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 $\mathbf{C} = \mathbf{AB}$, where $\mathbf{A}$ is of the size $n_1 \times n_2$, $\mathbf{B}$ is of size $n_2 \times n_3$, and $\mathbf{C}$ 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 $\mathbf{A}$ and $\mathbf{B}$ 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 $\mathbf{A}$ 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.

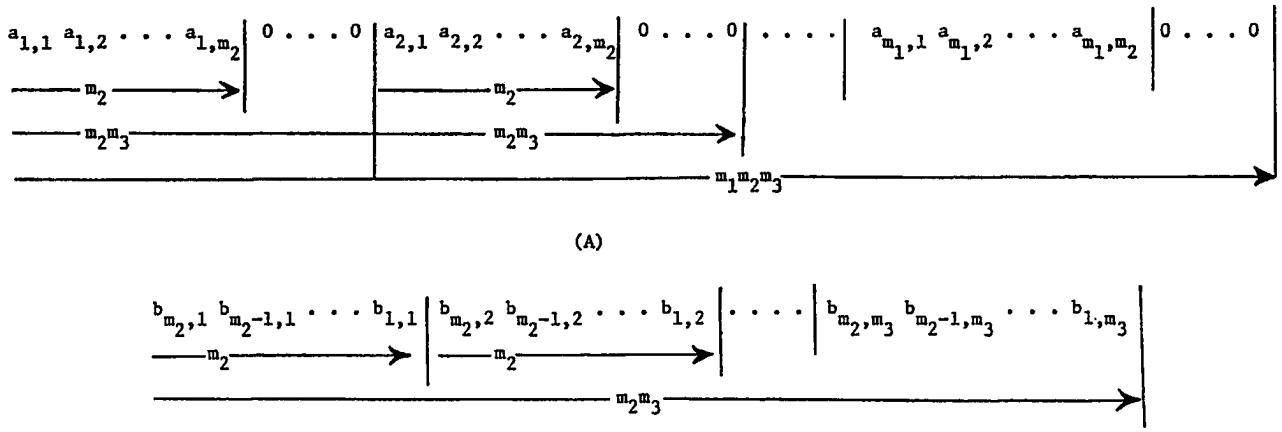

Fig. 1. Layout of the p vector (A) and q vector (B).

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:

$$\alpha: \quad s = (i-1)n_2n_3 + j - 1; \tag{9a}$$

$$\beta: \quad (i-1)n_2n_3 < s < (i-1)n_2n_3 + n_2; \tag{9b}$$

$$\gamma: \quad t = m - s = kn_2 - j; \tag{9c}$$

$$\delta: \quad t < n_2n_3. \tag{9d}$$

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

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

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

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

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

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

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

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 matrices 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.

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		

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**, $\mathbf{E}_i^\dagger\,\mathbf{A}$ as the ith block row of **A**, and $\mathbf{E}_i^\dagger\,\mathbf{AE}_j$ as the (i,j)th block element of **A**; thus we have

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

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.,

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

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

jth colum of $\mathbf{A} = \mathbf{A}\mathbf{e}_j$,

ith 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):

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

Consequently,

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

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).