UNIVERSITÄT TRIER
Mathematik / Informatik

Forschungsbericht Nr. 95–10

The graph of integer multiplication is hard for read-k-times networks

Stasys Jukna

Electronic copies of technical reports are available:

Via FTP:	Host ftp.informatik.uni-trier.de, directory /pub/reports
Via gopher:	Host gopher.informatik.uni-trier.de, port 70, type 1, pathes 1/TechReports/Abstracts and 1/TechReports/FullText
Via WWW:	URL http://www.informatik.uni-trier.de/Reports/List
Via email:	Send a mail to ftpmail@ftp.informatik.uni-trier.de, subject 'MAIL ME CLEAR', body 'TechReports.HowTo' followed by an empty line, for detailed instructions

Printed copies:

Trierer Forschungsberichte
Fachbereich IV -
Mathematik / Informatik
Universität Trier
D-54286 Trier

ISSN 0944-0488

THE GRAPH OF INTEGER MULTIPLICATION IS HARD FOR READ-K-TIMES NETWORKS

STASYS JUKNA[1]

Abstract. We prove that the graph of integer multiplication requires non-deterministic read-k-times branching programs of exponential size. On the other hand we show that one can add polynomially many integers by small deterministic read-once-only branching programs. This shows that the reason for the hardness of multiplication is *not* the necessity to add many integers (and hence, to get rid of the carry numbers) but the necessity to add different subsets of these integers.

Key words. Branching programs, integer addition, integer multiplication, linear codes, circuit complexity, lower bounds problem.

1. Introduction

It is known that the graph of multiplication is hard for depth-2 threshold circuits (Hajnal *et al.* 1987) and for AC^0 circuits (Buss 1992). In this paper[2] we prove that the graph of integer multiplication function is hard for non-deterministic read-k-times branching programs.

Read-k-times branching programs have gained a considerable attention during the last decade. There are at least two reasons for this. The first reason is that this model captures space limitations of so-called eraser Turing machines which erase each input cell after a fixed number k of readings. Another reason is that such programs have become an important tool in computer-aided design of digital circuits (see, e.g. Bryant 1992 for a survey). Restricted versions of such programs (namely, oblivious read-k-times programs) have one nice property: logical operations ($\wedge, \vee, \oplus$, etc) can be effectively applied directly to such programs. There are packages based on reduced oblivious read-once-only branching program (OBDD, for short[3]) representation which work well for design and verification of concrete circuits.

[1]Research supported by DFG-project Me 1077/5-2.

[2]This is a revised version of the manuscript distributed in April 1994.

[3]In computer-aided design (CAD) literature the term "binary decision diagram" is used instead

Although good in many situations, these packages cannot verify some very important circuits. One such example is the integer multiplication. Up to now all the attempts to verify 32-bit multiplicator have failed. Bryant (1991) explained this failure by proving that the n-th bit in the binary representation of $\mathbf{x} \cdot \mathbf{y}$, is hard for oblivious read-once-only programs. Bollig *et al.* (1993) and Gergov (1994) observed that Bryant's result combined with known lower bounds arguments for oblivious branching programs implies that the multiplication is hard also for oblivious read-k-times programs even with $k = o(\log n)$. The requirement for a program to be oblivious is very strong restriction: in each s-t path the variables must appear in the same (fixed for the program) order. It is therefore natural to ask if the multiplication can be done by a small (polynomial) size *non-oblivious* read-k-times branching program.

The lower bound argument for oblivious branching programs is based on the observation that any such program for f must be large if, for *at least one* partition of inputs into two sets, the communication complexity of f is large. Bryant takes a partition where Alice gets $\mathbf{x}$ and Bob gets $\mathbf{y}$, and proves that (under this partition) they need $\Omega(n)$ bits of communication to compute the n-th bit of $\mathbf{x} \cdot \mathbf{y}$, Unfortunately, this construction holds only for this very special partition of input bits and fails for others. Recently Ponzio (1995) has found another argument which works for *any* partition where Alice receives $\Theta(\sqrt{n})$ less significant bits of both integers $\mathbf{x}$ and $\mathbf{y}$, and Bob receives the remaining bits. This, combained with the generalization of "cut-and-paste" method by Simon and Szegedy (1993), leads to a $2^{\sqrt{n}}$ lower bound for the n-th bit of $\mathbf{x} \cdot \mathbf{y}$ in the class of deterministic read-once-only branching programs.

Our main result is the following lower bound for the graph of integer multiplication in the calss of non-deterministic read-k-times branching programs. We will identify binaty vectors $\mathbf{x} = (x_1, \ldots, x_n)$ in $\{0,1\}^n$ with the corresponding integers $\mathbf{x} = \sum_{i=1}^{n} x_i 2^{n-i}$. The projection of $\mathbf{x}$ onto a set of coordinates $S = \{i_1, \ldots, i_m\}$ is the vector $\mathbf{x}\rceil_S = (x_{i_1}, \ldots, x_{i_m})$. Given a subset S, the corresponding *graph* of multiplication is the Boolean function $mult^n_S$ defined by $mult^n_S(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 1$ iff $(\mathbf{x} \cdot \mathbf{y})\rceil_S = \mathbf{z}$.

THEOREM 1. *For any sufficiently large $n \geq 0$ there exists an n-bit integer $\mathbf{a}$ and subset $S \subseteq \{1, \ldots, n\}$, $|S| \leq \sqrt{n}$, such that every non-deterministic read-k-times branching program computing $mult^n_S(\mathbf{a}, \mathbf{y}, \mathbf{0})$ has size at least* $\exp\left(\Omega(n^{1/4}/k^{2k})\right)$.

Our proof follows the framework of (Borodin *et al.* 1993) and is done via simple reduction of $mult^n$ to the problem of recognizing codewords of a linear code, for which a lower bound of $\exp\left(\Omega(\sqrt{n}/k^{2k})\right)$ is proved in (Jukna 1993).

Then we show that the integer addition of polynomially many n-bit numbers *can* be done by small deterministic read-once-only branching programs (= OBDDs). This shows that the reason for the hardness of multiplication is *not* the necessity to add many integers (and hence, to get rid of the carry bits) but the necessity to add many

of "branching program". OBDD stands for "Ordered Binary Decision Diagram"; this is exactly the oblivious deterministic read-once-only branching program.

shifts of the same integer (integer **a** from the theorem), and hence, the necessity to get rid of shifting.

We consider well known model of branching programs. Recall that such a program is a directed acyclic graph G with a distinguished source node s and a distinguished sink node t. The outdegree of each non-sink node is at most 2. Each edge is labeled by some variable or its negation. If a vertex has outdegree 2 then the two outgoing edges are labeled by x_i and $\neg x_i$. The branching program becomes *nondeterministic* if we allow "guessing nodes" that is nodes with both two outgoing edges being *unlabeled*. The size(G) is the number of *labeled* edges in G. A program is *oblivious* if the underlying graph G is levelized so that the edges from the i-th level go only to the $i+1$-th level and all are labeled by the same variable. The program G computes a Boolean function $f:\{0,1\}^n \to \{0,1\}$ in the obvious way: for each $\mathbf{u} \in \{0,1\}^n$ we let $f(\mathbf{u})=1$ iff there exists at least one (directed) path starting in the source node s and leading to the sink node t and such that all labels along this path are consistent with $\mathbf{u}$.

A program is *read-k-times* program if each variable occurs at most k times along *each* path going from s. We denote the corresponding complexity measures by $BP_k(f)$ and $NBP_k(f)$. It is clear that $BP_k(f) \le NBP_k(f)$ for every function f. It is also known (see Jukna 1993) that non-deterministic model is much more powerful than the deterministic one: there is a sequence of Boolean functions $\langle f_n | n \ge 0\rangle$ such that $NBP_1(f_n) = O(n)$ but $BP_k(f_n) = \exp(\Omega(n^\epsilon))$ for any $k = o(\log n)$.

2. Multiplication is hard: the proof of Theorem 1

In this section we prove the lower bound stated in Theorem 1. We prove this lower bound in two steps: we reduce the function $mult^n_S(\mathbf{a},\mathbf{y},\mathbf{0})$ to linear codes and then apply the general lower bound on the read-k-times complexity for (general) codes proved in (Jukna 1993).

2.1. Reduction to linear codes. The *cyclic shift* of a vector $\mathbf{x}=(x_1,\ldots,x_n)$ is the vector $\mathbf{x}'=(x_2,\ldots,x_n,x_1)$. Given a vector $\mathbf{x}=(x_1,\ldots,x_n)$, we denote by $\mathtt{M}(\mathbf{x})$ the cyclic n by $2n$ matrix, the first row of which is $(0,\cdots,0,x_1,\ldots,x_n)$, and all other rows are obtained by shifting this vector ($i+1$-th row is the cyclic shift of the i-th one):

$$\mathtt{M}(\mathbf{x}) = \begin{pmatrix} 0 & \cdots & 0 & 0 & x_1 & \cdots & x_{n-1} & x_n \\ 0 & \cdots & 0 & x_1 & x_2 & \cdots & x_n & 0 \\ & \vdots & & & & \vdots & & \\ x_1 & \cdots & & x_{n-1} & x_n & \cdots & 0 & 0 \end{pmatrix}.$$

The multiplication matrix is the $|\mathbf{y}|$ by $2n$ submatrix[4] $\mathtt{M}(\mathbf{x},\mathbf{y})$ of $\mathtt{M}(\mathbf{x})$ defined by the set $\{i : y_i = 1\}$, i.e. the i-th row of $\mathtt{M}(\mathbf{x})$ is in $\mathtt{M}(\mathbf{x},\mathbf{y})$ iff $y_i = 1$. The matrix $\mathtt{M}(\mathbf{x},\mathbf{y})$ is what one uses when multiplying the numbers $\mathbf{x}$ and $\mathbf{y}$ "by hand". Namely, the k-th bit z_k of $\mathbf{x}\cdot\mathbf{y}$ is

$$z_k = s_k + \sum_{j=1}^{k-1} s_j \cdot 2^j \ \left(\mathrm{mod}\, 2^k\right)$$

where s_j is the sum of ones in the j-th (from the right) column of $\mathtt{M}(\mathbf{x},\mathbf{y})$.

Recall that a set $C \subseteq \{0,1\}^r$ is a linear (r,s,d)-code if there is an $r \times s$ "parity check" matrix H over the field $\mathsf{F}_2 = GF(2)$ such that:

(i) $\mathbf{u} \in C$ iff $\langle \mathbf{u},\mathbf{v}\rangle = 0$ for all s columns $\mathbf{v}$ of H (here $\langle \mathbf{u},\mathbf{v}\rangle = u_1v_1 \oplus \cdots \oplus u_nv_n$ is the standard skalar product over F_2), and

(ii) any $d-1$ rows of H are linearly independent (over F_2).

Any linear (r,s,d)-code has 2^{r-s} vectors and the Hamming distance between any two of them is at least $2d+1$.

In what follows f_C will stand for the characteristic function of a subset $C \subseteq \{0,1\}^r$, i.e. $f_C(\mathbf{u}) = 1$ iff $\mathbf{u} \in C$. A function g is a *subfunction* of f if g can be obtained from f by assigning constants to some of its variables.

We now turn to the proof that (characteristic functions of) linear codes are subfunctions of $mult^n_S(\mathbf{a},\mathbf{y},\mathbf{0})$ for appropriate n-bit integer $\mathbf{a}$ and subset of coordinates S. We do this in two steps. First, we prove that the parity check matrix H of C is a submatrix of $\mathtt{M}(\mathbf{a},\mathbf{b})$ for some fixed integers $\mathbf{a}$ and $\mathbf{b}$. Second, we show that, for some set of coordinates S, the value of $\mathbf{a}\cdot\mathbf{y}$ on all these coordinates are 0 iff the vector $\mathbf{y}\wedge\mathbf{b} = (y_1 \wedge b_1, \ldots, y_n \wedge b_n)$ is orthogonal to all the columns of this submatrix. Set $p = \log n + 1$ and $n = rsp$.

CLAIM 2. *There exist two n-bit vectors $\mathbf{a},\mathbf{b} \in \{0,1\}^n$ such that $|\mathbf{b}| = r$ and the matrix $\mathtt{M}(\mathbf{a},\mathbf{b})$ contains the submatrix*

$$H^* = [\mathbf{v}_1 \,|\, \Box \,|\, \mathbf{v}_2 \,|\, \Box \,|\, \cdots \Box \,|\mathbf{v}_s \,|\, \Box\,]$$

where $\mathbf{v}_1,\ldots,\mathbf{v}_s$ are the columns of H and $\Box$ is the $r \times p$ matrix of nulls.

PROOF. Take the vector $\mathbf{w} = (\mathbf{u}_1 \ldots \mathbf{u}_n) \in \{0,1\}^{rs}$ where $\mathbf{u}_1,\ldots,\mathbf{u}_r$ are the rows of the matrix H. Let

$$\mathbf{a} = \left(w_1\vec{0}w_2\vec{0}\cdots w_{n-1}\vec{0}w_n\right)$$

where $\vec{0}$ denotes p many 0's. Since the rows of $\mathtt{M}(\mathbf{a})$ are cyclic shifts of the first row $(0\ldots0\mathbf{a})$, the submatrix $\mathtt{M}(\mathbf{a},\mathbf{b})$ with $b_i = 1$ iff $i \in \{1, sp, 2sp, \ldots, (r-1)sp\}$, contains H^* as a submatrix. ∎

[4] $|\mathbf{y}| = \sum_i y_i$ is the number of 1's in $\mathbf{y}$.

CLAIM 3. *For any linear (r,s,d)-code C there is a subset $S \subseteq \{1,\ldots,2n\}$ such that the function f_C (on r variables) is a subfunction of $mult^n_S(\mathbf{a},\mathbf{y},\mathbf{0})$ (on $n+|S|$ variables).*

PROOF. Let S be the set indices of columns $\mathbf{v}_1,\ldots,\mathbf{v}_m$ of H in the matrix $\mathtt{M}(\mathbf{a},\mathbf{b})$. Take $J=\{j\,:\,b_j=1\}$ and consider the function $f_{\mathbf{a},\mathbf{b}}:\{0,1\}^J\to\{0,1\}$ defined by:

$$f_{\mathbf{a},\mathbf{b}}(\mathbf{y}) = mult^n_S(\mathbf{a},\mathbf{y}\wedge\mathbf{b},\mathbf{0})$$

where $\mathbf{y}\wedge\mathbf{b}$ is the vector in $\{0,1\}^n$ which coincides with $\mathbf{y}$ on J and has nulls elsewhere. The function $f_{\mathbf{a},\mathbf{b}}$ is the Boolean function on $|J|=|\mathbf{b}|=r$ variables. Observe that for each $j\in S$, the j-th bit z_j of $\mathbf{z}=\mathbf{a}\cdot\mathbf{b}$ is simply the sum modulo 2 of the bits in the corresponding column $\mathbf{v}_j$ of $\mathtt{M}(\mathbf{a},\mathbf{b})$. This is because the carry number obtained at previous column of H in $\mathtt{M}(\mathbf{a},\mathbf{b})$ has binary length at most $\log n$, and hence, has no influence to the value of z_j since we have $p>\log n$ null-columns between these columns. Hence, $z_j=\langle\mathbf{y},\mathbf{v}_j\rangle$ for all $j\in S$. Thus, $f_{\mathbf{a},\mathbf{b}}(\mathbf{y})=1$ iff the vector $\mathbf{y}$ is orthogonal to all the columns of H, and hence, $mult^n_S(\mathbf{a},\mathbf{y}\wedge\mathbf{b},\mathbf{0})=1$ iff $f_{\mathbf{a},\mathbf{b}}(\mathbf{y})=1$ iff $\mathbf{y}\in C$ iff $f_C(\mathbf{y})=1$. □

2.2. The lower bound. By claims 2 and 3, for any linear (r,s,d)-code C, the function f_C (on r variables) is a subfunction of the function $mult^n_S$ on $2n+|S|\le 3n=3rsp$ variables where $p=\log r+1$.

To complete the proof of Theorem 1, let $C\subseteq\{0,1\}^r$ be the BCH-code. For this code we have that $s\le d\log(r+1)$. It was proved in (Jukna 1993, Corollary 1) that, for $d=\Theta\left(\sqrt{r/k^k}\right)$, these codes require exponential size read-k-times networks:

$$NBP_k\left(f_C\right)\ge\exp\left(\Omega(\sqrt{r}/k^{2k})\right). \tag{1}$$

The function $mult^n_S(\mathbf{a},\mathbf{y},0)$ itself has $n=rsp=O\left(r^{3/2}\left(\log r\right)^2\right)=o(r^2)$ variables. Since f_C is the subfunction of $mult^n_S(\mathbf{a},\mathbf{y},\mathbf{0})$, the bound (1) immediately gives the bound

$$NBP_k\left(mult^n_S\right)\ge\exp\left(\Omega(n^{1/4}/k^{2k})\right). \tag{2}$$

which completes the proof of Theorem 1. □

Remark It is easy to see that our lower bound holds also for the following problem related to integer addition:

$$Add_S(\mathbf{y})=1 \iff \sum_{i:y_i=1}\mathbf{a}_i\in S$$

where $\mathbf{a}_1,\ldots,\mathbf{a}_n$ are fixed integers and S is some (also fixed) set of powers of 2. Thus, it is hard to add different subsets of the same (fixed) set of integers. On the other hand, the addition of many integers itself is extremely easy (even for oblivious read-once branching programs).

3. Addition is easy

Integer addition is an important component of the multiplication. Namely, the product $\mathbf{x} \cdot \mathbf{y}$ of two n-bit numbers $\mathbf{x}$ and $\mathbf{y}$ is simply the sum of n $(2n)$-bit integers represented by the rows of the matrix $\mathtt{M}(\mathbf{x}, \mathbf{y})$ considered above.

The aim of this section is to show that the addition of many integers can be performed effectively with oblivious deterministic read-once-only branching programs. This shows that the reason for the hardness of multiplication is *not* the necessity to add many integers (and hence, to get rid of the carry numbers) but the necessity to add different *shifts* of the same integer.

Given n-bit integers $\mathbf{x}^1, \ldots, \mathbf{x}^m \in \{0,1\}^n$, their sum is the $(n + \log m)$-bit integer $\mathbf{x}^1 + \ldots + \mathbf{x}^m$. (Recall that we identify vectors $\mathbf{x} \in \{0,1\}^n$ with integers $\mathbf{x} = \sum_{i=1}^{n} x_i 2^{n-i}$.) For $S \subseteq \{1, \ldots, n + \log m\}$, the corresponding *graph of addition* is the Boolean function add_S defined by $add_S(\mathbf{x}^1, \ldots, \mathbf{x}^m, \mathbf{z}) = 1$ iff $(\mathbf{x}^1 + \ldots + \mathbf{x}^m)\rceil_S = \mathbf{z}$. Note that, for each ν, the function $add_\nu(\mathbf{x}^1, \ldots, \mathbf{x}^m) = add_{\{\nu\}}(\mathbf{x}^1, \ldots, \mathbf{x}^m, 1)$ is the ν-th bit in the binary representation of the sum $\mathbf{x}^1 + \cdots + \mathbf{x}^m$. Let $OBDD(f)$ denote the minimum size of an oblivious deterministic read-once-only branching program computing f.

PROPOSITION 4. *For any S, $OBDD\,(add_S) = O(nm^2)$.*

PROOF. We describe the desired program for a function $add_\nu(\mathbf{x}^1, \ldots, \mathbf{x}^m)$, with ν arbitrary but fixed coordinate. The program then can be easily modified (without increasing the number of nodes) to compute add_S for any subset S.

The idea is simple and is based on the following observation. The main difficulty in computing the sum of many, say m, numbers is to get rid of the $\log m$ "carry" bits. Read-once-only programs have no "memory tape" where they could keep the results obtained so far and use them later without reading once more the input. It appears, however, that such a programs have "inner memory": they can keep the information not only in the labels of their nodes but also in the *topology* of underlying graphs.

We will add m integers $\mathbf{x}^1, \ldots, \mathbf{x}^m$ in a usual manner — column by column from the right to the left. (We look vectors $\mathbf{x}^1, \ldots, \mathbf{x}^m$ as rows). At the t-th step we compute the t-th "carry number" c_t which is defined as follows.

Let $\mathbf{s}$ be the binary representation of the sum of integers represented by initial parts (from the right) of length t of vectors $\mathbf{x}^1, \ldots, \mathbf{x}^m$. This vector $\mathbf{s}$ has length at most $t + \log m$ and has all the information about the summ over the first t columns. We may assume w.l.o.g. that $\mathbf{s} \in \{0,1\}^{t+\log m}$ (the left bits of $\mathbf{s}$ can be 0's). Define the *t-th carry number* c_t as the number represented by the first $\log m$ bits of this sum $\mathbf{s}$ (from the left). Thus,

$$\mathbf{s} = (\underbrace{s_1, \ldots, s_{\log m}}_{c_t}, s_{\log m+1}, \ldots, s_{\log m+t}).$$

For a vector $\mathbf{x} = (x_1, \ldots, x_n)$ its *shift* is the vector $\mathbf{x}' = (0, x_1, \ldots, x_{n-1})$. Observe that the t-th carry number c_t is obtained by adding the $(t-1)$-th carry number c_{t-1} to

the sum $\sum_{i=1}^m x_t^i$ of the bits in the t-th column and shifting the resulting integer one bit to the right, i.e.

$$c_t = \left(c_{t-1} + \sum_{i=1}^m x_t^i \right)' .$$

The desired oblivious read-once program P for the ν-th bit $add_\nu(\mathbf{x}^1, \ldots, \mathbf{x}^m)$ in the binary representation of the sum $\mathbf{x}_1 + \cdots + \mathbf{x}_m$ looks as follows. The program has ν levels. The t-th level consists of $m+1$ programs P_c^t, $c \in \{0, 1, \ldots, m\}$. The program P_c^t looks at the t-th column and computes the t-th carry number *assuming that c was the $(t-1)$-th carry number.* Namely, P_c^t depends on the variables $x_t^1, x_t^2 \ldots, x_t^m$ and has $m+1$ outputs $v_0, v_1, \ldots, v_m$ where for $b \in \{0, 1, \ldots, m\}$,

$$v_b = \begin{cases} 1 & \text{if } \ b = (c + \sum_{i=1}^m x_t^i)' \\ 0 & \text{if otherwise} \end{cases}$$

The output node v_b of P_c^t is the start node of the program P_b^{t+1} on the next $(t+1)$-th level.

Thus, being at the t-th level corresponds to looking at the t-th column (from the right) of the $m \times n$ matrix with rows $\mathbf{x}^1, \ldots, \mathbf{x}^m$. Simple but crusial observation is that being in the program P_c^t of this level we know that *c is the $(t-1)$-th carry number computed so far.* Exactly one output of the program P_c^t is evaluated to 1. This is the b-th output v_b where $b = c + \sum_{i=1}^m x_t^i$. Thus $v_b = \mathsf{Exact}_{b-c}(x_t^1, \ldots, x_t^m)$, where Exact_r computes 1 iff exactly r inputs take the value 1.

The programs from different levels depend on different variables corresponding to different columns. Each program P_c^t computes a collection of Exact_r functions ($r = 0, 1, \ldots, m$) and can be realized as an oblivious deterministic read-once-only branching programs of size $O(m^2)$. Thus, the total size of P is $O(tm^3)$. □

4. Concluding comment

We have shown that, for some subsets S, the graph of multiplication $mult_S^n$ cannot be computed by read-k-times non-deterministic branching program of polynomial size.

It is not readily apparent which problem is harder for *non-deterministic* read-k times networks, the graph or one bit of $\mathbf{x} \cdot \mathbf{y}$. For $k = 1$ and *deterministic* model we know that both, the graph and one bit, of $\mathbf{x} \cdot \mathbf{y}$ are hard (this follows from our lower bound together with the bound of Ponzio 1995). But what about non-deterministic model? Even the case $k = 1$ remains unclear: is one bit of $\mathbf{x} \cdot \mathbf{y}$ hard for *non-deterministic* read-once branching programs?

Intuitively, the graph seems to be harder since it requires *every* bit of $\mathbf{z}$ (in S) to be the desired bit in the binary representation of $\mathbf{x} \cdot \mathbf{y}$. In models closed under the AND operation (like AC^0 or threshold circuits) this can be done easily (and graphs are therefore not harder in these). But read-k-times models are *not* closed under AND operation.

And indeed, in read-k-times models there *exist* multi-output functions, the graphs of which are harder than the functions themselves. To be more specific, consider the following function f in n^2 Boolean variables, represented as a $n \times n$ matrix $X = \{x_{ij}\}$. Define: $F(X) = \mathbf{z}$ iff for $k \in \{1, \ldots, n\}$, $z_k = 1 \iff \sum_{j=1}^{n} x_{kj} = 1$, and for $k \in \{n+1, \ldots, 2n\}$, $z_k = 1 \iff \sum_{i=1}^{n} x_{i,k-n} = 1$. Thus, the function has $2n$ outputs, each of which is very simple test: "does the corresponding row or column of X has exactly one 1". Thus, every bit of $\mathbf{z} = F(X)$ can be computed easily by a read-once branching program. But its graph $f(X, \bar{1}) = 1 \iff F(X) = \bar{1}$ is "Exact-Perfect-Matching" function which is well known (see Jukna 1989 or Krause *et al.* 1991) to require non-deterministic read-once branching programs of size $\exp(\Omega(n))$.

References

Bollig, B., Sauerhoff, M., Sieling, D., and Wegener, I. Read k times ordered binary decision diagrams — efficient algorithms in the presence of null-chains. Technical Report Nr. 474, Universität Dortmund, 1993.

Borodin A., Razborov A. and R. Smolensky. On lower bounds for read-k times branching programs, *Computational Complexity*, **3** (1993), 1-18.

Bryant, R. E. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24:3 (1992), pp. 293-318.

Bryant, R. E. On the complexity of VLSI implementations and graph representations of Boolean functions with applications to integer multiplication. *IEEE Trans. on Computers*, 40:2 (1991), pp. 205-213.

Buss, S. R. The graph of multiplication is equivalent to counting. *Information Processing Letters*, 41 (1992), pp. 199-201.

Hajnal, A., Maass, W., Pudlák, P., Szegedy, M., & Turán, G. Threshold circuits of bounded depth. In *Proc. of28-th Symposium on Foundations of Computer Science*, (1987), pp. 99-110.

Gergov, J. Time-space tradeoffs for integer multiplication on various types of input oblivious sequential machines. Manuscript, March,1994.

Jukna, S. The effect of null-chains on the complexity of contact circuits, *Springer Lecture Notes in Computer Science*, **380** (1989), 246-256.

Jukna, S. A note on read-k times branching programs, *RAIRO Theoretical Informatics and Applications*, vol. 29, N. 1 (1995), pp. 75-83.

Krause M., Meinel Ch. and Waack S. Separating the eraser Turing machine classes L_e, NL_e, $co - NL_e$ and P_e, *Theor. Comp. Sci.*, **86** (1991), 267-275.

Ponzio, S. A lower bound for integer multiplication with read-once branching programs. In: *Proceedings of the 127h Annual ACM Symposium on Theory of Computing* (1995).