Lower bounds on the computational efficiency of optical computing systems

Richard Barakat and John Reif

A general model for determining the computational efficiency of optical computing systems, termed the VLSIO model, is described. It is a 3-D generalization of the wire model of a 2-D VLSI with optical beams (via Gabor's theorem) replacing the wires as communication channels. Lower bounds (in terms of simultaneous volume and time) on the computational resources of the VLSIO are obtained for computing various problems such as matrix multiplication.

I. Introduction

The advent of very large scale integrated (VLSI) circuitry has led to considerable decrease in the physical size of computers with a corresponding increase in speed of execution of operations. The development of algorithms which utilize the inherent capabilities of such chips has led to questions of the ultimate power of VLSI.

The computational constraints of VLSI were first investigated by Thompson.[1] For an introduction to this work see the basic text of Ullman[2] which contains references to subsequent work. It has been shown that any VLSI circuit with area A and time T requires at least $AT^2 = \Omega(n)$ to solve various computational problems such as FFT, convolution, and $m \times m$ matrix multiplication where $n = m^2$. The symbol Ω is defined in Ref. 2, $f(n) = \Omega[g(n)]$ means that a positive constant c exists such that for an infinite number of values of n we have $f(n) \geq cg(n)$.

VLSI suffers from the limitation that the technology on which it relies is inherently two-dimensional. Snyder's recent review[3] contains a very useful discussion of the constraints imposed on VLSI as regards planarity. In particular, conventional VLSI chips are constructed by superimposing a small number of layers on top of a substrate. This substrate has a thickness which is an order of magnitude greater than the size of the transistors and wire width. Input and output from a conventional VLSI chip must be made by a limited number of pads located on the sides of the chip. VLSI chip technology is changing almost daily; however, some of the more basic aspects are discussed in Barbe[4] and Einspruch.[5] Although an ensemble of 2-D chips can be placed on top of each other with holes drilled down through them for interchip communication, the total number of layers is seriously limited by the substrate thickness of each chip: consequently the resulting device cannot be properly termed 3-D VLSI. For this reason, it appears that truly 3-D VLSI will most likely not be possible to fabricate. Nevertheless some interesting theoretical investigations of 3-D VLSI have been carried out.[6,7]

The purpose of this paper is to investigate the computational performance of 3-D devices which make hybrid use of electronic and optical components to perform operations. In addition to the 3-D character of the problem, we have to deal with the replacement of electrical currents by optical beams. Our goal is to facilitate general statements on such electrooptical computations ith specific reference to lower bounds on their complexity. Since such devices may contain a large number of components, we term them VLSIO, with the O denoting optics. We note that a very useful overview of optical computing (more properly electrooptical computing) may be found in Caulfield *et al.*[8]

II. Preliminaries

To carry out such an analysis we outline the development of an abstract model of VLSIO which is essentially technology independent but incorporates the physical restrictions of light beam propagation as expounded by Gabor,[9] especially with respect to the very important fact that the amount of information carried by an optical beam is bounded. This physical constraint

The authors are with Harvard University, Division of Applied Sciences, Cambridge, Massachusetts 02138.

allows us to adapt previous VLSI lower-bound arguments to the VLSIO situation and allows for comparisons of electrooptical computing devices in terms of their volume V and the time T taken by VLSIO on a given input ($\equiv$ number of time units that elapse from the first input signal until the last output signal). We avoid making assumptions about the precise physics of the devices utilized as this would only limit the later application of these ideas as the physical models are improved and modified. We assume that any 2-D convolution of an $n \times n$ array of points can be achieved by a VLSIO device in unit time step. This assumption is reasonable because optical devices already exist which perform thusly and our lower bounds will even hold in this case.

We begin by discussing the well-known abstract 2-D model of a VLSI chip as an $L_1 \times L_2 \times L_3$ grid graph with height L_3 ($\ll L_1$ or L_2) held constant. The distance between grid points is w, the feature width. The chip processors are located at various distance nodes of the grid graph with each processor storing a state consisting of b bits. Furthermore the processors execute synchronously on a step consisting of a time unit duration τ seconds. The remaining nodes are used for wire routing or for input and output pods. Each wire can run along a path in the grid graph from an input pod, or a processor, to various output pods, or processors. Wires are not allowed to intersect. On each time step, a value consisting of b bits of information is transmitted across the wire grid from either an inpout pod or a processor. The state of each processor is then updated on each step by a fixed function of the values transmitted by the wires leading into the processor, and by the state of the processor, in the previous step. This unit step transmission time across wires is justified by the fact that wire transmission can be made generally faster than transistor switching times. This remarkably simple model is sufficient to determine the computational efficiency of VLSI devices.

Following the 2-D version, the fundamental building block of our VLSIO device is the optical box B. It is parallelepiped having lengths L_1, L_2, and L_3 with input and output faces F_{in} and F_{out}. These faces are assumed to take as input and as output 2-D integer arrays $I(x,y)$ and $O(x,y)$, respectively. For convenience, we consider the input sources and output detectors to be very small compared to the size of the optical box (in order to minimize optical diffraction effects), furthermore they are uniformly spaced a distance w apart (where again w is the feature width, a fixed constant). The input sources are taken to be LEDs and the detectors are unspecified except to state that they are sensitive only to the intensity of the LED radiation, assumed given within only b bits accuracy, where b is a fixed constant. Thus we may assume for simplicity that all variables and functions are Boolean as utilized in our lower-bound construction. The ancillary optical equipment (lenses, prisms, gratings, etc.) which spreads and then collects the light can be neglected in this version of the abstract model. The output array is computed on each time step with a duration τ as a fixed function Δ_B of the input array; Δ_B will, of course, depend on the detailed optical characteristics of B. (τ is assumed to be a fixed constant.) Let a VLSIO device consist of an ensemble of optical boxes with nonintersecting interiors. However, we allow communication between optical boxes via their 2-D surfaces.

VLSIO, in addition to being three-dimensional, also differs from VLSI in another way; namely, optical beams rather than wires provide storage and cross flow. Since the modus operandi is incoherent radiation, these beams can intersect without interacting. The basic question that now arises is: "To what extent do optical beams behave as wires?" A wire can only transport information at a finite rate depending on wire cross section, skin effects, etc. We would also expect an optical beam to perform similarly notwithstanding the greater information rate. This problem has already been addressed by Gabor.[9] The conclusion he draws is that a light beam always has a finite upper limit with respect to information rates, the upper limit depending on wavelength of light, smallest effective beam area, and solid angle of divergence. Although we do not require the explicit formulas (for our purposes it suffices that we can interpret an optical beam as a wire), technical details are outlined in the Appendix.

Given this equivalence, we turn to the important problem of determining lower bounds (in terms of simultaneous volume and time) on the computational resources required for VLSIO to solve various problems. In order not to unduly lengthen the text, it is assumed that the reader is familiar with Sec. 1.4, 2.1, and 2.2 of Ullman.[2]

III. Proof of Lower-Bound Theorem

Consider a Boolean function f with a set X of n input variables and a set Y of m output variables. Let X' be a subset of X; also let $P \equiv (X_L,X_R,Y_L,Y_R)$ where X_L,X_R and Y_L,Y_R are partitions of X and Y, respectively. We term P balanced if between one-third and two-thirds of X' lies in X_L and note it by P_b. If α and β are two input assignments, we term them a fooling pair of assignments to X if:

(1) output Y_L is distinct for input assignments $\alpha(X)$ and $\alpha(X_L)\beta(X_R)$,

(2) output Y_R is distinct for input assignments $\beta(X)$ and $\alpha(X_L)\beta(X_R)$.

In addition, let the fooling set for P be a set of assignments A of X such that for all distinct $\alpha,\beta \in A$, at least one of (α,β), (β,α) is a fooling pair.

Finally, we require that the locations and times of the input and output are given only once.

Crucial to the analysis is the concept of information content (essentially the amount of information that must cross a boundary in order to solve the problem). Formally, the information content of the Boolean function f is

$$I_f = \max_{X'} \min_{P_b} \max_{A} \log_2(|A|), \tag{1}$$

where A denotes the fooling set corresponding to P_b. The following functions (of importance in electrooptical computing) are known to have information content $I_f = \Omega(n)$ (see Ullman,[2] p. 75):

(a) n point discrete Fourier transforms and convolution.

(b) Multiplication and division of two n bit binary numbers.

(c) Any transitive problem of size n. A problem is defined to be transitive if its n inputs are partitioned into data inputs $x_1, \ldots, x_m$, and control inputs $x_{m+1}, \ldots, x_n$ so that for any permutation π of $(1, \ldots, m)$ there is a corresponding setting of the control inputs $x_{m+1}, \ldots, x_n$ and that with this setting the output $(Y_1, \ldots, Y_n) = [X_{\pi(1)}, \ldots, X_{\pi(m)}]$ is this permutation of the inputs [note that problems (a) and (b) can, for example, be shown to be transitive].

(d) Multiplication and inversion of two binary $m \times m$ matrices where $n = m^2$.

The following important result on lower bounds is due to Thompson[1] and Ullman[2]: "Any two-dimensional VLSI chip computing a Boolean function f requires simultaneous area A and time T satisfying $AT^2 = \Omega(I_f^2)$."

We now prove: Any 3-D VLSIO device computing a Boolean function f requires simultaneous volume V and time T satisfying $VT^{3/2} = \Omega(I_f^{3/2})$.

The proof is an adaptation of the 2-D technique. Let the device be a parallelepiped having dimensions $L_1 \leq L_2 \leq L_3$ with volume $V = L_1L_2L_3$. Choose X' to be the subset of X such that $I_f = I_f(X')$. For each $i = 1,2,3$ we can find a cut C_i of area

$$A_i \leq 2V/L_i \qquad i = 1; \tag{2}$$

which disconnects the device into two components each of which contains at most two-thirds, but no less than one-third, of the inputs of X'. By definition at least I_f bits must be transported across each cut; this requires time

$$T \geq I_f/A_i. \tag{3}$$

Consequently

$$V^2T^3 \geq A_1A_2A_3T^3 \geq I_f^{3/2}, \tag{4}$$

or

$$VT^{3/2} = \Omega(I_f^{3/2}), \tag{5}$$

which is the sought-for result. The main point to emphasize is that this result depends on the fact that we can treat light beams as if they were wires.

An immediate consequence of this theorem is that the $VT^{3/2} = \Omega(n^{3/2})$ lower bounds hold for optical computing on the following problems:

(a) n point convolution or n point discrete Fourier transforms,

(b) multiplication or division of two n bit binary numbers,

(c) any n input transitive problem,

(d) multiplication or inversion of two binary $m \times m$ matrices where $n = m^2$.

These results follow from the statements quoted after Eq. (1). It is important to remember that these bounds hold in spite of the fact that we allow the entire volume of each optical box in the VLSIO device to be operative.

Appendix

It is a basic principle of information theory as expounded by Shannon that a communication channel can transmit only a finite amount of data in a fixed (finite) time interval; in Gabor's language the channel constrains the signal to have only a finite number of degrees of freedom even though the signal initially had an infinite number of degrees of freedom.

Following Gabor[9] let us consider two planes a distance z apart with the first plane containing the light source. This source sends out a beam of radiation u of wavelength λ governed by the scalar wave equation

$$(\nabla^2 + k^2)u = 0. \tag{A1}$$

We could use the full vector approach but this will only affect the final answer by some proportionality factor; the qualitative conclusions are the same. This beam will be intercepted by the second plane.

The field u of the beam can be written as

$$u(x,y,z) = \sum_{n=1}^{\infty} u_n\phi_n(x,y,z), \tag{A2}$$

where the $\phi_n(x,y,x)$ are a complete set of orthogonal functions in the cross section of the beam at a distance z from the source plane. In the practical case where the source is a circular disk, the appropriate orthogonal functions are the circular prolate functions of Slepian.[10] Although the above series contains an infinite number of terms in principle, Gabor makes plausible (but does not actually prove) that the number N_G (hereby termed the Gabor number) of independent traveling wave solutions of Eq. (A1) are given by (here α,β are the angles of the rays with the coordinates x,y in this plane)

$$N_g = \frac{1}{\lambda^2} \int \int dxdyd(\cos\alpha)d(\cos\beta)$$

$$= \frac{1}{\lambda^2} A\Omega, \tag{A3}$$

where A is the cross-sectional area of the beam in this second receiving plane and Ω is the solid angle subtended by the maximum angular spread of the beam. Crudely speaking N_G is the number of degrees of freedom of the optical beam ($\equiv$ structural information). Thus the finite series in Eq. (A2) can be replaced by

$$u(x,y,z) = \sum_{n=1}^{N_G} u_n\phi_n(x,y,z), \tag{A4}$$

reflecting the fact that $u(x,y,z)$ has only a finite number of degrees of freedom and thus is only capable of transmitting a finite amount of data in the sense that only the low frequency components of $u(x,y,z)$ are viable, in particular, only those with wavelength feature width w. See Ref. 11 for valuable supplementary information on Gabor's theorem.

We should point out that the number of degrees of freedom as counted by Gabor is very crude. See Newsam and Barakat[12] for a different approach to the concept of degree of freedom via the essential dimension of a compact operator which includes a more refined counting.

Richard Barakat was supported in part by AFOSR under contract F49620-85-C-001 with RGB Associates, Inc. In addition, he was supported (through RGB Associates, Inc.) by the Innovative Science & Technology Office of the Strategic Defense Initiative Organization, administered through the Office of Naval Research under contract N00014-85-K-0479.

References

1. C. D. Thompson, "A Complexity Theory for VLSI," Ph.D. Thesis, Carnegie-Mellon U., Pittsburgh, PA, 1981).
2. J. D. Ullman, *Computational Aspects of VLSI* (Computer Science Press, Rockville, MD, 1984). Chap. 3.
3. L. Snyder, "Supercomputers and VLSI: The Effect of Large Scale Integration in Computer Architecture," in *Advances in Computers, Vol. 23*, M. C. Yovits, Ed. (Academic, Orlando, FL, 1984). pp. 1–33.
4. D. F. Barbe, Ed., *Very Large Scale Integration (VLSI): Fundamentals and Applications* (Springer-Verlag, New York, 1980).
5. N. G. Einspruch, Ed., "VLSI Electronics," in *Microstructure Science, Vols. 1–7* (Academic, Orlando, FL, 1980–1985).
6. A. L. Rosenberg, "Three-Dimensional Integrated Circuits," in *VLSI Systems and Computation*, H. Kung, R. Sproul, and G. Steel, Eds. (Computer Science Press, Rockville, MD, 1981), pp. 69–79.
7. F. Leighton and A. Rosenberg, "Three-Dimensional Circuit Layouts," unpublished memorandum (MIT, Cambridge, 1982).
8. J. Caulfield, J. Neff, and W. Rhodes, "Optical Computing: The Coming Revolution in Optical Signal Processing," Laser Focus/Electro-Optics 38, 100 (1983).
9. D. Gabor, "Light and Information," Prog. Opt. 1, 109 (1961).
10. D. Slepian, "Prolate Spheroidal Wave Functions, Fourier Analysis and Uncertainty—IV," Bell Syst. Tech. J. 44, 4009 (1964).
11. A. Walther, "Gabor's Theorem and Energy Transfer Through Lenses," J. Opt. Soc. Am. 57, 639 (1967).
12. G. Newsam and R. Barakat, "Essential Dimension as a Well-Defined Number of Degrees of Freedom of Finite-Convolution Operators Appearing in Optics," J. Opt. Soc. Am. A 2, 2040 (1985).

Polynomial convolution algorithm for matrix multiplication with application for optical computing

Richard Barakat and John Reif

First, we describe an algorithm (the polynomial convolution algorithm) for the multiplication of two rectangular matrices A and B. The algorithm codes the matrix elements of A and B into two polynomials in a common indeterminate; the degree of the polynomial characterizing A depends on the size of both A and B, while the degree of the polynomial characterizing B only involves the size of B. The matrix elements of the product C = AB are obtainable from the convolution of the two polynomials. Although the resultant analysis is quite complex, its implementation in optical computing can be carried out in straightforward fashion (see Sec. III). The algorithm is at least as fast as the outer product and Kronecker product algorithms advocated by Athale-Collins and Barakat, respectively, in the assumed conditions of equally accessible matrix elements. Second, we consider the situation where the matrices are so large that they cannot be stored simultaneously on optical masks. It is shown that the speed advantages of the outer product and Kronecker product algorithms are now lost in this situation, whereas the polynomial convolution algorithm, because of its modular structure, is robust with respect to the storage problem. Finally, we consider some partitioning strategies in the light of the storage problem.

I. Introduction

It is generally agreed that in the realm of computational linear algebra, particularly the multiplication of two matrices, optical computing has an inherent speed of execution advantage over digital electronics (but see Sec. IV). Investigators in optical computing have generally taken matrix multiplication algorithms directly from the mathematical literature and modified them for use in optical computing. Some representative papers are Refs. 1–4. Alternately optical architectures have been developed to carry out such computations, e.g., Refs. 5–11.

One purpose of the present paper is to describe our polynomial convolution algorithm which is an *ab initio* development of matrix multiplication for use in optical computing. A second purpose is to consider the situation where the matrices are so large that they cannot be stored simultaneously on optical masks (hereafter termed the storage problem). As we show in Sec. IV, the speed advantage of the methods advocated in Refs. 1–4 are compromised because the matrix elements are not equally accessible. Furthermore, we make plausible that the polynominal convolution algorithm is robust with respect to this debilitating situation in that it is still possible to obtain a reasonable concurrency over the more classical algorithms because of the simplified bookkeeping and modular structure of the convolution algorithm.

The authors are with Harvard University, Division of Applied Sciences, Harvard University, Cambridge, Massachusetts 02138.

II. Polynomial Convolution Algorithm

In view of the initial complexity of the algorithm we proceed in three stages. In the first stage we give the explicit expressions and verify these formulas in the second stage. Finally, we outline a construction which leads to the various formulas.

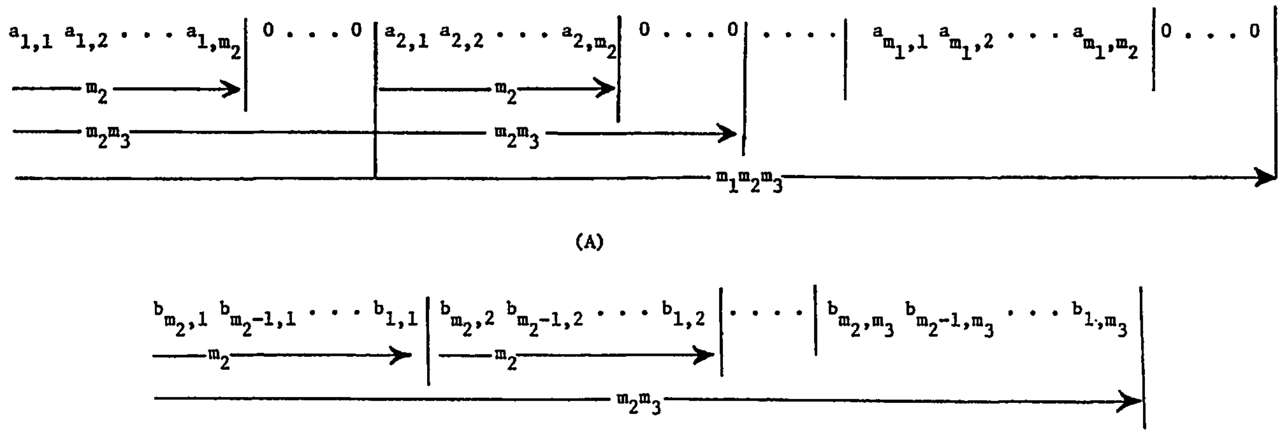
We begin by considering the matrix product C = AB, where A is of the size $n_1 \times n_2$, B is of size $n_2 \times n_3$, and C is of size $n_1 \times n_3$, with corresponding matrix elements a_{ij}, b_{jk}, and c_{ik}. Let x be an indeterminate and associate with A and B the polynomials $P(x)$ and $Q(x)$:

$$P(x) = \sum_{s=0}^{(n_1-1)n_2n_3+n_2-1} p_s x_s; \tag{1}$$

$$Q(x) = \sum_{t=0}^{n_2n_3-1} q_t x_t. \tag{2}$$

Note that the degree of $P(x)$ is $(n_1 - 1)n_2n_3 + n_2 - 1$, which involves not only the size of A through n_1 and n_2

Preprint of paper appearing in *Journal of Applied Optics*, Vol. 26, No. 14, July 15, 1987, pp. 2707-2711.



$a_{1,1}\ a_{1,2} \cdots a_{1,m_2}$ | $0 \cdots 0$ | $a_{2,1}\ a_{2,2} \cdots a_{2,m_2}$ | $0 \cdots 0$ | $\cdots$ | $a_{m_1,1}\ a_{m_1,2} \cdots a_{m_1,m_2}$ | $0 \cdots 0$
m_2
m_2m_3
$m_1m_2m_3$
(A)
$b_{m_2,1}\ b_{m_2-1,1} \cdots b_{1,1}$ | $b_{m_2,2}\ b_{m_2-1,2} \cdots b_{1,2}$ | $\cdots$ | $b_{m_2,m_3}\ b_{m_2-1,m_3} \cdots b_{1,m_3}$
m_2
m_2m_3
(B)

Fig. 1. Layout of the **p** vector (A) and **q** vector (B).

but also the size of **B** through n_3. The degree of $Q(x)$ is $n_2n_3 - 1$ and only involves the size of **B**, namely, n_2 and n_3. The p and q coefficients are related to the matrix elements of **A** and **B** by

$$p_s = a_{ij}, \quad \text{if } s = (i-1)n_2n_3 + j - 1 \tag{3a}$$
$$= 0, \quad \text{if } (i-1)n_2n_3 + n_2 \le s \le in_2n_3 \tag{3b}$$

and

$$q_t = b_{jk}, \quad \text{if } t = kn_2 - j \tag{4a}$$
$$= 0, \quad \text{if } t \le n_2n_3, \tag{4b}$$

with $1 \le i \le n_1$, $1 \le j \le n_2$, and $1 \le k \le n_3$.

We claim that the elements of the matrix product **C** are given by selected coefficients of the polynomial

$$R(x) = P(x)Q(x) = \sum_{m=0}^{n_1n_2n_3-1} r_m x^m, \tag{5}$$

where

$$r_m = \sum_{s=0}^{m} p_s q_{m-s} \tag{6}$$

is the discrete convolution of the p and q coefficients. These selected r_m are given by

$$r_m = c_{ik}, \quad \text{if } m = (i-1)n_2n_3 + kn_2 - 1. \tag{7}$$

A formal proof (which is really a verification of the formulas) is now given. We begin by rewriting Eq (6) in the form

$$r_m = \sum_s p_s q_{m-s} = \sum_{\alpha,\beta,\gamma,\delta} a_{ij} b_{jk}, \tag{8}$$

where the summation in the second series is over:

$$\alpha: \ s = (i-1)n_2n_3 + j - 1; \tag{9a}$$
$$\beta: \ (i-1)n_2n_3 < s < (i-1)n_2n_3 + n_2; \tag{9b}$$
$$\gamma: \ t = m - s = kn_2 - j; \tag{9c}$$
$$\delta: \ t < n_2n_3. \tag{9d}$$

The α term is simply Eq. (3a), while the β term is the negation of Eq. (3b). The γ term follows from Eq. (4a), while the δ term is the negation of Eq. (4b). On substitution of the α term into the β inequality we immediately see that this can only be true:

$$1 \le j \le n_2. \tag{10}$$

In like fashion, substitution of the γ term into the δ inequality leads to the requirement that

$$m = (i-1)n_2n_3 + kn_2 - 1, \tag{11}$$

which is Eq. (7). Thus the formulas are verified.

A construction which leads to the various formulas for p_s and q_t in terms of a_{ij} and b_{jk}, respectively, uses row vectors. Consider a row vector **p** whose elements we denote by p_s [coefficients of the polynomial $P(x)$] composed of the matrix elements a_{ij} of **A** and strings of zeros as depicted in Fig. 1(A). The range of s is

$$0 \le s \le n_1n_2n_3 - n_2n_3 + n_2 - 1. \tag{12}$$

Consequently

$$p_s \equiv 0, \quad \text{if } s \ge (n_1 - 1)n_2n_3 + n_2 \tag{13a}$$
$$\equiv 0, \quad \text{if } s \le in_2n_3. \tag{13b}$$

Furthermore the p_s is related to the a_{ij} as given by Eq. (3a), as the reader can verify by construction.

In like fashion, we construct another row vector **q** with elements q_t according to Fig. 1(B). Unlike **p**, **q** has no strings of zero elements. The range of t is

$$0 \le t \le n_2n_3 - 1, \tag{14}$$

so that

$$q_t \equiv 0, \quad \text{if } t \ge n_2n_3. \tag{15}$$

Within the range of t, the q_t is related to the b_{jk} by

$$q_t = b_{jk}, \quad \text{if } t = (k = 1)n_2 + n_2 - j, \tag{16}$$

which reduces to Eq. (4a).

As an illustrative example of the algorithm, consider the case where **A** is 2×2, **B** is 2×3 so that **C** is 2×3

(i.e., $n_1 = 2$, $n_2 = 2$, $n_3 = 3$). The upper limits on the polynomials P, Q, and R are 7, 5, and 11, respectively. The p_s, q_t, and r_m coefficients evaluated according to Eqs. (3), (4), and (7) are listed in Table I. On carrying out the convolution operation, Eq. (6), in conjunction with this table we have

$$r_1 = c_{11} = p_0q_1 + p_1q_0 = a_{11}b_{11} + a_{12}b_{21}, \tag{17a}$$

$$r_3 = c_{12} = p_0q_3 + p_1q_2 = a_{11}b_{12} + a_{12}b_{22}, \tag{17b}$$

$$r_5 = c_{13} = p_0q_5 + p_1q_4 = a_{11}b_{13} + a_{12}b_{23}, \tag{17c}$$

$$r_7 = c_{21} = p_6q_1 + p_7q_0 = a_{21}b_{11} + a_{22}b_{21}, \tag{17d}$$

$$r_9 = c_{22} = p_6q_3 + p_7q_2 = a_{21}b_{12} + a_{22}b_{22}; \tag{17e}$$

$$r_{11} = c_{23} = p_6q_5 + p_7q_4 = a_{21}b_{13} + a_{22}b_{23}. \tag{17f}$$

These are the matrix elements as obtained by more standard procedures.

This completes our description of the algorithm.

III. Implementation and Parallelism of Algorithm

In spite of the complicated looking nature of the algorithm, its implementation in optical computing can be carried out in straightforward fashion.

Examination of Fig. 1(A) shows that the matrix elements a_{ij} of **A** coded into the vector **p** consists of the rows of **A** in which strings of zeros are interspaced. Thus all we need to do to handle **A** in this algorithm is to store it on an optical mask according to Fig. 1(A). The vector **q** containing the matrix elements b_{jk} is simply the columns of **B** in reverse order [see Fig. 1(B)]. Obviously we need only code **B** as per Fig. 1(B) on an optical mask for this aspect of the implementation. Given that both these operations have been carried out, we proceed according to the various formulas quoted in the previous section.

The parallelism of the algorithm (assuming that all the matrix elements of **A** and **B** can be stored in primary storage) manifests itself through the corresponding **p** and **q** vectors. This is best seen by examination of Table I; the first two components of **p** (i.e., a_{11} and a_{12}) can then be combined simultaneously with (b_{21}, b_{11}), (b_{22}, b_{12}), and (b_{23}, b_{13}) of the vector **q**. While these operations are being carried out, the last two (nonzero) elements of **p** (i.e., a_{21} and a_{22}) are to be combined with (b_{21}, b_{11}), (b_{22}, b_{12}), (b_{23}, b_{13}). Thus we are able to carry out the manipulations leading to the six matrix elements of **C** simultaneously. The general case of two rectangular matrices does not require detailed comment. Consequently, the polynomial convolution algorithm is at least as fast as the methods advocated in Refs. 2 and 4 under the assumed conditions of equally accessible matrix elements.

IV. Influence of Storage Problem on Algorithm Parallelism in Matrix Multiplication

Although the issue of matrix multiplication, in the context of optical computing, has been cast as one of speed of execution of manipulations, this is only one aspect of the problem as we will now see. Realistic signal-processing requirements demand very large ma-

Table I. Listing of the p, q, and r Coefficients for the Case Where **A** is 2 × 2, **B** is 2 × 3, and **C** is 2 × 3.

	p_s	q_t	r_m
0	a_{11}	b_{21}	
1	a_{12}	b_{11}	c_{11}
2	0	b_{22}	
3	0	b_{12}	c_{12}
4	0	b_{23}	
5	0	b_{13}	c_{13}
6	a_{21}		
7	a_{22}		c_{21}
8	0		
9	0		c_{22}
10	0		
11	0		c_{23}
12	0		

trices to achieve the resolutions necessary to fulfill the desired goals. Because such large matrices are needed we must study the effect of storage (that is, the extent to which all matrix elements in the two matrices under multiplication are not equally accessible) on the inherent parallelism, and hence speed, of the various algorithms proposed.

When the matrices are small (for convenience we let them both be square and of size $n \times n$), the entire arrays containing the matrix elements of **A** and **B** can reside simultaneously in primary storage in the form of matrix masks as described in Goodman.[12] Then it is possible to carry out all the manipulations such as described in the algorithms promulgated in Refs. 1–4. Under the small n regime it is essentially true that all matrix elements are equally accessible. In fact, all the papers that we have succeeded in locating on matrix multiplication (via optical computing) tacitly make the assumption that all matrix elements are equally accessible, independent of n.

Let us consider, for example, the inner, intermediate, and outer product methods for the multiplication of matrices. Reference is made to the Appendix for development of an efficient formalism that yields these representations. Examination of these representations reveals that it is possible to perform the matrix–matrix product at two levels of parallelism. At the first level, the intermediate product methods speed up the execution over the inner product method by a factor of n. At the second level, the outer product method achieves a factor of n^2 over the inner product method. In fact, there are n parallel multiplications and $(n - 1)$ parallel additions to be performed rather than the n^3 sequential multiplications and $(n^3 - n^2)$ sequential additions required at the original element level algorithm. Unfortunately when n is large, the entire arrays cannot reside in primary storage but only portions thereof. This means that the speed advantage of the outer product method is now lost when computing large matrices, because the matrix elements are not equally accessible! A second tacit assumption is that all arithmetical operations of the same type are equivalent in both cost and accuracy. This too is violated when n is large.

Thus we cannot simply dismiss the use of the intermediate product representations when n is large. To improve the efficiency of the computation in this situation, it is necessary to maximize the use that is made of the matrix element data on a given matrix mask (containing parts of **A** or **B**) while it is in primary storage. It is probably advantageous to store matrix elements by columns. This is precisely what the column intermediate representation does: $\mathbf{Ce}_j$ is formed as a linear combination of $\mathbf{Ae}_k$ with a combination coefficient drawn from $\mathbf{Be}_j$. Obviously one can choose to stow rows so that the row intermediate representations are appropriate. In this scenario, we can only achieve a factor of n in the parallelism to accommodate the storage problem. There is also the bookkeeping question as to efficient storage and subsequent manipulation of the matrix elements in accordance with the particular algorithm requirements. See Hockney and Jesshope[13] for an overview of such considerations in digital electronic computers.

One possible solution for increasing parallelism when n is large is via partitioning. The idea is certainly not new as witness the recent paper by Caulfied *et al.*[3] who choose to use 2×2 matrices for the partitioning. Another viable approach using the formalism of the Appendix is the following. Suppose that **A**, **B**, and **C** are partitioned into submatrices. This means that the partitioning of the rows of **A** and those of **C** is the same, that the partitioning of the columns of **B** and those of **C** is the same, and that the partitioning of the columns of **A** and of the rows **B** is the same. The matrix product can then be formed blockwise. The foregoing remains valid if transcribed by replacing $\mathbf{e}_i$ by $\mathbf{E}_i$ etc. $\mathbf{E}_i$ is the ith block column of the appropriately partitioned identity matrix, the appropriate partitioning being that which is symmetric with respect to rows and columns for the matrix multiplication in question. Consequently, we recognize $\mathbf{AE}_j$ as the jth block column of **A**, $\mathbf{E}_i^+ \mathbf{A}$ as the ith block row of **A**, and $\mathbf{E}_i^+ \mathbf{AE}_j$ as the (i,j)th block element of **A**; thus we have

$$\mathbf{I} = \sum_k \mathbf{E}_k \mathbf{E}_k^+ . \tag{18}$$

It may be possible to store large matrices in partitioned form with the natural units to be stored and manipulated being the submatrices constituting the blocks.

What of the other approaches as influenced by the storage problem? The reduction to an equivalent matrix–vector problem advocated by Barakat[4] suffers the same fate as the outer product representation when n is large in that all the matrix elements are not equally accessible. Reference 4, see Eq. (1), shows that the Roth column decomposition of **AB** contains replicas of the matrix **A** along the principal diagonal, so that in this version all the matrix elements cannot be held in primary storage. Thus for large n, the parallelism inherent in the general reduction to the Roth column decomposition for matrix–vector multiplications is inhibited. However, there is also a Roth row decomposition of **AB** [see Eq. (4) of Ref. 4], in which the matrix elements of **A** are now spread along diagonals. It was hoped, in view of the previous work by Madsen *et al.*[14] on matrix multiplication by diagonals, that the storage problem could be circumvented. A detailed analysis which we need not reproduce indicates that the row decomposition is no more efficient than the column decomposition regarding the primary storage of matrix elements.

Finally we come to the algorithm of the present paper. The implementation of the algorithm as discussed in Sec. III bears directly on the storage problem. When the matrices are large enough to violate the equal accessibility condition, we can still maintain a reduced degree of parallelism because the convolution algorithm does not require the rather complicated bookkeeping that the column middle product decomposition necessitates before calculations can be carried out. Even though we cannot simultaneously store all the matrix elements of **A** and **B**, the convolution algorithm only requires the rows of **A** to be stored on separate optical masks so they can interact with the successive columns (in reverse order) of **B** and sequentially stand on optical masks to produce the various rows of **C**. Consequently when both **A** and **B** are large, we can still maintain a degree of parallelism because we do not require all the matrix elements of **A** and **B** to be in primary storage simultaneously. All we need in primary storage are the respective row and column of **A** and **B**. Thus the polynomial convolution algorithm seems to be more immune to the storage problem than do the algorithms in Refs. 2 and 4. This is because both the outer product and Kronecker product decomposition algorithms are not modular in structure; if the equal accessibility condition is violated there is no way to patch them up to work in the situation where the matrices are very large. It may be possible to employ partitioning as described in Ref. 3 or in the present paper; however, the bookkeeping is probably going to be a significant obstacle.

Richard Barakat was supported in part by AFOSR under contract F49620-85-C-001 with RGB Associates, Inc. In addition, he was supported (through RGB Associates, Inc.) by the Innovative Science and Technology Office for the Strategic Defense Initiative Organization and was administered through the Office of Naval Research under contracts N00014-85-K-0479 and N00014-86-K-0591.

Appendix

The purpose of this Appendix is to outline an efficient formalism (due to our colleague D. G. M. Anderson, unpublished) describing the inner product, intermediate product, and outer product representations of matrix multiplication. We further employ this formalism to discuss matrix partitioning, see Eq. (18).

To begin we avoid unnecessary complications by assuming that the two matrices, call them **A** and **B**, are square. It is also convenient to use the vector $\mathbf{e}_k$ which is the kth column of the unit matrix, i.e.,

$$\mathbf{I} = \sum_{k=1}^{n} \mathbf{e}_k \mathbf{e}_k^+ , \tag{A1}$$

and the plus sign denotes the transpose (thus $\mathbf{e}_+^k$ is a row vector). Given the square matrix **A**, we have

*j*th colum of $\mathbf{A} = \mathbf{A}\mathbf{e}_j$,

*i*th row of $\mathbf{A} = \mathbf{e}_i^+ \mathbf{A}$,

(i,j)th element of $\mathbf{A} = \mathbf{e}_i^+ \mathbf{A}\mathbf{e}_j$.

The usual element representation of the matrix product $\mathbf{C} = \mathbf{AB}$ reads in the above notation

$$\mathbf{e}_i^+\mathbf{C}\mathbf{e}_j = \sum_k (\mathbf{e}_i^+\mathbf{A}\mathbf{e}_k)(\mathbf{e}_k^+\mathbf{B}\mathbf{e}_j). \tag{A2}$$

The element representation is the old fashioned way that matrices were multiplied before high level programming languages were invented.

To obtain the inner product representation, we begin with the element representation, Eq. (A2), and delete the parenthesis on the right-hand; thus

$$\begin{aligned}\mathbf{e}_i^+\mathbf{C}\mathbf{e}_j &= \sum_k \mathbf{e}_i^+\mathbf{A}\mathbf{e}_k\mathbf{e}_k^+\mathbf{B}\mathbf{e}_j \\ &= (\mathbf{e}_i^+\mathbf{A})\left[\sum_k \mathbf{e}_k\mathbf{e}_k^+\right](\mathbf{B}\mathbf{e}_j) \\ &= (\mathbf{e}_i^+\mathbf{A})(\mathbf{B}\mathbf{e}_j).\end{aligned} \tag{A3}$$

The reason it is termed the inner product representation is that the matrices **A** and **B** are sandwiched between the unit vectors.

At the other extreme, we have the outer product representation which we obtain in the following fashion from the element representation, Eq. (A2):

$$\mathbf{e}_i^+\mathbf{C}\mathbf{e}_j = \sum_k \mathbf{e}_i^+\mathbf{A}\mathbf{e}_k\mathbf{B}\mathbf{e}_j = \mathbf{e}_i^+\left[\sum_k (\mathbf{A}\mathbf{e}_k)(\mathbf{e}_k^+\mathbf{B})\right]e_j. \tag{A4}$$

Consequently,

$$\mathbf{C} = \sum_k (\mathbf{A}\mathbf{e}_k)(e_k^+\mathbf{B}). \tag{A5}$$

The reason it is termed the outer product representation is that the matrices **A** and **B** now reside at the extreme left and right of the summation. This expression can be shown to be equivalent to the expression given in Athale and Collins[2], see their Eq. (2).

We next consider two intermediate representations which we term the column intermediate product representation and the row intermediate product representation. We return again to Eq. (A2):

$$\mathbf{e}_i^+\mathbf{C}\mathbf{e}_j = \sum_k \mathbf{e}_i^+\mathbf{A}\mathbf{e}_k\mathbf{e}_k^+\mathbf{B}\mathbf{e}_j = \mathbf{e}_i^+ \sum_k (\mathbf{A}\mathbf{e}_k)[\mathbf{e}_k^+(\mathbf{B}\mathbf{e}_j)], \tag{A6}$$

or

$$\mathbf{C}\mathbf{e}_j = \sum_k (\mathbf{A}\mathbf{e}_k)[\mathbf{e}_k^+(\mathbf{B}\mathbf{e}_j)]. \tag{A7}$$

This is the column intermediate product. The corresponding row intermediate product is

$$\mathbf{e}_i^+\mathbf{C}\mathbf{e}_j = \sum_k [(\mathbf{e}_i^+\mathbf{A})\mathbf{e}_k](\mathbf{e}_k^+\mathbf{B})\mathbf{e}_j \tag{A8}$$

or

$$\mathbf{e}_i^+\mathbf{C} = \sum_k [(\mathbf{e}_i^+\mathbf{A})\mathbf{e}_k](\mathbf{e}_k^+\mathbf{B}). \tag{A9}$$

It is a straightforward exercise to extend the above formalism to accommodate rectangular matrices; we omit the details.

References

1. D. Casasent and C. Neumann, "Iterative Optical Vector-Matrix Processors," in *Optical Information Processing for Aerospace Applications*, NASA Conf. Publ. 2207 (NTIS, Springfield, VA, 1981), p. 105.
2. R. A. Athale and W. C. Collins, "Optical Matrix–Matrix Multiplier Based on Outer Product Decomposition," Appl. Opt. 21, 2089 (1982).
3. H. Caulfield, C. Verber, and R. Stermer, "Efficient Matrix Partitioning for Optical Computing," Opt. Commun. 51, 213 (1984).
4. R. Barakat, "Optical Matrix–Matrix Multiplier Based on Kronecker Product Decomposition," Appl. Opt. 26, 191 (1987).
5. A. R. Dias, "Incoherent Optical Matrix–Matrix Multiplier," in *Optical Information Processing for Aerospace Applications*, NASA Conf. Publ. 2207 (NTIS, Springfield, VA, 1981), p. 71.
6. W. K. Cheng and H. Caulfield, "Fully-Parallel Relaxation Algebraic Operations for Optical Computers," Opt. Commun. 43, 251 (1982).
7. R. P. Bocker, H. J. Caulfield, and K. Bromley, "Rapid Unbiased Bipolar Incoherent Calculator Cube," Appl. Opt. 22, 804 (1983).
8. R. P. Bocker, "Advanced RUBIC Cube Processor," Appl. Opt. 22, 2401 (1983).
9. R. Bocker, K. Bromley, and S. Clayton, "A Digital Optical Architecture for Performing Matrix Algebra," Proc. Soc. Photo-Opt. Instrum. Eng. 431, 194 (1983).
10. H. Nakano and K. Hotate, "Optical System for Real-Time Processing of Multiple Matrix Product," Electron. Lett. 21, 436 (1985).
11. S. Cartwright and S. Gustafson, "Convolver-Based Optical Systolic Processing Architectures," Opt. Eng. 24, 59 (1985).
12. J. Goodman, "Architectural Development of Optical Data Processing Systems," Kinam 5C, 9 (1983).
13. R. W. Hockney and C. R. Jesshope, *Parallel Computers: Architecture, Programming and Algorithms* (Adam Hilger, Bristol, 1981), pp. 276–279.
14. N. Madsen, G. Rodrigue, and H. Karush, "Matrix Multiplication by Diagonals on a Vector/Parallel Processor," Inf. Process. Lett. 5, 41 (1976).

Efficient parallel algorithms for optical computing with the discrete Fourier transform (DFT) primitive

John H. Reif and Akhilesh Tyagi

Optical-computing technology offers new challenges to algorithm designers since it can perform an n-point discrete Fourier transform (DFT) computation in only unit time. Note that the DFT is a nontrivial computation in the parallel random-access machine model, a model of computing commonly used by parallel-algorithm designers. We develop two new models, the DFT–VLSIO (very-large-scale integrated optics) and the DFT–circuit, to capture this characteristic of optical computing. We also provide two paradigms for developing parallel algorithms in these models. Efficient parallel algorithms for many problems, including polynomial and matrix computations, sorting, and string matching, are presented. The sorting and string-matching algorithms are particularly noteworthy. Almost all these algorithms are within a polylog factor of the optical-computing (VLSIO) lower bounds derived by Barakat and Reif [Appl. Opt. **26**, 1015 (1987) and by Tyagi and Reif [*Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing* (Institute of Electrical and Electronics Engineers, New York, 1990) p. 14]. © 1997 Optical Society of America

Key words: Algorithms, discrete Fourier transform, optical computing, very-large-scale integrated optics model.

1. Introduction

During the past 20 years, VLSI has moved from theoretical abstraction to practical reality. As VLSI design tools and VLSI fabrication facilities such as the Metal-Oxide Semiconductor Implementation Service (MOSIS) became widely available, algorithm-design paradigms, such as systolic algorithms,[1] that were thought to be of theoretical interest only have been used in high-performance VLSI hardware. Along the same lines, the theoretical limitations of VLSI predicted by area–time trade-off lower bounds[2] have been found to be important limitations in practice. The field of electro-optical computing is in its infancy, comparable with the state of VLSI technology, say, 10 years ago. Fabrication facilities are not widely available—instead, the crucial electro-optical devices must be specially made in laboratories. However, a number of prototype electro-optical computing systems[3]—perhaps most notably at Bell Laboratories under Huang,[4,5]—as well as optical message-routing devices at the University of Colorado,[6] Boulder, Colo., Stanford University, Stanford, Calif., and the University of Southern California,[7–10] Los Angeles, Calif., have been built recently. The technology for electro-optical computing is likely to advance rapidly in the 1990's, just as VLSI technology advanced in the late 1970's and 1980's. Therefore, following our past experience with VLSI, it seems likely that the theoretical underpinnings for optical-computing technology—namely the discovery of efficient algorithms and of resource lower bounds—are crucial to guide its development.

What are the specific capabilities of optical computing that offer room for new paradigms in algorithm design? It is well known that optical devices exist that can compute a two-dimensional (2-D) Fourier transform or its inverse in unit time (see, for example, Goodman[11] or any of Refs. 12–18, which describe the fundamentals of optical computing). This is a natural characteristic of light. It would be reasonable to assume the existence of an optical-computing system with unit-time discrete Fourier transform (DFT) operation (for more details, see Subsections 2.A, 2.E, and 2.F). This assumption opens up exciting opportunities for algorithm designers.

In the widely accepted model of parallel computation—parallel random-access machine (PRAM)—not

J. H. Reif is with the Department of Computer Science, Duke University, Durham, North Carolina 27706. A. Tyagi is with the Department of Computer Science, Iowa State University, Ames, Iowa 50011.

many interesting problems can be solved in constant $O(1)$ time. In particular, the best-known parallel algorithm for the DFT, the fast Fourier transform (FFT), takes time $O(\log n)$ for an n-point DFT. Given this powerful technology, the question we address in this paper is which problems can gainfully use the DFT computation *primitive*? It is not immediately clear how a problem apparently distant from the DFT, such as sorting, can be solved by use of several DFT applications. We identify two general techniques for algorithm design with a DFT operation that benefit a host of problems. First, we show a way to compute one-dimensional (1-D) n-point DFT's efficiently using a series of 2-D DFT's. Note that the optical devices compute a 2-D DFT. However, the 1-D DFT seems to be the one that is more naturally usable in most problems. Second, we demonstrate an efficient way to perform a parallel-prefix computation with DFT primitives. Given n input values $x_0, x_1, \ldots, x_{n-1}$ and an associative operation denoted by the symbol $\circ$, the parallel-prefix problem is to compute all the prefix values $x_0, x_0 \circ x_1, x_0 \circ x_1 \circ x_2, \ldots, x_0 \circ x_1 \circ x_2 \circ \ldots \circ x_{n-1}$ in parallel. The parallel-prefix computation problem is important since it is used as an algorithm-design paradigm in the parallel-computing community. Equipped with these two techniques, we propose constant or near-constant time solutions for a variety of problems including sorting, matrix computations, and string matching.

Our results make a compelling argument in favor of supporting several DFT computation units in a digital optical-computing architecture. The algorithm designer and the optical-computing-architecture computing communities should identify other optical-computing primitives that result in efficient parallel algorithms.

Here we consider discrete models for optical computing with a DFT primitive. In particular, an n-point DFT operation or its inverse can be computed in unit time by use of n processors. The development of a new model of computation is a task full of trade-offs. Only the essential characteristics of the underlying computing medium should be reflected in the model. Any unnecessary characteristics serve only to undermine the usefulness of such a model. The PRAM[19,20] has provided a much needed model for the development of parallel algorithms for some time now. The existence of such a model frees up the algorithm designers from worrying about underlying networks and the details of timing inherent in the VLSI technology used to implement the processors. In a similar vein, our objective is to develop a model that captures the essence of the optical-computing medium with respect to algorithm design. We believe that the most important characteristic that distinguishes optical technology from VLSI technology is the ability to compute a function of many spatial points in unit time captured by a powerful primitive, the DFT. Not surprisingly then, this is the focus of our models. Our new models are the

- DFT–circuit model: Here we permit an n-point DFT primitive gate along with the usual scalar operations of bounded fan-in.
- DFT–VLSIO Model: Here we extend the standard VLSI model to three-dimensional (3-D) optical-computing devices that compute the 2-D DFT as a primitive operation. We refer to an electro-optical computation as VLSIO, where the uppercase letter O stands for *optics* (described in detail in Section 2, below).

Note that, although we did not mention a PRAM–DFT model, in which a set of n processors can perform a DFT in unit time, all the algorithms in the DFT–circuit model work for such a PRAM–DFT model. A PRAM–DFT model can simulate a DFT–circuit of size $s(n)$ and time $t(n)$ with $s(n)$ processors in a time of $O(t(n))$. Hence, a PRAM–DFT model is an equally acceptable choice for the development of parallel algorithms in optical computing.

Our main results are efficient parallel algorithms for solving a number of fundamental problems in these models. The problems solved include

1. The prefix sum.
2. Shifting.
3. Polynomial multiplication and division.
4. Matrix multiplication, inversion, and transitive closure.
5. Toeplitz-matrix multiplication, polynomial GCD, interpolation, and inversion.
6. Sorting.
7. One-dimensional and 2-D string matching.

The sorting and string-matching algorithms were *not* at all obvious. Although we do not have any lower bounds in the DFT–circuit model, many of these parallel algorithms are optimal with respect to the VLSIO model. The known lower-bound results for VLSIO are as follows: Barakat and Reif[21] showed a lower bound of $\Omega(I_f^{3/2})$ on $VT^{3/2}$ of a VLSIO computation for a function f with information complexity I_f, where V denotes the volume of the VLSIO system computing f. We[22] proved a lower bound of $\Omega[I_f f(\sqrt{I_f})]$ on the energy–time product for a VLSIO model with the energy function $f(x)$. Table 1 compares our results with the best-known PRAM algorithms for the corresponding problems. All the bounds are in uppercase letter O notation (O), also known as big-O notation.

A summary of related studies follows. VLSIO (electro-optical VLSI, introduced in Barakat and Reif[21]) is the more general model of optical computing (described in Subsection 2.B). They considered volume–time trade-offs and lower bounds in this model. We[22] demonstrated energy and energy–time product lower and upper bounds for optical computations. We are not aware of any algorithm design investigation in this model. Karasik and Sharir[30] proposed an enhancement of our model whereby the architecture supports several unit-time primitives (in addition to the DFT). MacKenzie and Ramachand-

Table 1. Comparison of the DFT–Circuit and DFT–VLSIO Algorithms with PRAM Algorithms

	DFT–Circuit		DFT–VLSIO		PRAM–CRCW	
Algorithm	Size	Time	Volume	Time	Number of Processors	Time
1-D DFT	n	1	$n^{3/2}$	1	n	$\log n$
Poly multiplication	n	1	$n^{3/2}$	1	n	$\log n$
Barrel shift	n	1	$n^{3/2}$	1	n	$\log n$
Prefix sum	n	1	$n^{3/2}$	1	n	$\log n$
Poly division[a]	n^2	1			n	$\log n$
Toeplitz-matrix multiplication[b]	n^2	1	$n^{5/2}$	1	n^2	$\log n$
Inverse and poly GCD and interpolation[b]	n^2	$\log n$	$n^{5/2}$	$\log n$	n^2	$\log n$
Matrix multiplication	n^3	1	$n^{7/2}$	1	n^3	$\log n$
Matrix-inversion transitive closure[c]	n^3	$\log n$	n^5	$\log n$	$n^{2.376}$	$\log^2 n$
Sorting[d]	n^2	1	$n^{5/2}$	1	n	$\log n$
Or	Randomized		Randomized			
	$n^{3/2}$	1	$\frac{n^{9/4}}{(\log n)^{5/2}}$	1	n	$\log n$
Or			Randomized			
	$\frac{n^{3/2}}{\log n}$	$\log \log n$	$\frac{n^{9/4}}{(\log n)^{5/2}}$	1	n	$\log n$
Element distinctness	Same as sorting	Same as sorting	Same as sorting	Same as sorting	Same as sorting	Same as sorting
1-D string matching[e]	n	1	$n^{3/2}$	1	$\frac{n}{\log \log n}$	$\log \log n$
2-D string matching	n	1	$n^{3/2}$	1	$\frac{n}{\log \log n}$	$\log \log n$

[a]PRAM–concurrent read concurrent write (CRCW) values as reported in Ref. 23.
[b]PRAM–CRCW values as reported in Refs. 24 and 25.
[c]PRAM–CRCW values as reported in Ref. 26.
[d]PRAM–CRCW values as reported in Refs. 27 and 28.
[e]PRAM–CRCW values as reported in Ref. 29.

ran[31] have studied an exclusive-read, concurrent-write parallel random-access memory model motivated by the capabilities of a dynamically reconfigurable optical network.

This paper is organized as follows. In Section 2, we introduce the two models of computation—the VLSIO and DFT–circuit. We describe briefly the salient architectural characteristics of optical computing in this section. The existence of a unit-time DFT primitive is also justified. In Section 3, we describe the algorithms for a set of direct applications of the DFT. In Section 4, we describe two sorting algorithms and an algorithm for the element-distinctness problem in these models. In Section 5, we give both 1-D and 2-D string-matching algorithms. In Section 6, we compare the performance of DFT–VLSIO algorithms with the known VLSIO lower bounds. In Section 7, we describe a generalization of these models in which the model is parametrized by the displacement rank d.

2. Discrete Fourier Transform–VLSI Optics and Discrete Fourier Transform–Circuit Models

A. Power of Optical–Electronic Processing

The primitives supported by an optical–electronic computer are (i) all the usual operations done by conventional electronic components, including (a) logical operations and (b) fast communication along wires by use of a small number of layers in a 2-D substrate, and (ii) the operations done by conventional 3-D optics, in particular the ability to do image convolutions and 2-D DFT's in constant time.

Optical processing provides new architectural paradigms as a result of the inherent parallelism in both computation and communication. McAulay[18] and Feitelson[14] provide a good introduction to the capabilities and limitations of optical computing. Optical communication, whether in free space or in optical fibers, allows light beams to pass through one another without distorting the information they carry. Similarly, optical lenses provide unit-time parallel linear computations, such as Fourier transforms and convolutions. Spatial light modulators and spatial light rebroadcasters are relatively recent components used in optical computing. Spatial light modulators are devices for interfacing both ways between optics and electronics; spatial light rebroadcasters trap energy from the light and provide a mechanism for performing arithmetic.

Conventional computer architectures suffer from the limited, finite fan-in of electronic devices and limited memory and communication bandwidth. The optical features discussed above help with computation and communication bandwidth. Holograms are a mechanism for providing a high-density and high-bandwidth memory as well.

Optics has some disadvantages, as well, *vis a vis* electronics. For instance, the degree of resolution when discretizing an analog amplitude in optics does not compare favorably with the almost-infinite analog-to-digital resolution in electronics. Both electronics and optics have their strengths and weaknesses. This combination of electronics and optics for exploiting the strengths of the two technologies is what makes the nature of computation so different with the new computing medium.

In summary, the key distinction of the physics of optical computation over conventional electronic components is the ability to perform communication across three dimensions and in free space by means of optical components. In contrast, conventional electronic components generally consist of only a small number of layers in a 2-D substrate. An excellent example is the use of conventional 3-D optics to compute a 2-D convolution in free space. From an architectural–implementation standpoint, we have the freedom to do fast parallel communications across a chip without having to deal with 2-D interconnect constraints.

B. VLSI Model

It has been observed many times that conventional electronic devices are inherently constrained by 2-D limitations. Indeed, this was the original motivation for the VLSI model developed by Thompson[32] that has been applied successfully to modeling such circuits. The widely accepted VLSI model allowed us both to compare the properties of algorithms, such as area and time, and to determine the ultimate limitations of such devices.

Let us first summarize the 2-D VLSI model, which is essentially the same as the one described by Thompson.[2] A computation is abstracted as a communication graph. A communication graph is very much like a flow graph, with the primitives being some basic operators that are realizable as electrical devices. Two communicating nodes are adjacent in this graph. A layout can be viewed as the convex embedding of the communication graph in a Cartesian grid. Each grid point can have either a processor or a wire passing through. A wire cannot go through a grid point with a processor unless it is a terminal of the processor at that grid point. The number of layers is limited to some constant γ. Thus both the fan-in and fan-out are bounded by 4γ. Wires have unit width and bandwidth, and processors have unit area. The initial data values are localized to some constant area to preclude encoding of the results. The input words are read at the designated nodes, called input ports. The input and subsequent computation are synchronous, and each input bit is available only once. The input and output conventions are WHERE–DETERMINATE, i.e., the locations of all the input–output ports are fixed in advance but need not be WHEN–DETERMINATE, i.e., the times when certain input or output bits become valid can depend on the input value.

C. VLSI Optics Model

The recent development of high-speed electro-optical computing devices[33,34] allows us to overcome the 2-D limitations of traditional VLSI. In particular, the optical-computing devices allow computation to be done in three dimensions, with full resolution in all the dimensions.

A rather different model for 3-D electro-optical computation is described in Ref. 21, which combines the use of optics and electronics components in ways that model currently feasible devices. This model is known as the VLSIO model, with the letter O standing for optics. In this model the fundamental building block is the optical box, consisting of a rectilinear parallelepiped whose surface consists of electronic devices modeled by the 2-D VLSI model and whose interior consists of optical devices. Communication from the surface is assumed to be carried out by means of electrical–optical transducers on the surface. Given specified inputs on the surface of the optical box, it is assumed that the output to the surface is produced in 1 time unit. Note that we do not rule out the possibility of two wide optical beams crossing while still transmitting distinct information. However, there is an assumption (justified by a theorem developed by Gabor[35]) that a beam of cross section A can transmit at most $O(A)$ bits per unit time. This is the only assumption made about the power of the optical boxes.

For the purposes of determining upper bounds we would have to be more specific about the computational power of the optical boxes. The use of electro-optical devices certainly will allow us to overcome the 2-D limitations. VLSIO potentially has more advantages over 2-D VLSI than just the 3-D interconnections of 3-D VLSI.[36,37] In particular, it is well known that a 2-D Fourier transform or its inverse can be computed by an optical device in unit time. In our discrete model we assume that an optical box of size $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$ with an input image of size $\sqrt{n} \times \sqrt{n}$ can compute its 2-D DFT in unit time. We call this the DFT–VLSIO model.

This assumption is consistent with the capabilities of the electro-optical components constructed in practice. In this case, the VLSIO model is clearly more powerful than is the 3-D VLSI model, e.g., since with the VLSI model we cannot do a DFT in constant time. A VLSIO device consists of a convex volume with a packing of optical boxes whose interiors do not intersect but may be connected by wires between their surfaces. This allows for communication between two optical boxes. Note that the VLSIO model encompasses the 3-D VLSI model as a *subcase*: Moreover, it is the particular subcase in which each optical box is just a 2-D *surface*, with no volume.

A VLSIO circuit is an embedded communication graph with the nodes corresponding to optical boxes in a 3-D grid. The volume of a VLSIO circuit is the volume of the smallest convex box enclosing it.

Since the Gabor theorem[35] establishes a finite upper bound on the bandwidth of an optical beam, without any loss of generality we can assume that only binary values are used for transmitting information.

D. Discrete Fourier Transform–Circuit Model

Let R be an ordered ring. A circuit over R consists of an acyclic graph with a distinguished set of input nodes and labeling of all the noninput nodes with a ring operation. In the DFT–circuit model, we allow

1. Scalar operations such as multiplication, addition, and comparison with two inputs.
2. DFT gates with n inputs and n outputs.

Note that all the gate-level operations are limited to occurring over an ordered ring R. This is not truly a limitation since in electronic computing the number of input bits to a gate is already fixed in the implementation (the data-bus width, usually 32 bits or 64 bits). This already limits the gate operations to occurring over a *de facto* ring.

The only optical gate in this model is the DFT gate. A spherical lens generates the Fourier transform of an image in analog form. A DFT gate can be implemented with such a lens-based optical device. The resulting analog image needs to be discretized in both space and amplitude (or some other parameter such as phase) to interpret it as an n-point DFT. However, there are limits to the dynamic range of optical systems, which is the approximate number of distinguishable intensity levels. Feitelson[14] reports the typical dynamic range to be a few hundred (of the order of 8 bits of resolution) for the current technology. Note that this limitation already constrains each DFT gate to computing over an ordered ring. In Subsections 2.E and 2.F, we describe how this dynamic-range limitation affects our unit-time DFT assumption.

The *size* of the DFT circuit is the sum of the number of edges and the number of nodes. Recall from Parberry and Schnitger[38] that a *threshold* circuit is a Boolean circuit of unbounded fan-in, where each gate computes the threshold operation. Threshold circuits are shown in Reif and Tate[23] to compute a large number of algebraic problems, such as polynomial division, triangular Toeplitz inversion, integer division, sine, cosine, etc., in an $n^{O(1)}$ size and simultaneous $O(1)$ depth. Many of these algorithms translate into DFT–circuit algorithms.

Let a DFT gate have n input bits $x_0, x_1, \ldots, x_{n-1}$. Note that the least significant bit of the output is $\sum_{i=0}^{n-1} x_i$. This observation, along with the availability of comparison gates, shows that a DFT circuit is at least as powerful as a threshold circuit of the same size and depth. The question we address in this paper is the power of the DFT operation above and beyond its power to compute the threshold. Note that no nontrivial lower bounds on a threshold circuit computing a DFT are known. But just by its definition at least n threshold gates are required for a DFT computation.

E. Chinese Remaindering

Here we explain how we can use the well-known Chinese-remaindering techniques for encoding large numbers so as to provide an equivalence between the discrete and analog optical models. Recall that the dynamic range of a typical optical-computing system is limited to be of the order of 8 bits (Ref. 14, p. 40). Hence we can reasonably well expect a unit-time DFT primitive for an $\approx$256-element ring. However, if we wish to achieve higher accuracy of our DFT results, we can use the following technique.

We assume that we wish to work with b-bit numbers for a large b, but the optical system processes in analog form and permits only a very low accuracy (say only a logarithmic number, $\log b$, of bits of accuracy, where $\log b \leq 256$ in the current technology). Fix relatively prime (i.e., with no common factors) numbers $p_1, \ldots, p_k$ such that their product is $\Pi_{i=1}^{k} p_i > 2^b$. By the prime-number theorem, the density of b-bit primes among all b-bit numbers is $O(1/b)$, so we can find $k \leq b/\log b$ such numbers $p_1, \ldots, p_k$, such that each number p_i has $\leq \log b + O(\log \log b)$ bits.

Given a large integer x of b bits, we can represent x by a sequence of numbers $x_1, \ldots, x_k$, where for each $i = 1, \ldots, k$ we have $x_i \equiv x \bmod p_i$. This is the Chinese-remaindered representation of x. The Chinese-remainder theorem states that, given the Chinese-remaindered representation $x_1, \ldots, x_k$, we can construct x uniquely: that is, there is a unique x such that, for each $i = 1, \ldots, k$, we have a value $x_i \equiv x \bmod p_i$.

To apply the Chinese-remainder algorithm, we need quickly to (i) compute a Chinese-remaindered representation of a given b-bit number x, and (ii) reconstruct x from this representation. Both these problems have obvious $O(b^2)$ work algorithms that can be executed in parallel in $O(\log b)$ time by circuits. Moenck and Borodin[39] developed some of the first subquadratic work algorithms (which are quite simple and practical) for these modular arithmetic problems, and later $O(b \log^2 b)$ algorithms were discovered (e.g., see the texts by Borodin and Munro[40] and Bini and Pan[41]).

Thus we can use these well-known Chinese-remaindering techniques for encoding large numbers into a list of small numbers that are used in optical processing (for example, in the optical DFT transform used in this paper). This is not a new idea and has been well known to the optical-processing community.[18] Given a b-bit number x, we compute a Chinese-remaindered representation of x. We can then send the representation over an optical channel, which requires very low accuracy, in analog form by use of $\leq \log b + O(\log \log b)$ bits. Then on reception we can convert the representation to digital form, and we can easily reconstruct x by these fast algorithms.

F. Discrete Fourier Transform by Means of Chinese Remaindering

Next we explain how we can use the Chinese-remaindering techniques described above to compute the DFT with high accuracy (b bits), assuming the optical system computes the DFT only with low, say logarithmic, accuracy. We use the fact that the DFT is a linear operator. We take as input a vector $Y = (y_0, \ldots, y_{n-1})$, which we assume contains b-bit numbers. We wish to compute the DFT of this vector Y with high accuracy, up to $b' = O(b \log n)$ bits, giving an output of $U = (u_0, \ldots, u_{n-1})$ [that is, each output u_j is to be approximated by a $b' = O(b \log n)$-bit number]. Note that each output u_i is a linear combination of the n inputs, with coefficients that can be approximated by fixed $O(b \log n)$-bit numbers.

Fix in this case relatively prime numbers, $p_1, \ldots, p_k$, such that each of the p_i has $\leq \log b' + O(\log \log b')$ bits and $\Pi_{i=1}^{k} p_i > 2^b$. We represent each number y_i by a sequence of numbers $y_{i,1}, \ldots, y_{i,k}$, which are the Chinese-remaindered representation of the y_i modulo of the primes $p_1, \ldots, p_k$, respectively. We can compute this representation in one step with several modular arithmetic gates. Then, for each $j = 1, \ldots, k$, we use the optical system to generate an approximate DFT (up to a logarithmic number of bits) for each vector $(y_{0,j}, \ldots, y_{n-1,j})$, yielding an approximate DFT vector $(a_{0,j}, \ldots, a_{n-1,j})$. This step also takes unit time. Then we round each approximate value $a_{i,j}$ to an integer $I_{i,j}$, where $0 \leq I_{i,j} < p_j$. This rounding can also be done in one step by use of modular arithmetic gates. Finally, we apply the Chinese-remaindering theorem to construct, for each $i = 0, \ldots, n-1$, the b'-bit number u_i, $0 \leq u_i < \Pi_{j=1}^{k} p_j$, from the integers $I_{i,1}, \ldots, I_{i,k}$, where $I_{i,j} = u_i \bmod p_j$. This takes work $O(b' \log^2 b')$ for each of the n output points, which can be done in parallel for each point.

This $O(b' \log^2 b')$ work for each of the n output points can be done with the classical circuit model in $O(\log^2 b')$ steps, but it then requires $O(b'^2)$ VLSI area per output point. Alternatively, it can be done in $O(b' \log^2 b')$ steps, with $O(b' \log^2 b')$ VLSI area per output point. Note that the $(b' \log n)$-bit accurate DFT can be performed in $O(\log^2 b')$ time, which is dominated by the time for the Chinese-remaindered representation, to obtain ring-R representation. This provides our output $(u_0, \ldots, u_{n-1})$, which is the DFT of a vector $\mathbf{Y}$ with up to b'-bit accuracy.

Thus we have shown that Chinese-remainder encoding of large numbers provides an equivalence between our discrete and analog optical models. In particular, the available 8-bit accuracy of current optical systems can be leveraged as follows. Chinese remaindering allows us to obtain up to $\approx 2^8 = 256$ bits of accuracy (approximately $10^{7.5}$ levels of accuracy) within $O(\log^2 2^8) \approx 64$ additional steps. In general, a k-bit accuracy in the optical technology can be leveraged to a 2^k-bit accuracy with $\approx k^2$ additional steps. Note that this technique requires all the additions and multiplications to be performed modulo a fixed prime number. Also note that these prime numbers are not computed at run time. A set of prime numbers is built into the architecture for a desired level of accuracy. Again, this is not a new idea. The residue number system uses the Chinese-remaindering representation. McAulay[18] describes both the residue number system and the use of Chinese remaindering to extend the dynamic range of spatial light modulators.

Another important point to note is that the data that are processed optically need not always be converted back and forth between Chinese-remaindered and ring-R representations. For many applications we need to compute additions and multiplications, which are closed under these modular operations. In fact, a compiler could optimize code for these machines to cluster operations closed under modular arithmetic so that the need for conversion between representations is minimized. This could enable us to amortize the $\approx k^4$ cost of conversion over several modular operations to give a near-constant-time DFT primitive of nearly arbitrary precision.

3. Algorithms

We use the following scheme to describe the algorithms. For each problem, we state the problem, follow it with the DFT–circuit algorithm, and in turn follow that with the DFT–VLSIO algorithm.

Note that an optical device computes a 2-D DFT operation. However, in most applications we find it useful to employ a 1-D DFT. Hence, before we describe the other algorithms, let us consider the cost of computing a 1-D DFT using a 2-D DFT primitive.

A. Cost of Computing a One-Dimensional Discrete Fourier Transform

For DFT the input is a vector $\mathbf{x} = [x_0, x_1, \ldots, x_{n-1}]$ and the output is $\mathbf{y} = A\mathbf{x}$. A represents an $n \times n$ DFT matrix whose (i,j)th element is ω^{ij}, where ω is a principal nth root of unity.

The following algorithm, a variant of that of Agarwal and Burrus,[42] uses a series of 2-D DFT operations to realize a 1-D DFT. We assume a commutative ring R with a principal nth root of unity ω such that $x_i \in R$. Without a loss of generality, let us also assume that $\sqrt{n}$ is a power of 2. Let us define the following:

$$\langle i,j \rangle = i\sqrt{n} + j,$$
$$\langle i, - \rangle = (i\sqrt{n}, i\sqrt{n} + 1, \ldots, 2i\sqrt{n} - 1),$$
$$\langle -,j \rangle = (j, j + \sqrt{n}, \ldots, j + n - \sqrt{n}).$$

Let $\bar{A}^{(\sqrt{n})}$ be the $\sqrt{n} \times \sqrt{n}$ circulant matrix such that $\bar{A}_{ij}^{(\sqrt{n})} = \omega^{\sqrt{n}\,ij}$.

1. *Algorithm 1*

1. For $j = 0, \ldots, \sqrt{n} - 1$ in parallel, DO $y_{\langle j,- \rangle} = \bar{A}^{(\sqrt{n})} x_{\langle -,j \rangle}$.
2. For $j = 0, \ldots, \sqrt{n} - 1$ and $v = 0, \ldots, \sqrt{n} - 1$ in parallel, DO $z_{\langle j,v \rangle} = y_{\langle j,v \rangle}\omega^{jv}$.
3. For $v = 0, \ldots, \sqrt{n} - 1$ in parallel, DO $f_{\langle -,v \rangle} = \bar{A}^{(\sqrt{n})} z_{\langle -,v \rangle}$.

The output is $\mathbf{f} = [f_0, f_1, \ldots, f_{n-1}]$.

2. Proof of Correctness

$$f_{\langle u,v\rangle} = \sum_{j=0}^{\sqrt{n}-1} z_{\langle j,v\rangle}\omega^{\sqrt{n}ju} \qquad \text{by step 3,}$$

$$= \sum_{j=0}^{\sqrt{n}-1} (y_{\langle j,v\rangle}\omega^{jv})\omega^{\sqrt{n}ju} \qquad \text{by step 2,}$$

$$= \sum_{i=0}^{\sqrt{n}-1}\sum_{j=0}^{\sqrt{n}-1} [(x_{\langle i,j\rangle}\omega^{\sqrt{n}iv})\omega^{jv}]\omega^{\sqrt{n}ju}$$

$$= \sum_{i=0}^{\sqrt{n}-1}\sum_{j=0}^{\sqrt{n}-1} x_{\langle i,j\rangle}\omega^{\sqrt{n}iv+jv+\sqrt{n}ju}$$

$$= \sum_{i=0}^{\sqrt{n}-1}\sum_{j=0}^{\sqrt{n}-1} x_{\langle i,j\rangle}\omega^{\langle i,j\rangle\cdot\langle u,v\rangle}.$$

Hence, for all $s = 0, \ldots, n-1$, $f_s = \sum_{k=0}^{n-1} x_k\omega^{ks}$. □

Let us consider the cost of this algorithm. Recall that in the DFT–VLSIO an n-point 2-D DFT takes a time of $O(1)$ and a volume of $n^{3/2}$. In algorithm 1, the first and third steps perform $\sqrt{n}$, $\sqrt{n}$-point 2-D DFT computations, hence they take a time of $O(1)$ and a volume of $n^{5/4}$. But the second step performs an n-point 2-D DFT, hence it takes a time of $O(1)$ and a volume of $n^{3/2}$. Thus, the total time and volume used by algorithm 1 are $O(1)$ and $O(n^{3/2})$, respectively. From now on we assume that the 1-D DFT is also available as a primitive operation in the DFT–VLSIO mode. The time and volume costs of performing an n-point, 1-D DFT are $O(1)$ and $n^{3/2}$, respectively. The term DFT refers to the 1-D DFT throughout the rest of this paper, unless specified otherwise.

Algorithm 1 also allows us to assume that DFT gates in the DFT–circuit model perform 1-D DFT's of n bits in unit time. The cost of this operation in the DFT–circuit model is $O(1)$ gates, $O(1)$ depth, and $O(n)$ size.

Let us present some polynomial algorithms. In the following $\mathrm{DFT}_n(\mathbf{x})$ refers to the n-point 1-D DFT of the vector $\mathbf{x} = [x_0, x_1, \ldots, x_{n-1}]$, where the values x_i come from the underlying ring R on which the DFT is defined. $\mathrm{DFT}_n{}^{-1}(\mathbf{y})$ refers to the n-point inverse 1-D DFT of the vector $\mathbf{y}$.

B. Polynomial Multiplication

1. Input

Two $(n-1)$th-degree polynomials with values of $p(x) = \sum_{i=0}^{n-1} a_ix^i$ and $q(x) = \sum_{j=0}^{n-1} b_jx^j$. Let $\mathbf{a} = [a_0, a_1, \ldots, a_{n-1}]$ be the vector of the coefficients of $p(x)$. Similarly, $\mathbf{b}$ is the vector of the coefficients $q(x)$.

2. Output

The product of $p(x)$ and $q(x)$, a $(2n-2)$th-degree, polynomial with a value of $p(x)q(x) = \sum_{i=0}^{2n-2} (\sum_{j=0}^{i} a_jb_{i-j})x^i = \sum_{i=0}^{2n-2} c_ix^i$. Let $\mathbf{c}$ be the vector of the coefficients from $p(x)q(x)$.

3. Algorithm

1. Compute $\underline{\mathbf{a}} = \mathrm{DFT}_{2n-1}(\mathbf{a})$ and $\underline{\mathbf{b}} = \mathrm{DFT}_{2n-1}(\mathbf{b})$.
2. $\underline{\mathbf{c}} = \underline{\mathbf{a}} \cdot \underline{\mathbf{b}}$, i.e., the dot product of two vectors.
3. $\mathbf{c} = \mathrm{DFT}_{2n-1}{}^{-1}(\underline{\mathbf{c}})$. Note that $\mathbf{c}$ contains the coefficients of $p(x)q(x)$.

4. Analysis

DFT–circuit Model: The $(2n-1)$-point DFT's in step 1 take $(2n-1)$-size gates and $O(1)$ time. The dot product of the two DFT vectors requires unit time with $(2n-1)$ two-input scalar gates. The dot product can, therefore, be performed in unit time with a $(4n-2)$ size. The inverse DFT in step 3 can also be done in unit time with a size of $(2n-1)$. The whole process takes $(8n-4)$ size and $O(1)$ time.

DFT–VLSIO Model: Steps 1 and 3 can be implemented directly in the DFT–VLSIO model with an optical box, taking a time of $O(1)$ and volume of $O(n^{3/2})$. However, the principal difficulty comes with the implementation of several integer multiplications in step 2. The DFT–circuit model has an advantage with respect to this step, as scalar gates performing multiplication in unit time are available. To multiply $(2n-1)$ coefficients of $\log n$ bits each, we can use a Wallace-tree-type multiplier realized in VLSI (Ref. 43, pp. A46–A49). Such a multiplier takes $O(\log^2 n)$ volume with a time of $O(\log\log n)$. For n such multiplications, the total volume is $O(n\log^2 n)$ with a time of $O(\log\log n)$.

However, we can do even better if we reduce the integer multiplication of two $(\log n)$-bit numbers to a polynomial multiplication. A modulo p ring, for an appropriate prime p, will have this property. A $(\log n)$-bit integer $A = a_{\log n-1}, \ldots, a_1, a_0$ can be considered as a polynomial $\sum_{i=0}^{\log n-1} a_ix^i$, with $x = 2$. To multiply the two $(\log n)$-bit integers A and B, multiply A and B as polynomials. For the polynomial multiplication, take their DFT's in a volume of $O(\log^{3/2} n)$ and a time of $O(1)$. The dot product of these $(2\log n - 1)$-bit DFT vectors can again be done recursively as a polynomial multiplication of two $(\log\log n)$-bit numbers. This recursive procedure for polynomial multiplication takes a time of $O(\log^* n)$, where $k = \log^* n$ if $\lfloor\log^{(k)} n\rfloor = 1$. Here, $\log^{(k)} n$ stands for k repeated applications of the logarithmic function, as in $\log\log\ldots\log\log n$. The resource requirement of this algorithm is also $O(n^{3/2})$ volume. To see this, consider the ith level of recursion for $i = 1, \ldots, \log^* n$. At this point, there are $n\log n\log\log n\ldots\log\log^{(i)} n$ instances of $(\log^{(i+1)} n)$-bit multiplications. The volume required for this step, then, is $n\log n\log\log n\ldots\log^{(i)} n[\log^{(i+1)} n]^{3/2}$. This is $O(n^{3/2})$. Most of this analysis is a very simplistic elaboration of the point. A more exact analysis builds recurrence equations for $T(n)$ and $V(n)$, the time and volume, respectively, for computing the product of two nth-degree polynomials:

$$T(n) = T[\log(2n-1)] + c,$$
$$T(1) = 1, \quad \text{for a constant } c,$$
$$V(n) = (2n-1)^{3/2} + (2n-1)V[\log(2n-1)],$$
$$V(1) = 1, \text{ with no resource reuse},$$
$$V(n) = \max\{(2n-1)^{3/2}, (2n-1)V[\log(2n-1)]\},$$
$$V(1) = 1, \text{ with resource reuse},$$
$$T(n) \le c\{\log^* n + \log^*(\log^* n) + \log^*[\log^*(\log^* n)] + \cdots + 1\}, \quad \text{at most } 2c \log^* n.$$

$V(n)$ also has an upper bound of $2(2n-1)^{3/2}$.

There is a third way to carry out the integer multiplication so as to perform polynomial multiplication in $O(1)$ time with $O(n^{3/2})$ volume. However, the hardware is used more inefficiently in this algorithm, which might argue for using one of the previous algorithms in practice. As we showed in Subsection 2.D, a DFT gate can perform thresholding in a trivial way. The first bit of the DFT vector is the sum of all the input bits. A comparison corresponds to an addition, which can also be performed with one DFT operation. Reif and Tate[23] show that integer multiplication can be done in a constant depth $[O(1/\epsilon^5)$, for any $\epsilon > 0]$ with threshold gates of a total size of $O(n^{2+2\epsilon})$. A threshold circuit of size $O(n^{2+2\epsilon})$ corresponds to a VLSIO circuit of volume $O(n^{3+3\epsilon})$. Hence, for performing integer multiplication of two n-bit integers in $O(1)$ time, $O(n^{3+\epsilon})$ volume is required for some $\epsilon > 0$.

The polynomial multiplication requires $(2n-1)\log n$-bit integer multiplications. These can be done in a time of $O(1)$ and a volume of $(2n-1)(\log n)^6$. Hence VLSIO polynomial multiplication is feasible in $O(1)$ time with $O(n^{3/2})$ volume. The constants in this algorithm have large magnitudes; hence, in practice one of the other algorithms with a higher asymptotic volume requirement is likely to be more efficient.

C. Barrel Shifting

1. Input

The inputs to this problem are a vector, $\mathbf{x} = [x_0, x_1, \ldots, x_{n-1}]$, and a shift value, $0 \le c \le n-1$.

2. Output

The output vector is cyclically shifted by $0 \le c \le n-1$: $\mathbf{y} = [y_0, y_1, \ldots, y_{n-1}]$, where $y_i = x_{(i-c)\bmod n}$.

3. Algorithm

One can reduce a cyclic shift to a right-hand shift by doubling the vector size, as described by Vuillemin.[44] Let vector $\mathbf{X}$ be the concatenation of $\mathbf{x}$ to itself: $\mathbf{xx}$. A right-hand shift by c on $\mathbf{X}$ is equivalent to multiplication of the polynomial corresponding to $\mathbf{X}$ by the polynomial x^c. We have already developed a polynomial-multiplication algorithm.

4. Analysis

DFT–circuit Model: The multiplication of two $2n$-degree polynomials can be done in a time of $O(1)$ and with $O(n)$ size. The duplication of $\mathbf{x}$ and selection of the left-hand half of the output also take a time of $O(1)$ and a size of $O(n)$, for a total cost of $O(1)$ time and $O(n)$ size.

DFT–VLSIO Model: Once again, the barrel-shift cost is the cost of input duplication, output selection, and polynomial multiplication. The input duplication and output selection take unit time and $O(n)$ volume, even in VLSI technology. Hence the polynomial-multiplication costs dominate. This leads to a total cost of *either* a volume of $O(n^{3/2})$ and a time of $O(\log^* n)$ *or* a volume of $O(n^{3/2})$ with a time of $O(1)$, depending on the integer-multiplication method employed.

D. Prefix Sum

1. Input

The input consists of $n+1$ elements $x_0, x_1, \ldots, x_n$.

2. Output

The output is all the prefix sums: $\sum_{j=0}^{i} x_j$, for all $0 \le i \le n$.

3. Algorithm

The prefix sum can be reduced to a polynomial multiplication. In particular, consider the multiplication of two polynomials: $\sum_{i=0}^{n} x_i y^i$ and $\sum_{j=0}^{n} y^j$. The multiplication of these two polynomials is $\sum_{i=0}^{2n} (\sum_{j=0}^{i} x_j) x^i$. Thus the ith coefficient of this product is the ith prefix sum for $0 \le i \le n$.

4. Analysis

DFT–circuit Model: The resource requirements correspond to those of polynomial multiplication. Hence this computation takes $O(n)$ size with $O(1)$ time.

DFT–VLSIO Model: Once again, this takes either a volume of $O(n^{3/2})$ with a time of $O(\log^* n)$ or $O(n^{3/2})$ volume with $O(1)$ time.

E. Polynomial Division

1. Input

The inputs to this problem are two n- and m-degree polynomials $p(x) = \sum_{i=0}^{n} a_i x^i$ and $q(x) = \sum_{i=0}^{m} b_i x^i$, respectively.

2. Output

The output is two polynomials $d(x)$ and $r(x)$, such that $p(x) = d(x)q(x) + r(x)$, where degree$[r(x)] <$ degree$[q(x)]$.

3. Algorithm

The polynomial division can be done with an n^2 size in a time of $O(1)$. In this case, the series $1/(1-\gamma) = \sum_{i=0} \gamma^i$ is used in the following way:

1. Compute the degree-m polynomial $q'(x) = 1 - q(x)$. This process requires negating the coefficients $b_1, \ldots, b_m$ and computing $1 - b_0$. Recall that the first coefficient of a DFT is the sum of all the inputs. Hence $(m + 1)$ DFT operations of constant degree lead to the polynomial $q'(x)$.

2. Get the reciprocal of $q(x)$ of degree $n - m$ (0 if $m \geq n$) by use of the expression $1/[q(x)] = 1/[1 - q'(x)] = \Sigma_{i=0}^{n-m} [q'(x)]^i$. To find the reciprocal of $q(x)$ to within the desired accuracy, we need to compute $[q'(x)]^i$ for $1 \leq i \leq (n - m)$. For computing $[q'(x)]^i$, let $\mathbf{q}' = [1 - b_0, -b_1, \ldots, -b_m]$. First, get $\mathrm{DFT}_{m+1}(\mathbf{q}') = [c_0', c_1', \ldots, c_m']$. In the Fourier domain, every component of $\mathrm{DFT}(\mathbf{q}')$ is raised to i to derive the DFT of $[q'(x)]^i$, i.e., $\mathrm{DFT}\{[q'(x)]^i\} = [c_0'^i, c_1'^i, \ldots, c_m'^i]$. These powers of the coefficients can be read from a lookup table. An inverse DFT then gives $[q'(x)]^i$. The $(n - m)$ inverse DFT's and polynomial additions lead to the value of $1/q(x)$ within degree $(n - m)$. Note that, since we need to compute $1/q(x)$ as a degree-$(n - m)$ polynomial, these DFT's and inverse DFT's are $(n - m)$-point transforms.

3. Multiply $1/q(x)$, derived above, and $p(x)$ to derive the output $d(x)$, a degree-$(n - m)$ polynomial.

4. Multiply $d(x)$ and $q(x)$ to get a degree-n polynomial $p'(x)$. Subtract $p'(x)$ from $p(x)$ to get $r(x)$.

Reif and Tate[23] give a general result in their corollary 3.3 that any function $f(x)$ with a convergent Taylor series expansion of the form $f(x) = \Sigma_{i=0}^{\infty} c_i(x - x_0)^i$ over an interval $|x - x_0| \leq \epsilon$, for $0 < \epsilon < 1$, with rational coefficients of magnitude of at most $2^{n^{O(1)}}$ can be computed with threshold circuits of size $n^{O(1)}$ in constant depth. The polynomial quotient and remainder problems fit this profile according to the discussion above, hence they can be computed with threshold circuits of size $n^{O(1)}$ in constant depth. This corresponds to a DFT–circuit size of $n^{O(1)}$ and time of $O(1)$, just what we derived. For the DFT–VLSIO, this translates into a volume of $n^{(3/2)O(1)}$ and a time of $O(1)$.

4. *Analysis*

DFT–circuit Model: The first step requires $m + 1$ subtraction gates and unit time. The most complicated part is getting the reciprocal $1/q(x)$. The DFT of $q'(x)$ takes a size of $m + 1$ and a time of $O(1)$. Lookup is a unit-time operation. The size requirements for inverse DFT's are $(n - m)\max[(m + 1), (n - m)]$ with a time $O(1)$. Addition of $[q'(x)]^i$ requires a size of $(n - m)\max[(m + 1), (n - m)]$ and time of $O(1)$. The third step is a plain polynomial multiplication with a size of $O(n)$ and time of $O(1)$. Step 4 is also a polynomial multiplication of degree-m and degree-$(n - m)$ polynomials; hence it takes a size of $O(n)$ with $O(1)$ time. The subtraction also has the same resource bounds. Hence the total resource requirements are a size of $O((n + m)^2)$ and a time of $O(1)$.

F. Toeplitz-Matrix Multiplication, Inverse and Polynomial Greatest Common Divisor and Interpolation

An $n \times n$ matrix M is a Toeplitz matrix if all the entries on the same diagonal are identical, i.e., $M[i, j] = M[i - k, j - k]$ for all $0 \leq i, j \leq n - 1$, and $\forall k$ such that $0 \leq i - k, j - k \leq n - 1$. Note that an upper- or lower-triangular Toeplitz matrix can be multiplied by a vector by use of a single convolution, which reduces it to the polynomial multiplication. For a lower- (upper-) triangular Toeplitz matrix, a convolution of the input vector with the bottom-most (topmost) row gives the result.

Let the bottom-most row of an $n \times n$ lower-triangular Toeplitz matrix M be $M[n - 1, i] = d_i$, for $0 \leq i \leq n - 1$. Let the input vector $\mathbf{x} = [x_0, x_1, \ldots, x_{n-1}]$. The vector $\mathbf{y} = M\mathbf{x}$ can be derived by computation of the polynomial $y(z) = \Sigma_{i=0}^{2n-1} y_i z^i = (\Sigma_{i=0}^{n-1} d_{n-i-1} z^i)(\Sigma_{i=0}^{n-1} x_i z^i)$. The first n coefficients of the polynomial $y(z)$ give the output vector $\mathbf{y} = [y_0, y_1, \ldots, y_{n-1}]$. This leads to a DFT–circuit cost of $O(n)$ size in constant time and a DFT–VLSIO cost of $O(n^{3/2})$ volume with $O(1)$ time.

Now consider the multiplication of two $n \times n$ lower-triangular Toeplitz matrices M and N. In this case, multiply the two polynomials $\Sigma_{i=0}^{n-1} M[n - 1, n - i - 1]x^i$ and $\Sigma_{i=0}^{n-1} N[0, i]x^i$. The coefficients of x^i for $0 \leq i \leq n - 1$ provide the $P[0, i]$ for $P = MN$. Since P is also a lower-triangular Toeplitz, the zeroth column provides the whole matrix. The multiplication of two upper-triangular Toeplitz matrices is identical. To multiply an $n \times n$ lower-triangular Toeplitz matrix M by an $n \times n$ upper-triangular Toeplitz matrix, we need to perform n such polynomial multiplications. This can be seen as n instances of matrix–vector multiplication, where M is multiplied by each column of N by one polynomial multiplication of degree n. The DFT–circuit cost of this operation is $O(n^2)$ size with constant time. The DFT–VLSIO model takes $O(n^{5/2})$ volume with $O(1)$ time.

The Toeplitz-matrix multiplication can be reduced to four triangular Toeplitz-matrix multiplications. Two of these multiplications involve one degree-n polynomial multiplication each, whereas the other two require n, degree-n polynomial multiplications. This gives a DFT–circuit (size, time) cost of $[O(n^2), O(1)]$ for Toeplitz-matrix multiplication. It takes a volume of $O(n^{5/2})$ and a time of $O(1)$ in the DFT–VLSIO model.

The *inverse* of an $n \times n$ triangular Toeplitz matrix is reducible to a degree-n polynomial division. Hence it can be done in the DFT–circuit with a size of n^2 and $O(1)$ time. The general Toeplitz inverse has the same complexity as GCD and polynomial interpolation. All these problems require $\log n$ stages of Toeplitz steps, as shown by Pan and Reif.[24] Hence they all take a time $O(\log n)$ with a size $O(n^2)$ for the DFT–circuit. In the DFT–VLSIO, $O(n^{5/2})$ volume and $O(\log n)$ time are needed.

G. Matrix Multiplication

1. *Input*

The input for matrix multiplication is two $n \times n$ matrices, A and B.

2. *Output*

The output for matrix multiplication is an $n \times n$ matrix C, such that $c_{i,j} = \Sigma_{k=0}^{n-1} a_{i,k}b_{k,j}$.

3. *Algorithm*

There are n^2, n-point inner products to be performed. For each inner product, we first perform the componentwise product and then compute the DFT of this product. Each inner product is the first component of the DFT, which is the sum of the products. This involves n^3 integer multiplications and n^2, n-point DFT operations.

4. *Analysis*

DFT–circuit Model: The n^3 integer multiplications take a size of n^3 and a time of $O(1)$. The following n^2, n-point DFT's will also take a size of n^3 and a time of $O(1)$ for a total cost of an n^3 size and $O(1)$ time.

DFT–VLSIO Model: The integer multiplications require $O(n^2 \log^3 n)$ volume and $O(1)$ time, assuming that the integer's sizes are $O(\log n)$ bits. The DFT cost is $O(1)$ time with an $n^{7/2}$ volume. Hence the complete computation takes $O(1)$ time with $O(n^{7/2})$ volume.

H. Matrix Inversion

1. *Input*

The input for matrix inversion is a nonsingular $n \times n$ matrix A.

2. *Output*

The output for this operation is the inverse matrix A^{-1} of A, such that $AA^{-1} = I$.

3. *Algorithm*

The algorithm is based on Newton's iteration method, which is described in Pan and Reif.[25] Let $A_i'^{-1}$ denote the ith-iteration approximation to A^{-1}. The next approximation to $A^{-1}, A_{i+1}'^{-1}$, is computed with the following equation: $A_{i+1}'^{-1} = 2A_i'^{-1} - A_i'^{-1}AA_i'^{-1}$.

The residual-error matrix $R_i = I - AA_i'^{-1}$ converges very rapidly, $R_i = R_0^{2^i}$. The choice of the initial approximation $A_0'^{-1}$ has a strong bearing on the convergence of the algorithm. Pan and Reif[25] discuss several methods of choosing $A_0'^{-1}$. In particular, the choice of $A_0'^{-1} = (1/m)A^T$, where m is the trace (sum of its diagonal entries) of A^TA, leads to good results. In fact, A^{-1} can be computed to within n bits of accuracy within $\log n$ iterations of the equation above.

4. *Analysis*

DFT–circuit Model: The computation of $A_0'^{-1}$ requires one matrix multiplication to compute A^TA, followed by the addition of n numbers and the division of A^T. The cost of matrix multiplication, which takes n^3 size and a time of $O(1)$, dominates.

Each iteration of the equation involves a scalar multiplication of a size of n^2 in a time of $O(1)$ and two matrix multiplications. Once again, the matrix-multiplication cost dominates. There are $\log n$ iterations of the equation taking an n^3 size and $O(\log n)$ time.

DFT–VLSIO Model: Once again the cost of matrix multiplication dominates, except for two differences. In this model we do not have division and multiplication gates. The division in the construction of $A_0'^{-1}$ can be performed in $O(1)$ time with a threshold size of $(1/\epsilon)n^{1+\epsilon}$ in a time of $1/\epsilon^3$ [or a volume of $O(n^3)$ in a time of $O(1)$], as is shown in Reif and Tate.[23] Hence the initial approximation can be done in a volume $O(n^5)$ and a time $O(1)$.

The cost of each iteration consists of one multiplication by 2, two matrix multiplications, and a matrix subtraction. The multiplication by 2 is just a left-hand shift in VLSI. The matrix multiplication cost is $O(1)$ time with an $n^{7/2}$ volume. Hence the total cost is $O(\log n)$ time and $O(n^5)$ volume.

I. Transitive Closure

Given an input of an $n \times n$ matrix A, its transitive-closure computation can be reduced to a matrix multiplication. Hence it takes a time $O(\log n)$ with an n^3 size in the DFT–circuit model. The DFT–VLSIO cost is $O(\log n)$ time with an n^5 volume.

4. Sorting

In this section, we describe sorting algorithms in the DFT–circuit and DFT–VLSIO models. We show that the sorting can be performed in a size of n^2 in a time of $O(1)$ deterministically. A randomized algorithm sorts with a size of $O(n^{3/2})$ in a time of $O(1)$ or in a time of $O(\log \log n)$ with a size of $O(n^{3/2}/\log n)$.

A. Input for Sorting Algorithms

The input is a sequence S of n values $a_1, a_2, \ldots, a_n$, where each value is $\log n$ bits long. The output is a sequence of the same values in a nondescending order.

DFT–circuit Model: The algorithm is a variation of Flashsort, as reported in Reif and Valiant.[45] Let us first show that a sequence of n numbers can be rank sorted in a time of $O(1)$ with an n^2 size. The gate $p_{i,j}$ compares a_i and a_j and has an output of 1 if $a_i > a_j$. The output is 0 otherwise. The rank of a_i is the sum of the output values of the gates $p_{i,1}$, $p_{i,2}, \ldots, p_{i,n}$. The zeroth component of the DFT of these n values yields this sum. Let us present the Flashsort-based sorting algorithm.

1. Take a random sample of n^ϵ elements of S to form a sample set S' of size n^ϵ, for $0 < \epsilon < 1/2$.
2. Rank sort S' in a time of $O(1)$ with an n size.
3. Form a set S'' by choosing every $(\log n)$th element from S'. A result reported in Reif and Valiant[45] shows that S'' splits S into the subsets of the expected size of $n^{1-\epsilon}c \log n$ and with a high probability, $1 - 1/(\log^c n)$, of a size of at most $(1 + \mu)n^{1-\epsilon}c \log n$, where μ is of the order of $d/(\log n)$ where $c, d = 2$.
4. Separate S into the sets $S_0, S_1, \ldots, S_t$ on the basis of S'', where t is in the range from $n^\epsilon/[(1 + \mu)c$

$\log n] + 1$ to $n^{\epsilon}/(c \log n) + 1$. This split can be done with rank ordering by use of $c[n^{\epsilon}/(\log n)]$-sized circuit for each element in S in a time of $O(1)$ for a total size of $c[n^{1+\epsilon}/(\log n)]$.

5. Use the algorithm recursively for each S_i in parallel until the subproblems are reduced to a size of n^{ϵ} each. Then the $n^{1-\epsilon}$ instances of n^{ϵ} subproblems can be rank sorted with an $n^{1+\epsilon}$ size in a time of $O(1)$. In this case, the whole algorithm takes the expected time of $O((1 - \epsilon)/\epsilon)$ with $O(n^{1+\epsilon})$ size. This is a time of $O(1)$ with $O(n^{3/2})$ size for the maximum value of ϵ. Or the recursion can terminate when the subproblems have a size of $O(1)$ (Ref. 46). Then the total time is $O(\log \log n)$ with a size of $O(n^{1+\epsilon}/\log n)[O(n^{3/2}/\log n)$ for the maximum value of ϵ].

The straight rank ordering gives an $O(1)$-time algorithm with an n^2 size.

DFT–VLSIO Model: The rank-sorting algorithm can be implemented in the DFT–VLSIO in a volume of $n^{5/2}$ in a time of $O(1)$. The n^2 comparisons of ($\log n$)-bit values correspond to n^2, ($\log n$)-point DFT's that can be done in an $n^2(\log n)^{3/2}$ volume in unit time. The Boolean values can be generated by VLSI circuits on the basis of the sign of the outcome. The summation of n Boolean values for the rank of each output takes one n-point DFT, hence all the n DFT's take a volume of $n^{5/2}$ with a time of $O(1)$.

The Flashsort algorithm can also be mapped into the DFT–VLSIO model. Step 2 takes a time of $O(1)$ and a volume of $n^{5/4}$ for rank-sorting $\sqrt{n}$ elements. In step 4, each element can be placed into the proper set by rank-sorting it with the $c\sqrt{n}/\log n$ splitters. This takes a time of $O(1)$ and a volume of $O(n^{5/4}/(\log n)^{5/2})$ for each input value, hence a time of $O(1)$ and a volume of $O(n^{9/4}/(\log n)^{5/2})$ in all. The rank sorting of $\sqrt{n}$, $\sqrt{n}$-sized subproblems in step 5 can be done in a time of $O(1)$ and a volume of $n^{7/4}$. Hence the total resource use for this option is a time of $O(1)$ and a volume of $O(n^{9/4}/(\log n)^{5/2})$. The second option takes the same volume but more time ($O(\log \log n)$), hence it is inferior to the first approach.

B. Element Distinctness

The input to this problem is a set of n, $\log n$-bit values. The problem is to determine if all the n words are distinct.

DFT–circuit Model: Sort the n elements of the set. Then compare each element in the sorted list with its left- and right-hand neighbors. The complexity is dominated by the sorting algorithm. Hence this is an $O(1)$-time algorithm with a size of $O(n^{3/2})$. All the other bounds from sorting also hold.

DFT–VLSIO Model: Once again, the sorting part has the complexity that was derived in the previous subsection. This is followed by $3n$ comparisons. Each comparison can be done in a time of $O(1)$ and $O(\log^{3/2} n)$ volume. Hence the sorting complexity of $O(1)$ time and $O(n^{9/4}/(\log n)^{5/2})$ volume still dominates the problem complexity.

5. String Matching

A. One-Dimensional String Matching

Given a binary string of $A = a_0, a_1, a_2, \ldots, a_n$ and a binary pattern of $B = b_0, b_1, b_2, \ldots, b_m$ with $m \le n$, find all the occurrences of B in A.

1. *Algorithm*

We reduce the problem of string matching to that of polynomial multiplication. The reduction of string matching to integer multiplication was known to Kosaraju[47] and is due to M. Fischer. We extend it to the reduction to a polynomial multiplication as follows.

Consider two polynomials derived from the strings A and B: $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{j=0}^{m} b_{m-j} x^j = \sum_{j=0}^{m} b_j' x^j$. The product of two polynomials $C(x) = A(x)B(x)$ can be written as $\sum_{i=0}^{m+n} c_i x^i$, where $c_i = \sum_{j=0}^{n} a_j b'_{i-j}$. Note that a coefficient c_i, for $m \le i \le n$, equals the number of places where $b_0 \, b_1 \ldots b_m$ 1-matches the substring $a_{i-m} \, a_{i-m+1} \ldots a_i$, i.e., the number of places where both strings have a 1. Repeat the same process for the complementary strings of A and B by building the polynomials $\bar{A}(x) = \sum_{i=0}^{n} \bar{a}_i x^i$ and $\bar{B}(x) = \sum_{j=0}^{m} \bar{b}_{m-j} x^j = \sum_{j=0}^{m} \bar{b}_j' x^j$. Once again compute the product $\bar{C}(x) = \bar{A}(x)\bar{B}(x) = \sum_{i=0}^{m+n} \bar{c}_i x^i$, such that $\bar{c}_i = \sum_{j=0}^{n} \bar{a}_j \bar{b}'_{i-j}$. The sum of c_i and $\bar{c}_i$ is $m + 1$ if $b_0 \, b_1 \ldots b_m$ matches the substring $a_{i-m} \, a_{i-m+1} \ldots a_i$. This procedure requires two polynomial multiplications and $O(n)$ scalar operations.

2. *Analysis*

DFT–circuit Model: The two polynomial multiplications of degree n take a time of $O(1)$ with $O(n)$ size. $O(n)$ scalar operations can also be done in an $O(1)$ time with $O(n)$ size.

DFT–VLSIO Model: The same reduction gives $O(1)$ time and $O(n^{3/2})$ volume VLSIO circuit. The complements can be derived in unit time and $O(n)$ volume by simple VLSI inverters. The sum of c_i and $\bar{c}_i$ for all $m \le i \le n$ can be accomplished in $O(n \log^{3/2} m)$ volume and unit time.

B. Two-Dimensional String Matching

This idea can be extended to 2-D string matching as well. Here the input is $A = (a_{i,j}|_{0 \le i,j \le n})$ and $B = (b_{i,j}|_{0 \le i,j \le m})$. We wish to find a match of B in A, that is, if $\exists i, j, \forall k, l \in (0, \ldots, m) a_{i+k,j+l} = b_{k,l}$.

1. *Algorithm*

The solution uses the 2-D DFT in a way that is similar to 1-D string matching. The 2-D string-matching problem can be reduced to a multiplication of two polynomials in two variables. Let us form polynomials $A(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} a_{i,j} x^i y^j$ and $B(x, y) = \sum_{k=0}^{m} \sum_{l=0}^{m} b_{m-k,m-l} x^k y^l$. Let the product of $A(x, y)$ and $B(x, y)$ be given by $\sum_{i=0}^{mn} \sum_{j=0}^{mn} c_{i,j} x^i y^j$, where $c_{i,j} = \sum_{k=0}^{n} \sum_{l=0}^{n} a_{i,j} b_{m-i+k,m-j+l}$. Similarly let $(\sum_{i,j=0}^{n} (1 - a_{i,j}) x^i y^j)[\sum_{k,l=0}^{m} (1 - b_{m-k,m-l}) x^k y^l](\sum_{i',j'=0}^{mn} \bar{c}_{i',j'} x^{i'} y^{j'})$. Then there is a match at i', j' if $c_{i',j'} + \bar{c}_{i',j'} = (m + 1)^2$.

Table 2. Comparison of the DFT-VLSIO Algorithms with the VLSIO Lower Bounds

Algorithm	Lower Bound $E(VT^{3/2})$	Upper Bounds E	Upper Bounds $VT^{3/2}$
1-D DFT	$\Omega(n^{3/2})$	$O(n^{3/2})$	$O(n^{3/2})$
Poly multiplication	$\Omega(n^{3/2})$	$O(n^{3/2})$	$O(n^{3/2})$
Barrel shift	$\Omega(n^{3/2})$	$O(n^{3/2})$	$O(n^{3/2})$
Prefix sum	$\Omega(n^{3/2})$	$O(n^{3/2})$	$O(n^{3/2})$
Toeplitz matrix	$\Omega(n^{3/2})$	$O(n^{5/2} \log n)$	$O(n^{5/2} \log^{3/2} n)$
Inverse and poly GCD and interpolation	$\Omega(n^{3/2})$	$O(n^{5/2} \log n)$	$O(n^{5/2} \log^{3/2} n)$
Matrix multiplication	$\Omega(n^3)$	$O(n^{7/2})$	$O(n^{7/2})$
Matrix-inversion transitive closure	$\Omega(n^3)$	$O(n^5)$	$O(n^5 \log^{3/2} n)$
Sorting	$\Omega(n^{3/2})$	$O(n^{5/2})$	$O(n^{5/2})$
Or		Randomized	Randomized
	$\Omega(n^{3/2})$	$O(n^{9/4}/(\log n)^{5/2})$	$O(n^{9/4}/(\log n)^{5/2})$
Element distinctness	Same as sorting	Same as sorting	Same as sorting
1-D string matching	$\Omega(n^{3/2})$	$O(n^{3/2})$	$O(n^{3/2})$
2-D string matching	$\Omega(n^{3/2})$	$O(n^{3/2})$	$O(n^{3/2})$

2. *Analysis*

DFT–circuit: This requires two n^2-point 2-D DFT operations. The resource requirements are $O(1)$ time with a linear size of $O(n)$.

DFT–VLSIO Model: Once again, $O(1)$-time and $O(n^{3/2})$-volume VLSIO circuit suffices for 2-D string matching.

6. Comparison with VLSIO Lower Bounds

As we stated in Section 1, we do not have lower bounds for the DFT–circuit model to compare the optimality of our algorithms. However, Barakat and Reif[21] showed a lower bound of $\Omega(I^{3/2})$ on $VT^{3/2}$ of a VLSIO computation, where V is the volume and T is the time of the computation. This lower bound applies to the DFT–VLSIO model as well. In Ref. 22, we derive a lower bound of $\Omega(I^{3/2})$ for the uniswitch energy of a VLSIO computation. All these algorithms can be realized as uniswitch computations. Then the uniswitch energy is equivalent to the volume. Hence the other useful lower bound on these problems in the DFT–VLSIO is $V = \Omega(I^{3/2})$. For most of the problems presented in Sections 3–5, the information complexity I is $\Omega(n)$. Table 2 compares our algorithms with respect to these lower bounds.

All the algorithms except those for matrix multiplication, inversion, and transitive closure are within a polylog factor of the lower bounds. The deterministic sorting algorithm is also off by a factor of n. Since the algorithms for the DFT–circuit and DFT–VLSIO models are identical, in the absence of lower bounds for the DFT–circuit model we surmise that the same type of optimality is achieved in the DFT–circuit model, as well.

7. Generalization of the Discrete Fourier Transform Model

Our assumption that an optical box can compute only a 2-D DFT in unit time is appropriate for many thin (linear) optical filters. But this assumption may be too restrictive to model thick optical components (such as volume holograms). In this case, we generalize our models so that an optical box or gate can compute a matrix multiplication of *displacement* rank d in unit time using an n-sized circuit (or n processors). The resulting model is called the DFT_d model here.

A matrix A has a displacement rank d if $A = \sum_{i=1}^{d} U_i L_i$, where U_i (L_i) is the upper- (lower-) triangular Toeplitz matrix (as defined in Subsection 3.F). The notion of displacement rank was first introduced in Ref. 48 and is restated in Ref. 49. Note that, if A has a displacement rank d, then it can be multiplied in $2d$ triangular Toeplitz matrix multiplications and thus $2d$ convolutions. Thus the DFT_d model can be simulated by the DFT_1 model to within a factor of $2d$ slowdown.

Recently, Karasik and Sharir[30] considered a more general model of optical computing to solve various computational geometry problems in constant time. They expand our optical-computing model by incorporating several constant-time primitive operations: pointwise addition, subtraction, and multiplication, complement, thresholding, 1-D and 2-D Fourier transforms, conformal change of coordinates, Radon transform, convolution, differentiation, and full thresholding. Note that we have assumed the availability of only *one* constant-time optical primitive operation, the 2-D Fourier transform. In this model, Karasik and Sharir give constant-time algorithms for computing unions, intersections, and Minkowski sums of plane figures. They also construct the convex hull of a planar set of points in constant time.

Another variant of the modeling of capabilities of optical technology emphasizes optical communication. Anderson and Miller[50] employed dynamically configurable optical routing switches to consider pointer-based efficient algorithms in a model called the optical communication parallel computer. MacKenzie and Ramachandran[31] explore the relation between the optical communication parallel computer model and the exclusive read concurrent write (ERCW)–PRAM model.

8. Conclusions

VLSI is, perhaps, the most commonly used technology for building parallel processors. However, we do not write our algorithms at that level of abstraction. The PRAM has proved to be a nice abstraction of parallel architectures from an algorithm designer's perspective. However, the primitive operations supported by a PRAM are not necessarily the strong points of an optical computer. This paper attempts to identify those natural strengths of optical-computing technology that support a framework for parallel-algorithm development.

As a first step in this direction, we have identified the capacity of current optical-computing technology to perform a DFT in unit time as the transform most easily exploited in algorithm design. Hence we propose a model of parallel computing that incorporates the DFT as a primitive operation. We have strived to develop a "bag of tricks" for an algorithm designer working with an optical-computing architecture that supports DFT operation. In particular, we used two algorithms very frequently. We provide an efficient algorithm for computing the 1-D DFT from the physically available 2-D DFT. We also provide an efficient solution to the parallel-prefix computation. Using these two techniques, we have provided constant-time or near-constant-time algorithms for many problems, including matrix computations, sorting, and string matching. The string-matching algorithm is particularly new. We also showed that most of these algorithms are optimal to within a polylog factor with respect to VLSIO lower bounds.

We believe that the development of such algorithm-design paradigms is crucial for bridging the gap between the optical-computing-architecture and algorithm-designer communities. An increased synergy between the two communities can lead to the identification of the best optical-computing-architecture primitives that are likely to be exploited by the algorithm designers.

There are many more applications that can benefit from optical computing, hence algorithms for many other applications need to be developed for this model. Similarly, some nontrivial lower bounds for this model would be desirable.

John Reif was supported in part by the Defense Advanced Research Projects Agency (DARPA)–Army Research Office under contract DAAL03-88-K-0195, by the Air Force under contract AFOSR-87-0386, by the DARPA–Information Systems Technology Office under contract N00014-88-K-0458, by NASA under subcontract 550-63 of primecontract NAS5-30428, and by the National Science Foundation (NSF) under grant NSF-IRI-91-00681. Akhilesh Tyagi was supported by the NSF under grant MIP-8806169, by the North Carolina Board of Science and Technology under grant 89SE04, and by a Junior Faculty Development award from the University of North Carolina (UNC), Chapel Hill. This study was performed when A. Tyagi was with UNC, Chapel Hill. A preliminary version of this paper appeared in Ref. 51. The authors are grateful to the two anonymous reviewers whose comments led to a significant improvement of the paper.

References and Notes

1. H. T. Kung, "Let's design algorithms for VLSI systems," in *Proceedings of the Caltech Conference on Advanced Research in VLSI: Architecture, Design, Fabrication* (Caltech, Pasadena, Calif., 1979), pp. 65–90.
2. C. D. Thompson, "Area–time complexity for VLSI," in *Proceedings of the ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, 1979), pp. 81–88.
3. D. P. Casasent, "A hybrid digital/optical computer system," IEEE Trans. Comput. **C-22,** 852–858 (1973).
4. A. Huang, "Design for an optical general purpose digital computer," in *1980 International Optical Computing Conference I*, W. T. Rhodes, ed., Proc. SPIE **232,** 119–127 (1980).
5. A. Huang, "Architectural considerations involved in the design of an optical digital computer," Proc. IEEE **72,** 780–786 (1984).
6. E. S. Maniloff, K. M. Johnson, and J. Reif, "Holographic routing network for parallel processing machines," in *Holographic Optics II: Principles and Applications*, Y. N. Denisyuk and T. H. Jeong, eds., Proc. SPIE **1183,** 283–300 (1989).
7. A. Louri, "Three-dimensional optical architecture and data-parallel algorithms for massively parallel computing," IEEE Micro **11,** 24–81 (1991).
8. L. R. McAdams and J. W. Goodman, "Liquid-crystal 1 × *n* optical switch," Opt. Lett. **15,** 1150–1152 (1990).
9. L. R. McAdams, R. N. McRuer, and J. W. Goodman, "Liquid-crystal optical routing switch," Appl. Opt. **29,** 1304–1307 (1990).
10. A. Sawchuk and T. Strand, "Digital optical computing," Proc. IEEE **72,** 758–779 (1984).
11. J. W. Goodman, "Architectural development of optical data processing systems," Aust. J. Electr. Electron. Eng. **2,** 139–149 (1982).
12. T. E. Bell, "Optical computing: a field of flux," IEEE Spectrum **23,** 34–57 (1986).
13. H. J. Caulfield, J. A. Neff, and W. T. Rhodes, "Optical computing: the coming revolution in optical signal processing," Laser Focus **19**(11), 100–110 (1983).
14. D. G. Feitelson, *Optical Computing, A Survey for Computer Scientists* (MIT Press, Cambridge, Mass., 1988).
15. J. L. Horner, *Optical Signal Processing* (Academic, San Diego, Calif., 1987).
16. K. Iizuka, *Engineering Optics*, 2nd ed., Vol. 35 of Springer Series in Optical Sciences (Springer-Verlag, Berlin, 1983).
17. M. V. Klein and T. E. Furtak, *Optics*, 2nd ed. (Wiley, New York, 1986).
18. A. D. McAulay, *Optical Computer Architectures: the Application of Optical Concepts to Next Generation Computers* (Wiley, New York, 1991).
19. S. Fortune and J. Wyllie, "Parallelism in random access machines," in *Proceedings of the ACM Symposium on the Theory of Computing* (Association for Computing Machinery, New York, 1978), pp. 114–118.
20. W. J. Savitch and M. Stimson, "Time bounded random access machines with parallel processing," J. Assoc. Comput. Mach. **26,** 103–118 (1979).
21. R. Barakat and J. Reif, "Lower bounds on the computational efficiency of optical computing systems," Appl. Opt. **26,** 1015–1018 (1987).
22. A. Tyagi and J. Reif, "Energy complexity of optical computa-

tions," in *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing* (Institute of Electrical and Electronics Engineers, New York, 1990), pp. 14–21.

23. J. Reif and S. Tate, "On threshold circuits and polynomial computation," SIAM J. Comput. **21,** 896–908 (1992).
24. V. Pan and J. Reif, "Some polynomial and Toeplitz matrix computations," in *Proceedings of the Twenty-seventh IEEE Symposium on Foundations of Computer Science* (Institute of Electrical and Electronics Engineers, New York, 1987), pp. 173–184.
25. V. Pan and J. Reif, "Fast and efficient parallel solution of dense linear systems," Comput. Math. Appl. **17,** 1481–1491 (1989).
26. D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *Proceedings of the ACM Symposium on the Theory of Computing* (Association for Computing Machinery, New York, 1987), pp. 1–6.
27. M. Ajtai, J. Komlos, and E. Szemeredi, "An $O(n \log n)$ sorting network," Combinatorica **3,** 1–19 (1983).
28. R. Cole, "Parallel merge sort," SIAM J. Comput. **17,** 770–785 (1988).
29. O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin, "Highly parallelizable problems," in *Proceedings of the ACM Symposium on the Theory of Computing* (Association for Computing Machinery, New York, 1989), pp. 309–319.
30. Y. B. Karasik and M. Sharir, "Optical computational geometry," in *Proceedings of the Eighth Annual Symposium on Computational Geometry* (Association for Computing Machinery, New York, 1992), pp. 232–241.
31. P. D. MacKenzie and V. Ramachandran, "ERCW PRAM's and optical communication," in *Proceedings of the European Conference on Parallel Processing, EUROPAR '96*, Vol. 1124 of Lecture Notes on Computer Science Series (Springer-Verlag, Berlin, 1996), pp. 293–302.
32. C. D. Thompson, "A complexity theory for VLSI," Ph.D. dissertation (Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., 1980).
33. V. P. Heuring, H. F. Jordan, and J. Pratt, "Bit-serial architecture for optical computing," Appl. Opt. **31,** 3213–3224 (1992).
34. H. F. Jordan, "Pipelined digital optical computing," OCS Tech. Rep. 89-34 (Optoelectronic Computing Center, University of Colorado, Boulder, Colo., 1989).
35. D. Gabor, "Light and information," in *Progress in Optics*, E. Wolf, ed. (North-Holland, Amsterdam, The Netherlands, 1961), pp. 111–153.
36. F. T. Leighton and A. L. Rosenberg, "Three-dimensional circuit layouts," SIAM J. Comput. **15,** 793–813 (1986).
37. F. P. Preparata, "Optimal three-dimensional VLSI layouts," Math. Systems Theory **16,** 1–8 (1983).
38. I. Parberry and G. Schnitger, "Parallel computation with threshold functions," J. Comput. Syst. Sci. **36,** 278–301 (1988).
39. R. Moenck and A. B. Borodin, "Fast modular transforms via division," in *Conf. Record, IEEE 13th Annual Symp. on Switching and Automata Theory* (IEEE Press, Piscataway, N.J., 1972), pp. 90–96.
40. A. B. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numerical Problems* (Elsevier, New York, 1975).
41. D. Bini and V. Pan, *Polynomial and Matrix Computations* (Birkhauser, Boston, Mass., 1994).
42. R. C. Agarwal and C. S. Burrus, "Fast one-dimensional digital convolution by multidimensional techniques," IEEE Trans. Acoust. Speech Signal Process. **ASSP-22,** 1–10 (1974).
43. J. L. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach* (Morgan Kaufmann, San Francisco, Calif., 1990).
44. J. Vuillemin, "A combinatorial limit to the computing power of VLSI circuits," IEEE Trans. Comput. **C-32,** 294–300 (1983).
45. J. Reif and L. Valiant, "A logarithmic time sort on linear size networks," J. Assoc. Comput. Mach. **34,** 60–76 (1987).
46. Technically, for the probabilistic analysis of Flashsort to work, the problem size should be at least a polynomial in log (polylog). At that point a less efficient deterministic algorithm can be used. However, for simplicity of exposition we have chosen to give this *inaccurate* version, as they both lead to the same amount of resources.
47. S. R. Kosaraju, Department of Computer Science, Johns Hopkins University, Baltimore, Md. (personal communication, 1989).
48. M. Morf and T. Kailath, "Recent results in least-squares estimation theory," Ann. Econ. Soc. Meas. **6,** 261–274 (1977).
49. T. Kailath, S. Y. Kung, and M. Morf, "Displacement ranks of matrices and linear equations," J. Math. Anal. Appl. **68,** 395–407 (1979).
50. R. J. Anderson and G. L. Miller, "Optical communication for pointer-based algorithms," Computer Research Institute Tech. Rep. 88-14 (University of Southern California, Los Angeles, Calif., 1988).
51. J. H. Reif and A. Tyagi, "Efficient parallel algorithms for optical computing with the DFT primitive," in *Proceedings of the Tenth Conference on the Foundations of Software Technology and Theoretical Computer Science*, Vol. 472 of Lecture Notes on Computer Science (Springer-Verlag, Berlin, 1990), pp. 149–160.