Then there are \( r \) vertex-disjoint paths from inputs \( \{x_{i_1}, \ldots, x_{i_r}\} \) to outputs \( \{y_{i_1}, \ldots, y_{i_r}\} \).

3.2. The Arithmetic Trade-offs

Suppose that \( \{x_1, x_2, \ldots, x_n\} \) and \( \{y_1, y_2, \ldots, y_n\} \) are two sets of indeterminates. Consider a set \( \{f_1, f_2, \ldots, f_m\} \) of \( m \) bilinear forms in these variables with coefficients from a commutative ring \( R \). Let \( f_i = \sum_{j=1}^{n} \sum_{k=1}^{n} m_{jk}^{(i)} x_j y_k \), and define \( n \times n \) matrices \( M_i \) by \( (M_i)_{jk} = m_{jk}^{(i)} \), for \( i \in \{1, 2, \ldots, m\} \). Note that \( f_i = x^T M_i y \), where \( x = [x_1, x_2, \ldots, x_n]^T \) and \( y = [y_1, y_2, \ldots, y_n]^T \).

Now we are ready to state the main theorem of this section. This theorem states that if a set of bilinear forms is "sufficiently independent," then arithmetic straight-line programs for computing it exhibit a communication-space trade-off. As corollaries to this theorem, we will prove the main results of this section: (a) multiplication of two \( n \times n \)-matrices requires \( CS = \Theta(n^3) \); (b) convolution of two degree \( n \) polynomials requires \( CS = \Theta(n^2) \).

THEOREM 6. Let \( G \) be an arithmetic circuit that computes \( f_i = x^T M_i y \), \( i \in \{1, 2, \ldots, m\} \), where

\[ \min_{x \neq \{0\}} \text{rank} \left( \sum_{i=1}^{m} x_i M_i \right) \geq r. \]

Then any pebbling strategy for \( G \) that uses \( C \) (two-way) communication steps and \( 2S \) pebbles satisfies \( C \geq \lceil m/(2S + 1) \rceil (r - 2S) \).

Lower bounds on the rank in a linear subspace of \( R^{n \times n} \), like the one used in Theorem 6, have been used earlier to relate the complexity of certain bilinear forms to the parameter of a corresponding error-correcting code [9, 17].

In order to prove this theorem, we will need the following result, whose proof is given later in this section.

LEMMA 7. Let \( f_1, f_2, \ldots, f_{2S+1} \) be \( 2S + 1 \) bilinear forms such that

\[ \min_{x \neq \{0\}^{2S+1}} \text{rank} \left( \sum_{i=1}^{2S+1} x_i M_i \right) \geq r. \]

Let \( \mathcal{H} \) be a bilinear circuit computing \( f_1, f_2, \ldots, f_{2S+1} \). If \( 2S \) pebbles are placed on arbitrarily chosen vertices of \( \mathcal{H} \), then there exist three sets of distinct vertices \( U = \{u_1, u_2, \ldots, u_r\} \), \( V = \{v_1, v_2, \ldots, v_r\} \), and \( W = \{w_1, w_2, \ldots, w_r\} \) such that (Fig. 1):

1. \( U \) is a subset of multiplication gates. Every gate in \( U \) has a pebble-free path to an output;

2. \( V \) is a subset of the inputs to the \( x \)-processor. \( \mathcal{H} \) contains \( r \) vertex-disjoint paths from \( v_i \) to \( u_i \) (\( i \in \{1, 2, \ldots, r\} \)); and

3. \( W \) is a subset of the inputs to the \( y \)-processor. \( \mathcal{H} \) contains \( r \) vertex-disjoint paths from \( w_i \) to \( u_i \) (\( i \in \{1, 2, \ldots, r\} \)).
Proof of Theorem 6 (using Lemma 7). By Lemmas 1 and 2 we can restrict our attention to bilinear circuits. Consider the pebble game of Section 2 on a bilinear circuit $\mathcal{G}$. We can partition the whole game into $\lceil m/(2S+1) \rceil$ phases. The $i$th phase begins immediately after a total of $(i-1)(2S+1)$ outputs have been pebbled, and it ends immediately after a total of $i(2S+1)$ outputs are pebbled. The last phase may be an exception; it ends when all outputs have been pebbled. Observe that during each phase (except possibly the last one) exactly $2S+1$ new outputs are pebbled. Using Lemma 7, we will prove that each phase requires at least $r - 2S$ communication steps. This, in turn, implies the theorem.

Consider the $i$th phase. Let $T$ be the set of $2S+1$ indices of new outputs pebbled in this phase. Let $\mathcal{H}$ be the subcircuit of $\mathcal{G}$ consisting of all the edges and the vertices that lie on directed paths to at least one of the vertices of $T$. Lemma 7 is applicable to $\mathcal{H}$, since

$$
\min_{x \neq \{0\}^{2S+1}} \text{rank} \left( \sum_{i \in T} x_i M_i \right) \geq \min_{x \neq \{0\}^m} \text{rank} \left( \sum_{i=1}^m x_i M_i \right) \geq r.
$$

Lemma 7 guarantees the existence of suitable sets $U$, $V$, and $W$ together with two sets of vertex-disjoint paths (which may have vertices in common); assume that these paths connect $v_i \in V$ and $w_i \in W$ to $u_i \in U$. 

**FIG. 1.** The configuration guaranteed by Lemma 7.
Let $U' = \{ u_i \mid u_i \in U \}$, and the path from $v_i$ to $u_i$ (and/or $w_i$ to $u_i$) is free of $y$-pebbles (resp. $x$-pebbles) at the beginning of this phase. Then $|U'| \geq r - 2S$, since each of the $2S$ pebbles eliminates at most one gate from $U'$. For each $u_i \in U'$, there must be at least one communication from the $x$-processor to the $y$-processor on the path from $v_i$ to $u_i$, or one from the $y$-processor to the $x$-processor on the path from $w_i$ to $u_i$. Hence, at least $|U'| \geq r - 2S$ communication steps must occur in this phase.

Before we prove Lemma 7, we need some additional notation. Suppose that there are $p$ multiplication gates in $H$. Number them from 1 to $p$, and let $q_i$ be the bilinear form that is the output of the $i$th multiplication gate. Then $q_i = g_ih_i$, where $g_i = \sum_{j=1}^{r} \mu_j x_j$ and $h_i = \sum_{k=1}^{r} \nu_{ki} y_k$. Define column vectors $A_i = [\mu_{1i} \mu_{2i} \cdots \mu_{pi}]^T$, $B_i = [v_{1i} v_{2i} \cdots v_{ni}]^T$. Then the coefficient matrix $Q_i$ for the bilinear form $q_i$ is given by $Q_i = A_iB_i^T$.

In order to prove Lemma 7, we need an auxiliary result.

**Lemma 8.** Let $f_1, f_2, \ldots, f_{2S+1}$ be $2S + 1$ bilinear forms such that

$$\min_{\alpha \neq \{0\}^{2S+1}} \text{rank} \left( \sum_{i=1}^{2S+1} \alpha_i M_i \right) \geq r.$$ 

Let $H$ be a bilinear circuit computing $f_1, f_2, \ldots, f_{2S+1}$. If $2S$ pebbles are placed on arbitrarily chosen vertices of $H$, then there is a set of $d$ multiplication gates, numbered $i_1, i_2, \ldots, i_d$, such that (a) each of them has a pebble-free path to at least one output and (b) the corresponding bilinear forms $q_{i_1}, q_{i_2}, \ldots, q_{i_d}$ satisfy the property

$$\max \{ \text{rank}(A \cdot B^T) \mid A \text{ is a } d \times d \text{ diagonal matrix} \} \geq r,$$

where $A = [A_{i_1} A_{i_2} \cdots A_{i_d}]$, $B = [B_{i_1} B_{i_2} \cdots B_{i_d}]$.

**Proof of Lemma 7 (using Lemma 8).** Using Lemma 8, choose $i_1, i_2, \ldots, i_d$, and $A$ such that $\text{rank}(A \cdot B^T) = k \geq r$. We will prove that, after a suitable rearrangement of indices, $V$ and $\bar{W}$ can be chosen as follows: $V = \{ v_i \mid v_i \text{ is the input vertex in } H \text{ corresponding to the input } x_i, 1 \leq i \leq k \}$; $W = \{ w_i \mid w_i \text{ is the input vertex in } H \text{ corresponding to the input } y_i, 1 \leq i \leq k \}$.

Corollary 4 implies that, after a suitable rearrangement of indices, there exists $Z \subseteq [d]$, with $|Z| = k$, such that $A([k], Z)$ and $B^T(Z, [k])$ are nonsingular. Fix some such $Z$ for the rest of this proof. Let $U = \{ u_i \mid u_i \text{ is the gate in } H \text{ corresponding to } q_j, j \in Z \}$. Notice that, if we set $x_{k+1} = \cdots = x_n = 0$ ($y_{k+1} = \cdots = y_n = 0$), then all the linear forms in $x(y)$ that are inputs to the multiplication gates in $U$ are components of the vector $[x_1 \cdots x_k] A([k], Z)$ (respectively, $B^T(Z, [k])[y_1 \cdots y_k] ^T$). By Valiant's lemma, there are $k$ vertex-disjoint paths between $U$ and each of $V$ and $\bar{W}$ because $A([k], Z)$ and $B^T(Z, [k])$ are nonsingular, respectively.
Proof of Lemma 8. Assume that \( X = \{ q_1, q_2, \ldots, q_l \} \) is the set of all multiplication gates that have a pebble-free path to at least one of the outputs \( f_1, f_2, \ldots, f_{2s+1} \), and

\[
\max \{ \text{rank}(A \Delta B^T) \mid A \text{ is an } l \times l \text{ diagonal matrix} \} < r,
\]

where \( A = [A_1, A_2, \ldots, A_l] \), \( B = [B_1, B_2, \ldots, B_l] \). Then, we will prove that there is another multiplication gate \( \tilde{q} \notin X \) that has a pebble-free path to at least one of the outputs \( f_1, f_2, \ldots, f_{2s+1} \). This contradicts the maximality of \( X \), and hence will be sufficient to prove the lemma. We know that

\[
f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{2s+1} \end{bmatrix} = \begin{bmatrix} \cdots \\ \Sigma \\ \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_l \\ q_{l+1} \\ \vdots \\ q_p \end{bmatrix}, \tag{1}
\]

where \( \Sigma \) is a \((2S+1) \times l\) matrix, and \( \Pi \) is a \((2S+1) \times (p-l)\) matrix. Then

\[
t = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_{2s+1} \end{bmatrix} = f - \Sigma \begin{bmatrix} q_1 \\ \vdots \\ q_l \\ q_{l+1} \end{bmatrix} = \Pi \begin{bmatrix} q_1 \\ \vdots \\ q_p \end{bmatrix}.
\]

Next we show that \( t_1, t_2, \ldots, t_{2s+1} \) are linearly independent. Suppose that this is not the case. Then there are constants \( \alpha_1, \alpha_2, \ldots, \alpha_{2s+1} \), not all zero, such that \( \sum_{i=1}^{2s+1} \alpha_i t_i = 0 \). Substituting for \( t_i \), we get \( \sum_{i=1}^{2s+1} \alpha_i f_i - \sum_{i=1}^{l} \beta_i q_i = 0 \), or \( \sum_{i=1}^{2s+1} \alpha_i f_i = \sum_{i=1}^{l} \beta_i q_i \). This identity means that the coefficient of all the \( n^2 \) terms \( x_j y_k \) on both sides are equal, i.e., \( \sum_{i=1}^{2s+1} \alpha_i m_{jk}^{(i)} = \sum_{i=1}^{l} \beta_i \mu_{ji} \nu_{ki} \). These \( n^2 \) equations can be written in the matrix form as follows:

\[
\sum_{i=1}^{2s+1} \alpha_i M_i = \sum_{i=1}^{l} \beta_i Q_i = A \text{ diag}(\beta_1, \beta_2, \ldots, \beta_l) B^T.
\]

But this is not possible, since rank(\( \sum_{i=1}^{2s+1} \alpha_i M_i \)) \( \geq r \) and rank\( (A \text{ diag}(\beta_1, \beta_2, \ldots, \beta_l) B^T) \) \( < r \). Therefore, \( t_1, t_2, \ldots, t_{2s+1} \) are linearly independent bilinear forms.

Linear independence of \( t_1, \ldots, t_{2s+1} \) implies that \( \Pi \) has some nonsingular \((2S+1) \times (2S+1)\) minor. Referring back to Eq. (1), Valiant's lemma implies that there must be \( 2S + 1 \) vertex-disjoint paths to \( f_1, \ldots, f_{2s+1} \) from some \( 2S + 1 \) multiplications gates numbered at least \( l + 1 \). At least one of these paths, say from \( f_i \) to \( q_j \), is pebble-free. Then \( q_j \) can be chosen as \( \tilde{q} \), as asserted at the beginning of the proof.

Using similar techniques, it can be shown that, for any set of bilinear forms satisfying the conditions of Theorem 6, \( C \geq r \), independent of \( S \).
Theorem 9. Let $G$ be an arithmetic circuit that computes $f_i = x^T M_i y$, $i \in \{1, 2, ..., m\}$. Suppose the values of $x_1, x_2, ..., x_n$ can be fixed so that $f_1, f_2, ..., f_m$ are linearly independent in $y_1, y_2, ..., y_n$. Similarly, suppose the values of $y_1, y_2, ..., y_n$ can be fixed so that $f_1, f_2, ..., f_m$ are linearly independent in $x_1, x_2, ..., x_n$. Then any pebbling strategy for $G$ that uses $C$ (two-way) communication steps satisfies $C > m$, independent of the number of pebbles used.

Proof. By Valiant's lemma, there are $m$ vertex-disjoint paths from some subset of $\{y_1, y_2, ..., y_n\}$ (resp. $(x_1, x_2, ..., x_n)$) to the outputs. Note that these two sets of vertex disjoint paths may intersect. Despite this, there must be at least $m$ communication steps, as in the proof of Theorem 6.

Now we are ready to prove a communication-space trade-off for convolution of two vectors $x = [x_0, x_1, ..., x_{n-1}]^T$ and $y = [y_0, y_1, ..., y_{n-1}]^T$. Recall that the convolution of two vectors $x$ and $y$ is a vector $z = [z_0, z_1, ..., z_{n-1}]$, where $z_k = \sum_{i=0}^{n-k} x_i y_{(k-i)} \mod n$, for $0 \leq k < n$. Let $M_k$ be the coefficient matrix of the bilinear form $z_k$, i.e., $(M_k)_{ij}$ is the coefficient of $x_i y_j$ in $z_k$.

Corollary 10. The arithmetic straight-line complexity for convolution of two degree $n$ polynomials satisfies $CS = \Theta(n^2)$.

Proof. Consider only the first $\lceil n/2 \rceil$ outputs. If not all the $x_i$'s are zero, then a nonsingular square submatrix of size at least $\lceil n/2 \rceil$ can be isolated in the bottom right corner of $\sum_{i=0}^{\lceil n/2 \rceil - 1} x_i M_i$. Therefore,

$$\text{rank} \left( \sum_{i=0}^{\lceil n/2 \rceil - 1} x_i M_i \right) \geq \left\lceil \frac{n}{2} \right\rceil,$$

whenever not all of the $x_i$'s are equal to zero. The lower bound proof can be separated into the following two cases:

Case 1. $S > n/8$. Then Theorem 9 implies that $C \geq \lceil n/2 \rceil$.

Case 2. $S \leq n/8$. In this case, Theorem 6 implies that

$$C \geq \left\lceil \frac{n}{2(2S+1)} \right\rceil \left( \frac{n}{2} - 2S \right).$$

The desired lower bound is an immediate consequence.

For a matching upper bound, we divide the computation of $\{z_0, ..., z_{n-1}\}$ into $\lceil n/S \rceil$ phases. In the $i$th phase the coefficients $z_{(i-1)S}, ..., z_{iS-1}$ are computed by the $y$-processor as follows. At the beginning of each phase, the $y$-processor resets all its registers to zero. Then the $x$-processor starts transmitting the sequence $x_0, x_1, ..., x_n$. After receiving $x_i$, the $y$-processor updates its registers so that they contain $\sum_{l=0}^{i-1} x_l y_{(k-l)} \mod n$, for $(i-1)S < k \leq iS$. Since each phase has exactly $n$ communication steps, the total communication required is $O(n^2/S)$.
A similar argument can be used to prove a communication-space trade-off for matrix multiplication. Let \( A \) and \( B \) be \( n \times n \) matrices whose elements are drawn from a ring. Let \( C = AB \), \((A)_{ij} = x_{ij}\), \((B)_{ij} = y_{ij}\), and \((C)_{ij} = z_{ij}\). Then 
\[
z_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj},
\]
for any \( 1 \leq i, j \leq n \). Initially the \( A \)-processor and the \( B \)-processor have inputs \( \{x_{11}, \ldots, x_{nn}\} \) and \( \{y_{11}, \ldots, y_{nn}\} \), respectively. They cooperate to compute all entries of \( C \) in an arbitrary order. Again, each \( z_{ij} \) can be viewed as a bilinear form in the \( n^2 \) entries of \( A \) and \( B \). Let \( M_{ij} \) be the \( n^2 \times n^2 \) coefficient matrix of the bilinear form \( z_{ij} \). \( M_{ij} \) can be arranged into an \( n \times n \) matrix of blocks, where each block submatrix has size \( n \times n \), such that its \((i,j)\)th block is an identity matrix, and the others are all equal to zero.

**Corollary 11.** The arithmetic straight-line complexity for multiplying two \( n \times n \) matrices satisfies \( CS = O(n^3) \).

**Proof.** The block structure of \( M_{ij} \)'s implies that \( \text{rank}(\sum_{ij}(x_{ij}M_{ij})) \geq n \), whenever not all of the \( x_{ij} \)'s are equal to zero. The proof can be completed along the lines of the proof of Corollary 10.

A matching upper bound is provided by an algorithm similar to the one described in Corollary 10 for convolution. We split the computation in phases, computing \( S \) outputs in each phase using one-way communication.

4. **Trade-offs for Boolean Straight-line Protocols**

In this section we turn from arithmetic circuits to Boolean circuits. The lower bounds are complicated by the fact that we know less in this case about the internal structure of the circuits. In particular, there are no distinguished internal gates corresponding to the multiplication gates in bilinear circuits that indicate where the communications occur. As a result, we must be content with bounds on the model with one-way communication.

In Theorems 14 and 15 we prove a tight communication–space trade-off for any Boolean circuit that multiplies a matrix by a vector, one theorem for each direction of communication. The technique used is derived from that of Grigoryev [13] for Boolean time–space trade-offs. We also need the following combinatorial lemma of Sauer [22] and Perles and Shelah [23].

**Lemma 12 (Sauer; Perles and Shelah).** For \( k \geq 1 \), let \( B_k \) be a \( 2^k \times k \) matrix whose rows are the \( 2^k \) distinct elements of \( \{0, 1\}^k \). Let \( M \) be an \( r \times n \) Boolean matrix with distinct rows, and not containing as a submatrix any matrix whose rows are a permutation of the rows of \( B_k \). Then
\[
r \leq \sum_{j=0}^{k-1} \binom{n}{j}
\]
(with the usual convention that \( \binom{n}{h} = 0 \) unless \( 0 \leq h \leq n \)).
Proof. Case 1 \((k = 1)\). Since the rows of \(M\) are distinct, \(r \leq 1\) as required.

Case 2 \((k > 1)\). The proof of this case is by induction on \(n\).

Basis \((n = 1)\). Then \(r \leq 2\), as required when \(k > 1\).

Induction \((n > 1)\). Let \(D\) be the matrix formed by removing the first column of \(M\). Some rows of \(D\) may occur twice. Let \(D_1\) be the \(r_1 \times (n - 1)\) matrix consisting of the distinct rows of \(D\). Let \(D_2\) be the \(r_2 \times (n - 1)\) matrix consisting of the rows that appear twice in \(D_1\), each such row appearing only once in \(D_2\). As \(M\) does not contain any row permutation of \(B_k\) as a submatrix, \(D_1\) and \(D_2\) cannot contain any row permutations of \(B_k\) and \(B_{k-1}\), respectively, as submatrices. By the induction hypothesis,

\[
\begin{align*}
r_1 &\leq \sum_{j=0}^{k-1} \binom{n-1}{j} \\
r_2 &\leq \sum_{j=0}^{k-2} \binom{n-1}{j} = \sum_{j=0}^{k-1} \binom{n-1}{j-1}.
\end{align*}
\]

Therefore,

\[
r = r_1 + r_2 \leq \sum_{j=0}^{k-1} \left( \binom{n-1}{j} + \binom{n-1}{j-1} \right) = \sum_{j=0}^{k-1} \binom{n}{j}.
\]

Lemma 13. Let \(G\) be any Boolean circuit that inputs an \(n \times n\) matrix \(A\) and a vector \(x \in \{0, 1\}^n\), and outputs \(Ax \in \{0, 1\}^n\). Consider any pebbling strategy on \(G\) that has one-way communication from the \(x\)-processor to the \(A\)-processor and that uses \(S\) \(A\)-pebbles and any number of \(x\)-pebbles. Then in the course of pebbling any set \(Y\) of \(S + 1\) outputs, starting from any configuration of pebbles on \(G\), communication steps occur at more than \(n - (S + 1) \log_2 n\) gates.

Proof. Note that, because of the one-way communication, all the outputs in \(Y\) must be \(A\)-pebbled. Suppose, by way of contradiction, that communication steps occur only at gates in the set \(K\), where \(|K| \leq n - (S + 1) \log_2 n\). There must be a set \(F \subseteq \{0, 1\}^n\) of at least \(2^n/2^{|K|} \geq n^{S+1}\) distinct assignments of values to the inputs in \(x\) that fix the values of the gates in \(K\). Construct an \(|F| \times n\) Boolean matrix \(M\) whose rows are the elements of \(F\). Lemma 12 demonstrates that \(M\) contains some row permutation of \(B_{S+1}\) as a submatrix, since

\[
|F| \geq n^{S+1} > \sum_{j=0}^{S} n^j \geq \sum_{j=0}^{S} \binom{n}{j}.
\]

That is, there is a set \(X\) of \(S + 1\) input vertices in \(x\) and a set \(F' \subseteq F\) that assigns each of the possible \(2^{S+1}\) values to \(X\). Note that the \(2^{S+1}\) assignments in \(F'\) fix the values computed at \(K\).

Now fix \(A\) to be a permutation matrix that maps the \(S + 1\) inputs in \(X\) to the \(S + 1\) outputs in \(Y\). As the assignments to \(x\) vary over \(F'\), the outputs in \(Y\) assume all \(2^{S+1}\) possible values. But since \(A\) is fixed and the values communicated at \(K\) are fixed, the values computed at \(Y\) are completely determined by the values computed
at the $S$ gates that were initially A-pebbled. The contradiction follows from the fact that these $S$ gates can assume only $2^S$ values as assignments to $x$ vary over $F$.

**Theorem 14.** Let $\mathcal{G}$ be any Boolean circuit that inputs an $n \times n$ matrix $A$ and a vector $x \in \{0, 1\}^n$, and outputs $Ax \in \{0, 1\}^n$. Consider any pebbling strategy on $\mathcal{G}$ that has one-way communication from the $x$-processor to the $A$-processor, and uses $S$ A-pebbles, any number of $x$-pebbles, and $C$ communication steps. Then $CS = \Omega(n^2)$, provided $S = o(n/\log n)$.

**Proof.** By Lemma 13, $n - o(n)$ communication steps occur every $S + 1$ outputs.

**Theorem 15.** Let $\mathcal{G}$ be any Boolean circuit that inputs an $n \times n$ matrix $A$ and a vector $x \in \{0, 1\}^n$, and outputs $Ax \in \{0, 1\}^n$. Consider any pebbling strategy on $\mathcal{G}$ that has one-way communication from the $A$-processor to the $x$-processor, and uses $S$ $x$-pebbles, any number of $A$-pebbles, and $C$ communication steps. Then $CS = \Omega(n^2)$.

**Proof.** By an argument similar to that of Lemma 13, it will be shown that $n$ communication steps occur for every set $Y$ of $S + 1$ outputs pebbled, from which the stated result follows. Suppose, by way of contradiction, that communication steps occur only at gates in the set $K$, where $|K| \leq n - 1$. There must be a set $F \subseteq \{0, 1\}^n$ of at least $2^{n(S + 1)/2}|K| \geq 2^{nS} + 1$ distinct assignments of values to the inputs in the relevant $S + 1$ rows of $A$ that fix the values of the gates in $K$. Some column $i$ takes on more than $2^S$ values as the assignments vary over $F$. Now fix $x$ to have a one in the $i$th component and zeros elsewhere. As the assignments to $A$ vary over $F$, the outputs in $Y$ assume more than $2^S$ values. But since $x$ is fixed and the values communicated at $K$ are fixed, the values computed at $Y$ are completely determined by the values computed at the $S$ gates that were initially $x$-pebbled. The contradiction follows from the fact that these $S$ gates can assume only $2^S$ values as assignments to $A$ vary over $F$.

It is straightforward to construct a protocol achieving $CS = O(n^2)$, similar to that for convolution given in Section 3. Notice that this only requires one-way communication from the $x$-processor to the $A$-processor.

### 5. Search Problems

In this section we demonstrate that there are some problems for which the product of communication and space is $o(n)$. (For the purpose of this section, both resources will be measured in units of bits, and both processors will be assumed to be deterministic Turing machines.) A simple example of such a problem is any symmetric Boolean function in $\text{DSPACE}(S(n))$, for any function $S(n) \geq \log n$. Such a Boolean function can be computed using $O(\log n)$ communications and $O(S(n))$ space, simply by having one processor send the number of 1's in its input.
More interesting examples derive from the “search problems” of Karchmer and Wigderson [16]. Associated with any language $L$ is the following search problem $S_L$: the 1-processor receives any $x \in L$, and the 0-processor any $y \notin L$. At the end of their computation, they must output some index $i$ such that their inputs differ in the $i$th position.

**Theorem 16.** For any $L \in \text{ATIME}(T)$, the search problem $S_L$ can be solved (uniformly) in $O(T)$ communications and $O(T)$ space.

**Proof.** The proof is identical to a corresponding one of Karchmer and Wigderson [16, Lemma 2.1], except that they had no need to account for the space used by the communicating processors.

Let $M$ be an alternating Turing machine that accepts $L$ in time $T$. Assume without loss of generality that each configuration of $M$ has at most two successors and that $M$ reads only one input character along any computation path, halting immediately after doing so. The latter is accomplished as follows. Whenever $M$ intended to read an input in the middle of its computation, $M$ instead existentially guesses the value to be read and universally does two things: verify that the read value is correct and, in parallel, continue with the successor of the original read configuration as though the guess were correct.

To solve $S_L$, each processor uses its space to record $M$'s current configuration $Q$, whose length is proportional to the space bound of $M$ and hence $O(T)$. Initially, $Q$ is $M$'s initial configuration. In general, if $Q$ is existential (universal), the 1-processor (resp., 0-processor) chooses a successor of $Q$ that leads to acceptance (resp. rejection) of its own input. It can do so in space $O(T)$, since $\text{ATIME}(T) \subseteq \text{DSPACE}(T)$ [10]. It then communicates a single bit to the other processor indicating which of the two successors it has chosen, and they each update $Q$ accordingly. After at most $T$ communicated bits, $Q$ will be a configuration in which $M$ is reading some input character at some position $i$, at which point both processors output $i$.

By a straightforward induction on the number of steps that $M$ has taken, $M$, when begun in configuration $Q$ on the input given to the 1-processor (0-processor), will eventually accept (resp., reject). In particular, the inputs given to the processors must differ in the $i$th position, since $M$ halts immediately after reading this input.

**Corollary 17.** For any $k \geq 1$ and $L \in NC^k$, the search problem $S_L$ can be solved (uniformly) in $O(\log^k n)$ communications and $O(\log^k n)$ space.

### 6. Some Problems That Do Not Exhibit a Communication–Space Trade-off

In this section, we study sorting, ranking, and the discrete Fourier transform. We show that these problems do not exhibit any communication–space trade-offs by
exhibiting algorithms that use minimum communication and minimum space, simultaneously. The fact that these problems do not exhibit a communication-space trade-off is somewhat surprising at first glance, because these problems are known to exhibit time-space trade-offs in the single processor model. This is evidence that the problem of determining a communication-space trade-off for a given problem is inherently different from the problem of determining a time-space trade-off for the same problem.

6.1. Sorting and Ranking

In the sorting problem, the two processors input sets \( X \) and \( Y \) of integers, respectively. For convenience, assume that \( X = \{x_1, x_2, \ldots, x_n\} \) and \( Y = \{y_1, y_2, \ldots, y_n\} \) are such that \( X \cup Y \) consists of \( 2n \) distinct integers from the set \( \{0, 1, \ldots, 2^n - 1\} \). The processors sort these integers, and the \( Y \)-processor outputs them in ascending order. It is known [25] that \( \Omega(n^2) \) bits of communication are required even if each processor has unbounded space available to it. Below, we present an algorithm that requires only \( O(\log n) \) space, and \( O(n^2) \) bits of communication.

The algorithm works in \( 2n \) phases. At the end of the \( k \)-th phase, the \( k \)-th smallest integer from the set \( X \cup Y \) is output by the \( Y \)-processor. At all times, both the processors maintain pointers \( i \) and \( j \) to the smallest elements of the sets \( X \) and \( Y \), respectively, that have not been output so far. During any phase, the numbers pointed to by the pointers \( i \) and \( j \) are compared (using \( n \) bits of communication); the smaller one is output and the corresponding pointer is updated.

In order to compare \( x_i \) and \( y_j \), the \( X \)-processor starts transmitting the bits of \( x_i \), starting with the most significant bit. The \( Y \)-processor receives these bits and simultaneously begins to compare these bits to the leading bits of \( y_j \). If the result of a bit comparison is an equality, then that bit is output. As soon as the \( Y \)-processor determines the first bit position in which \( x_i \) and \( y_j \) differ, it knows which one of them is the smaller one. It outputs the rest of the smaller number and then notifies the \( X \)-processor of the inequality. Then, one of the processors updates its pointer, and the phase ends. In order to update the pointer \( i \), we must determine the smallest element in the set \( X \) that is larger than \( x_i \). This can be done easily in \( O(\log n) \) space. The pointer \( j \) can be updated in a similar manner.

The ranking problem is similar to the sorting problem. As in the sorting problem, the two processors input sets \( X \) and \( Y \) of integers. The \( X \)-processor outputs integers \( r_1, r_2, \ldots, r_n \) (in that order) such that \( r_k \) is the number of elements in the set \( X \cup Y \) that are less than \( x_k \). It is known [25] that \( \Omega(n^2) \) bits of communication are required even if each processor has unbounded space available to it. A slight modification of the sorting algorithm described above can be used to solve this problem using only \( O(\log n) \) space, and \( O(n^2) \) bits of communication.

The \( X \)-processor can easily determine the number of elements of \( X \) that are less than \( x_k \). The number of elements of \( Y \) that are less than \( x_k \) can be determined in \( O(n) \) bits of communication as follows. After determining the number of elements in \( Y_j = \{y_1, y_2, \ldots, y_j\} \) that are less than \( x_k \), the \( Y \)-processor also maintains two pointers \( l \) and \( m \) with the property that of all the elements in \( Y_j, y_m \) and \( x_k \) have
the largest common prefix, and the length of this common prefix is \( l \leq n \). Initially, \( m \) is undefined, and \( l = 0 \). In order to check if \( y_{j+1} \leq x_k \), the Y-processor compares the first \( l \) bits of \( y_m \) and \( y_{j+1} \). If this is not sufficient to determine the relation between \( x_k \) and \( y_{j+1} \), then the X-processor transmits additional bits of \( x_k \), starting with the \((n-l-1)\)th bit, until the inequality between \( x_k \) and \( y_{j+1} \) is determined. Then, the Y-processor updates the pointers \( l \) and \( m \).

This shows that the problems of sorting and ranking can be solved with linear (i.e., \( O(n^2) \)) communication even with a minimal amount of space and that there is no possibility of a nontrivial communication-space trade-off for this problem.

6.2. Discrete Fourier Transform

The eight-point FFT (fast Fourier transform) circuit is shown in Fig. 2, along with two possible ways of distributing the inputs to the two processors. In general, the \( 2^k \)-point FFT circuit has inputs labeled with the binary representations of \( 0, 1, ..., 2^k - 1 \) in the natural order. There are two natural policies for distributing these inputs between the processors, namely, according to either the first or last bit of this label. To be more precise, \textit{Policy A} assigns \( \{x_0^A, x_1^A, ..., x_{2^k-1}^A\} \) to the x-processor and \( \{y_0^A, y_1^A, ..., y_{2^k-1}^A\} \) to the y-processor, where \( x_i^A \) is the input with label \( 0i \) and \( y_i^A \) is the input with label \( 1i \). On the other hand, \textit{Policy B} assigns \( \{x_0^B, x_1^B, ..., x_{2^k-1}^B\} \) to the x-processor and \( \{y_0^B, y_1^B, ..., y_{2^k-1}^B\} \) to the y-processor, where \( x_i^B \) is the input with label \((0i)^R\) and \( y_i^B \) is the input with label \((1i)^R\), where \( w^R \) is the reversal of the string \( w \). These policies are illustrated in Fig. 2.

In the remainder of this subsection, we argue that there is no communication-space trade-off under Policy A (and hence none for the general discrete Fourier

<table>
<thead>
<tr>
<th>Policy</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>( x_0^A )</td>
<td>( x_0^B )</td>
</tr>
<tr>
<td>( x_1^A )</td>
<td>( y_0^B )</td>
</tr>
<tr>
<td>( x_2^A )</td>
<td>( x_2^B )</td>
</tr>
<tr>
<td>( x_3^A )</td>
<td>( y_2^B )</td>
</tr>
<tr>
<td>( y_0^A )</td>
<td>( x_1^B )</td>
</tr>
<tr>
<td>( y_1^A )</td>
<td>( y_1^B )</td>
</tr>
<tr>
<td>( y_2^A )</td>
<td>( x_3^B )</td>
</tr>
<tr>
<td>( y_3^A )</td>
<td>( y_3^B )</td>
</tr>
</tbody>
</table>

Fig. 2. The eight-point FFT, with two policies for distributing the inputs.
transform), while there is a nontrivial communication–space trade-off for the FFT under Policy B.

Consider a 2^k-point FFT circuit where the inputs are distributed under Policy A. From the structure of the FFT circuit, it is clear that any immediate predecessor of any output can be pebbled with k pebbles and without any communication. Each such gate can be communicated exactly once to produce two outputs.

This shows that the 2^k-point FFT circuit can be pebbled with linear (i.e., 2^k) communication even with the minimum amount O(k) of space. Therefore, there in no possibility of a communication–space trade-off for this circuit under Policy A. In contrast, the following theorem demonstrates that there is a nontrivial communication–space trade-off for pebbling the FFT circuit under Policy B.

**Theorem 18.** Let the inputs be distributed between the two processors in accordance with Policy B. If a pebbling strategy for the n-point FFT circuit uses C (two-point) communication steps and 2S pebbles, then \( C = \Omega(n^2/S) \).

**Proof:** Consider the pebble game of Section 2 on an n-point FFT circuit, where the inputs are distributed between the two processors in accordance with Policy B. Assume that \( S \leq n/16 \), since the result is immediate otherwise. We can partition the whole game into \( \lceil n/4S \rceil \) phases. The \( r \)th phase begins immediately after a total 4S(t-1) outputs have been pebbled, and it ends immediately after a total of 4St outputs have been pebbled. The last phase may be an exception; it ends when all outputs have been pebbled. Observe that during each phase (except possibly the last one) exactly 4S new outputs are pebbled. We will prove that each phase requires at least 2S \( \lceil n/8S \rceil \) communication steps. This, in turn, implies the theorem.

Consider the \( r \)th phase. Let \( Z = \{z_1, z_2, \ldots, z_{4S}\} \) be the set of new outputs pebbled in this phase. Consider any set \( X = \{x_{m+1}, x_{m+2}, \ldots, x_{m+4S}\} \) of 4S consecutive (according to Policy B) inputs of the x-processor. By [26, Lemma 3] and Lemma 5, there are 4S vertex-disjoint paths from \( X \) to \( Z \).

Let \( U \) be the set of gates that are immediate successors of some vertex in \( X \) and have a pebble-free path to one of the outputs in \( Z \). Let \( |U| = 4S - a \). Since each pebble can block only one of the 4S vertex-disjoint paths from \( X \) to \( Z \), there must be at least a pebbles on non-input vertices. All the gates in \( U \) must be pebbled in the \( r \)th phase. Since there are at most 2S - a pebbles on the circuit inputs, pebbling all the vertices in \( U \) requires at least \( |U| = (2S - a) = 2S \) communication steps. Since this is true for each of the \( \lceil n/8S \rceil \) blocks of 4S consecutive inputs to the x-processor and each of the \( \lceil n/4S \rceil \) phases, the total number of communication steps is \( C \geq 2S \lceil n/8S \rceil \lceil n/4S \rceil = \Omega(n^2/S) \), since \( S \leq n/16 \).

In this argument, we used the fact that a certain submatrix of the discrete Fourier transform matrix is nonsingular. As an alternative, we could use the fact that the FFT circuit is a grate in order to prove Theorem 18. (See Tompa [26].)

Finally, it is worth pointing out that, for any set of linear forms under any policy for distributing the inputs to the two processors, there exists a circuit computing
these forms that can be pebbled simultaneously in $O(1)$ pebbles and a linear number of communication steps. The same is true of several other problems, e.g., the reduced sensitivity analysis problem of Bentley and Brown [4].

7. OPEN PROBLEMS

There are numerous directions for future research suggested by this work:

1. Prove Theorem 14 (or some other trade-off in the Boolean straight-line model) for two-way communication. This appears to require new techniques.

2. The lower bounds presented in this paper were all for the straight-line model. It would be interesting to prove communication-space trade-offs for nonoblivious algorithms, perhaps using branching program techniques previously employed in proving time-space trade-offs by Borodin et al. [5, 7]. Unlike those results, the problem exhibiting the communication-space trade-off will not be sorting.

3. Prove a communication-space trade-off for a single-output function. The only examples of this in the time-space trade-off literature are the results on element distinctness [6, 29], which provably has no communication-space trade-off. Corollary 11 shows that any straight-line protocol that multiplies two $n \times n$ matrices $A$ and $B$ in $O(\log n)$ space requires $\Omega(n^3/\log n)$ communication steps. Suppose that such a trade-off could be proved for the following related decision problem: One processor is given nonsingular matrices $A$ and $C$, and the other is given $B$, and the problem is to decide if $AB = C$. Suppose it could be proved that this problem cannot be solved simultaneously in space $O(\log n)$ and communication $O(n^2)$. Then it would follow that $n \times n$ matrices cannot be inverted in space $O(\log n)$, since one possible protocol for the decision problem is for the first processor to compute $A^{-1}C$, which it then transmits to the other processor in $n^2$ communication steps for direct comparison to $B$. (Tiwari [24] had posed a similar open problem, namely, proving that $CS = \Omega(n^2)$ for determining whether $xy \equiv 1 \pmod{2^n}$ for $n$ bit integers $x$ and $y$. However, this problem can be solved in $O(n)$ communications and $O(\log^2 n)$ space, by computing $x^{-1} \pmod{2^n}$ using the algorithm of von zur Gathen [12].)

4. The lower bounds proved here were all for deterministic algorithms. Is there a problem whose simultaneous communication and space complexity is decreased when randomization is allowed? This would be analogous to the result of Mehlhorn and Schmidt [18] on communication steps alone.

5. Section 5 adds to the motivation for studying the communication complexity of Karchmer and Wigderson's search problems. For instance, showing that some search problem $S_L$ cannot be solved in $O(\log n)$ communications and $O(\log n)$ space simultaneously would show that $L \notin NC^1$. 
6. Finally, all these studies can be taken to the general case of more than two processors, where one might measure the four resources of space, communication steps, computation steps, and number of processors.

ACKNOWLEDGMENTS

We thank Michael Loui for discussions that led to the formulation of these problems, Jim Shearer for pointing out Lemma 12, and Joachim von zur Gathen for helpful discussions.

Note added in proof. Problem 1 of Section 7 has been solved by Beame, Tompa, and Yan [31]. A technique from Mansour, Nisan, and Tiwari [32] has been used in the solution.

REFERENCES


