Efficient Method for Computing Double Matrix Products

David R. O'Hallaron
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Charles M. Fiduccia
Supercomputing Research Center
Lanham, MD 20706

Abstract

We introduce an efficient method for computing matrix products of the form $Y = AXB$, where A and B are sparse and constant. We analyze the complexity of the method, develop quantitative criteria for determining when it can be used effectively, and demonstrate its use in a Kalman filter.

1 Introduction

Matrix products of the form $Y = AXB$, where A and B are sparse and constant, arise frequently in estimation algorithms. For example, a Kalman filter for target tracking applications must compute such a product repeatedly and in real-time [1].

In this paper, we describe an efficient method, which we call the *outer product method*, for computing the matrix Y. Simply put, the method works as follows: For each y_{ij}, an outer product $F^{(ij)}$ is precomputed using the *ith* row of A and the *jth* column of B. The expression for y_{ij} is then generated by a straightforward inspection of matrix $F^{(ij)}$. As we shall see, the outer product method can be easily automated, is simple to understand, and has a nice visual interpretation. Further, when A and B are sufficiently sparse the expressions generated are more efficient than the expressions generated using a conventional inner product method, where the computation of Y is decomposed into two distinct matrix multiplications.

In Section 2 we describe the outer product method. In Section 3 we derive the time complexity of the expressions formed using the outer product method,

compare these expressions to those formed using a conventional inner product method, and give quantitative criteria for choosing one method over the other. Finally, in Section 4 we apply the outer product method to a Kalman filter for target tracking applications.

2 Description

Consider the matrix product

$$Y_{m\times q} = A_{m\times n}X_{n\times p}B_{p\times q} \tag{1}$$

where $A = (a_{ij})$ and $B = (b_{ij})$ are sparse and constant matrices, $X = (x_{ij})$ is arbitrary, and $Y = (y_{ij})$. Let $r_i(M)$ and $c_j(M)$ denote the *ith* row and *jth* column of some matrix $M = (m_{ij})$, respectively. Let e_i denote a column vector such that the only non-zero element in e_i is a 1 in the *ith* position. The following identities will be useful:

$$\begin{aligned} M &= \sum_{i,j} m_{ij}e_i e_j^T && (2)\\ m_{ij} &= e_i^T M e_j && (3)\\ r_i(M) &= e_i^T M && (4)\\ c_j(M) &= M e_j && (5) \end{aligned}$$

Assuming that scalar multiplication is commutative, we see from (2) that

$$\begin{aligned} Y &= AXB \\ &= A\left(\sum_{k,l} x_{kl}e_k e_l^T\right)B \\ &= \sum_{k,l} x_{kl}Ae_k e_l^T B \end{aligned} \tag{6}$$

From (3) and (6) we have

$$\begin{aligned} y_{ij} &= e_i^T Y e_j \\ &= e_i^T\left(\sum_{k,l} x_{kl}Ae_k e_l^T B\right)e_j \\ &= \sum_{k,l} x_{kl}(e_i^T Ae_k)(e_l^T Be_j) \end{aligned}$$

Noticing that $(e_i^T A e_k)$ and $(e_l^T B e_j)$ are scalars, and using the fact that a scalar is its own transpose, we see that

$$\begin{aligned} y_{ij} &= \sum_{k,l} x_{kl} (e_i^T A e_k)^T (e_l^T B e_j)^T \\ &= \sum_{k,l} x_{kl} (e_k^T A^T e_i)(e_j^T B^T e_l) \\ &= \sum_{k,l} x_{kl} e_k^T (A^T e_i)(e_j^T B^T) e_l \end{aligned}$$

and from (4) and (5) we have

$$\begin{aligned} y_{ij} &= \sum_{k,l} x_{kl} e_k^T \left(c_i(A^T) r_j(B^T) \right) e_l \\ &= \sum_{k,l} x_{kl} e_k^T \left(r_i(A)^T c_j(B)^T \right) e_l \end{aligned}$$

If we let $F^{(ij)} = (f_{kl}^{(ij)})$ denote the constant outer product

$$F^{(ij)} = r_i(A)^T c_j(B)^T = \begin{bmatrix} a_{i1} \\ \vdots \\ a_{in} \end{bmatrix} \begin{bmatrix} b_{1j} & \cdots & b_{pj} \end{bmatrix} = \begin{bmatrix} a_{i1}b_{1j} & \cdots & a_{i1}b_{pj} \\ \vdots & & \vdots \\ a_{in}b_{1j} & \cdots & a_{in}b_{pj} \end{bmatrix} \quad (7)$$

then by (7) and (3) we have the central result of this paper:

$$\begin{aligned} y_{ij} &= \sum_{k,l} x_{kl} e_k^T F^{(ij)} e_l \\ &= \sum_{k,l} f_{kl}^{(ij)} x_{kl} \end{aligned} \quad (8)$$

In words, each $F^{(ij)}$ is a constant matrix (obtained by taking the outer product of the transposed *ith* row of A with the transposed *jth* column of B), having the same dimensions as X, that shows us in a straightforward and visual manner how to compute y_{ij}. An intuitive interpretation of (8) is to view each $F^{(ij)}$ as a "mask" that is placed on top of X. Each non-zero element of the mask is multiplied by the corresponding element of X (some readers may recognize this as the Hadamard product); the sum of these products is y_{ij}. Equation (8) suggests a technique, which we call the *outer product method*, for generating efficient expressions that compute Y:

Outer Product Method (General Case): For each y_{ij}, precompute $F^{(ij)}$. Then use the non-zero elements of $F^{(ij)}$ and the corresponding elements of X to generate the expression that computes y_{ij}.

For example, consider a simple case where $m = n = p = q = 2$ and let

$$A = \begin{bmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{bmatrix} \; X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \; B = \begin{bmatrix} b_{11} & 0 \\ 0 & b_{22} \end{bmatrix}$$

To generate the expression for y_{12}, we precompute $F^{(12)}$ according to (7).

$$F^{(12)} = r_1(A)^T c_2(B)^T = \begin{bmatrix} a_{11} \\ a_{12} \end{bmatrix} \begin{bmatrix} 0 & b_{22} \end{bmatrix} = \begin{bmatrix} 0 & a_{11}b_{22} \\ 0 & a_{12}b_{22} \end{bmatrix}.$$

Using the precomputed matrix $F^{(12)}$ as a mask for X, we immediately see from (8) that

$$y_{12} = (a_{11}b_{22})x_{12} + (a_{12}b_{22})x_{22}.$$

Expressions for y_{11}, y_{21}, and y_{22} are generated in a similar manner.

There is an interesting special case of (1) where A, B, X, and Y are of equal dimension ($m = n = p = q$), where $B = A^T$, and where X and Y are symmetric.

$$Y_{n\times n} = A_{n\times n} X_{n\times n} A^T_{n\times n} \tag{9}$$

One of the nice properties of the outer product method is that it can be trivially modified to exploit the symmetry of Y.

Outer Product Method (Symmetric Case): For each y_{ij}, $j \geq i$, precompute $F^{(ij)}$. Then use the non-zero elements of $F^{(ij)}$ and the corresponding elements of X to generate the expression that computes element y_{ij}.

3 Analysis

In this section we analyze the complexity of the expressions generated by the outer product method versus those obtained using a conventional inner product method.

3.1 General Case

In the general case we are concerned with matrix products of the form

$$Y_{m\times q} = A_{m\times n} X_{n\times p} B_{p\times q} \tag{10}$$

where $A = (a_{ij})$ and $B = (b_{ij})$ are sparse and constant matrices, and $X = (x_{ij})$ is arbitrary. We also assume that A and B have no identically zero rows or columns. This implies that $Y = (y_{ij})$ is full.

Let s be the number of non-zero elements in A and let s_i be the number of non-zero elements in row i of A. Let t be the number of non-zero elements in B and let t_j be the number of non-zero elements in column j of B.

Similarly, let u be the number of unit elements in A and let u_i be the number of unit elements in row i of A. Let v be the number of unit elements in B and let v_j be the number of unit elements in column j of B.

We assume that multiplication by unit elements is trivial and therefore not counted.

Outer Product Method

Consider the number of scalar multiplications required to compute Y using the outer product method. Each y_{ij} requires $s_i t_j - u_i v_j$ scalar multiplications, for a total of

$$\sum_{i=1}^{m} \sum_{j=1}^{q} (s_i t_j - u_i v_j) = st - uv \tag{11}$$

scalar multiplications to compute Y. Notice that the number of scalar multiplications is simply the sum of the number of non-zero non-unit elements in all of the F matrices.

Now consider the number of scalar additions. Each y_{ij} requires $s_i t_j - 1$ scalar additions, for a total of

$$\sum_{i=1}^{m} \sum_{j=1}^{q} (s_i t_j - 1) = st - mq \tag{12}$$

scalar additions to compute Y.

Inner Product Method

The conventional alternative to the outer product method is to perform two matrix multiplications,

$$Z = XB \tag{13}$$

$$Y = AZ \tag{14}$$

where

$$z_{ij} = \sum_{k} x_{ik} b_{kj}$$

and

$$y_{ij} = \sum_k a_{ik} z_{kj}$$

and where Z and Y are both full. Using this so called *inner product method*, a non-zero element in column j of B multiplies the n elements in column j of X. Recognizing that scalar multiplication by a unit element is free, we have $t - v$ non-zero non-unit elements of B, each of which multiplies n elements of X, for a total of $n(t - v)$ scalar multiplications to compute (13). Similarly, equation (14) requires $q(s - u)$ scalar multiplications, for a total of

$$q(s - u) + n(t - v) \tag{15}$$

scalar multiplications.

Now consider the number of scalar additions required by the inner product method. By the assumption that X is full, each z_{ij} in (13) requires $t_j - 1$ scalar additions, for a total of

$$\sum_{i=1}^{n} \sum_{j=1}^{q} (t_j - 1) = n(t - q) \tag{16}$$

scalar additions to compute Z in (13). Similarly, each y_{ij} in (14) requires $s_i - 1$ scalar additions, for a total of

$$\sum_{i=1}^{m} \sum_{j=1}^{q} (s_i - 1) = q(s - m) \tag{17}$$

scalar additions in (14). Combining (16) and (17), we see that the inner product method requires

$$q(s - m) + n(t - q) \tag{18}$$

scalar additions to compute Y.

3.2 Symmetric Case

Here we are concerned with matrix products of the form

$$Y_{n \times n} = A_{n \times n} X_{n \times n} A^T_{n \times n} \tag{19}$$

where A, B, X, and Y are of equal dimension ($m = n = p = q$), where $B = A^T$ ($s = t, u = v$), and where X and Y are symmetric. Again, we assume that A and B have no identically zero rows or columns and X is full.

Outer Product Method

Let T denote the number of scalar multiplications required to compute each entry of a non-symmetric Y and let D denote the number of scalar multiplications required to compute the entries along the diagonal of Y. Then we have

$$(T - D)/2 + D = (T + D)/2$$

scalar multiplications to compute the entries of a symmetric Y. Since $T = s^2 - u^2$ and $D = \sum_i s_i^2 - \sum_i u_i^2$, we have

$$\left(\left(s^2 + \sum_{i=1}^{n} s_i^2 \right) - \left(u^2 + \sum_{i=1}^{n} u_i^2 \right) \right) /2 \tag{20}$$

scalar multiplications to compute Y. Similarly, since each y_{ij} requires $s_i s_j - 1$ scalar additions we have

$$\left(s^2 + \sum_{i=1}^{n} s_i^2 \right) /2 - n(n+1)/2 \tag{21}$$

scalar additions to compute Y.

Inner Product Method

Consider the number of scalar multiplications required by the inner product method. While a method for exploiting the symmetry of X to reduce the number of multiplications in (13) has been developed [4, 3], the method increases the number of additions and will not be considered in this paper. Thus the inner product method requires $n(s - u)$ scalar multiplications to perform (13). Because of the symmetry of Y, equation (14) requires

$$\sum_{i=1}^{n} (n - i + 1)(s_i - u_i)$$

scalar multiplications, for a total of

$$n(s - u) + \sum_{i=1}^{n} (n - i + 1)(s_i - u_i) \tag{22}$$

scalar multiplications to compute Y.

Now consider the number of scalar additions required by the inner product method for the symmetric case. We know that

$$\sum_{i=1}^{n} \sum_{j=1}^{n} (s_j - 1) = n(s - n) \tag{23}$$

Method	$Y = AXB$ (General Case)	$Y = AXA^T$ (Symmetric Case)
OPM	$st - uv$	$\left(\left(s^2 + \sum_{i=1}^{n} s_i^2\right) - \left(u^2 + \sum_{i=1}^{n} u_i^2\right)\right)/2$
IPM	$q(s-u) + n(t-v)$	$n(s-u) + \sum_{i=1}^{n}(n-i+1)(s_i - u_i)$

Figure 1: Scalar Multiplication Counts

scalar additions are required to compute Z in (13) and

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(s_i - 1) = \left(\sum_{i=1}^{n}\sum_{j=i}^{n} s_i\right) - n(n+1)/2 \tag{24}$$

scalar additions are required to compute Y in (14). Combining (23) and (24), we see that the inner product method requires

$$\left(\sum_{i=1}^{n}\sum_{j=i}^{n} s_i\right) - n(n+1)/2 + n(s-n) \tag{25}$$

scalar additions to compute Y in the symmetric case.

3.3 Summary of Analysis

The number of scalar multiplications required by the the outer product method (OPM) and the inner product method (IPM) for both the general and symmetric cases are summarized in Figure 1. The number of scalar additions are given in Figure 2.

Focusing on the scalar multiplications, we see from (11) and (15) that the outer product method is a good alternative to the the inner product method for those cases where

$$st - uv \leq q(s-u) + n(t-v). \tag{26}$$

Method	$Y = AXB$ (General Case)	$Y = AXA^T$ (Symmetric Case)
OPM	$st - mq$	$\left(s^2 + \sum_{i=1}^{n} s_i^2\right)/2 - n(n+1)/2$
IPM	$q(s-m) + n(t-q)$	$\left(\sum_{i=1}^{n}\sum_{j=i}^{n} s_i\right) - n(n+1)/2 + n(s-n)$

Figure 2: Scalar Addition Counts

To further understand the efficiencies of the outer product method versus the inner product method, consider a simpler instance of (10) where A, B, X, and Y are of equal dimension ($m = n = p = q$), $B = A^T$ ($s = t, u = v$), and X is arbitrary. In this case, expressions generated by the outer product method require $s^2 - u^2$ scalar multiplications, and expressions generated by the inner product method require $2n(s-u)$ scalar multiplications. For this simple case the outer product method is an attractive alternative to the inner product method when

$$\begin{aligned} s^2 - u^2 &\leq 2n(s-u) \\ (s-u)(s+u) &\leq 2n(s-u) \\ (s+u) &\leq 2n. \end{aligned}$$

Notice that the outer product method exploits the constant nature of A and B at the cost of recomputing some partial results. Conversely, the inner product method saves computations by reusing previously computed partial results at the cost of ignoring the constant nature of A and B. Loosely speaking, the efficiency of the expressions generated using the outer product method versus those generated by the inner product method is proportional to the sparsity of the A and B matrices. As these matrices become sparser, the outer product method becomes a more attractive alternative.

We conclude this section with a brief discussion of some qualitative properties of the outer product method. First, the F matrices provide us with a simple, compact, and highly visual representation of the expressions for the elements of

Method	Mults	Adds
OPM	288	243
IPM	216	162

Figure 3: Kalman Filter Operation Counts (General Case)

the solution matrix. Besides its pedagogical value, this simplicity makes the outer product method easy to automate. Second, it is easy to exploit special structure in the X and Y matrices. We have already discussed one sutionKalman Filter Example Kalman filtering is a popular technique for estimating the state of a dynamic system from noisy measurements. Applications include target tracking, orbit estimation, image processing, and industrial process control. A key computation in the solution of the Kalman equations for target tracking applications is a sequence of matrix multiplications of the form

$$X(t+1) = AX(t)A^T, \quad t = 0, 1, 2, \cdots \tag{27}$$

where A and X are $(n \times n)$ matrices, A is sparse, constant, and upper triangular, and X is symmetric [1]. Note that (27) is an instance of the symmetric case (19) from the previous section. In the target tracking filter discussed in [1], $n = m = p = q = 9$, and matrix A is a state transition matrix typically of the form

$$A = \begin{bmatrix} 1 & & & \delta & & & \sigma & & \\ & 1 & & & \delta & & & \sigma & \\ & & 1 & & & \delta & & & \sigma \\ & & & 1 & & & \delta & & \\ & & & & 1 & & & \delta & \\ & & & & & 1 & & & \delta \\ & & & & & & \alpha & & \\ & & & & & & & \alpha & \\ & & & & & & & & \alpha \end{bmatrix}$$

where α, δ, and σ are constants. Notice that there are 18 non-zeros ($s = t = 18$) and 6 unit elements ($u = v = 6$) in A. Consider the number of scalar multiplications and additions needed to compute (27). If we ignore the symmetry of $X(t+1)$, then the outer product method generates expressions that require

$$s^2 - u^2 = 288$$

Method	Mults	Adds
OPM	162	138
IPM	177	108

Figure 4: Kalman Filter Operation Counts (Symmetric Case)

scalar multiplications and

$$s^2 - n^2 = 243$$

scalar additions. On the other hand, the inner product method generates expressions that require

$$2n(s - u) = 216$$

scalar multiplications and

$$2n(s - n) = 162$$

scalar additions. These operation counts are summarized in Figure 3. Clearly, if we choose to ignore the symmetry of $X(t+1)$ the inner product method is the preferred method.

However, if we exploit the symmetry of $X(t+1)$, then the outer product method generates expressions that require

$$\left(\left(s^2 + \sum_{i=1}^{n} s_i^2\right) - \left(u^2 + \sum_{i=1}^{n} u_i^2\right)\right)/2 = 162$$

scalar multiplications and

$$\left(s^2 + \sum_{i=1}^{n} s_i^2\right)/2 - n(n+1)/2 = 138$$

scalar additions. On the other hand the inner product method generates expressions that require

$$n(s - u) + \sum_{i=1}^{n}(n - i + 1)(s_i - u_i) = 177$$

scalar multiplications and

$$\left(\sum_{i=1}^{n}\sum_{j=i}^{n} s_i\right) - n(n+1)/2 + n(s - n) = 108$$

scalar additions. These operation counts are summarized in Figure 4. We see that in the symmetric case, the outer product method requires fewer scalar multiplications, yet more scalar additions than the inner product method. Which method is to be preferred depends on the relative speeds of scalar addition and multiplication on the target computer. Some simple analysis gives the answer. Let

$$t_M = \alpha t_A$$

where t_M is the time required to perform a scalar multiplication and t_A is the time required to perform a scalar addition. Also, let

$$t_{OPM} = \alpha t_A(162) + t_A(138)$$

and

$$t_{IPM} = \alpha t_A(177) + t_A(108),$$

where t_{OPM} is an approximation of the running time of a subroutine that computes (27) using the outer product method and t_{IPM} is an approximation of the running time of a subroutine that computes (27) using the inner product method. Now if we consider the case where

$$t_{OPM} \leq t_{IPM}$$

we find that

$$\begin{aligned} \alpha t_A(162) + t_A(138) &\leq \alpha t_A(177) + t_A(108) \\ \alpha t_A(-15) &\leq t_A(-30) \\ \alpha t_A &\geq t_A(2) \\ \alpha &\geq 2 \end{aligned}$$

Thus for this particular Kalman filtering example, the outer product method is to be preferred over the inner product method if the cost of a scalar multiplication is at least twice that of a scalar addition. Otherwise the inner product method should be be used.

4 Conclusion

We have presented a technique, called the outer product method, for generating efficient expressions that compute the matrix product $Y = AXB$, where A and B are constant and sparse. The method is simple, easily automated, and has a nice visual interpretation. We analyzed the complexity of the resulting expressions and developed quantitative criteria for determining when the method might be used. Finally, we applied the method to a computation from a target tracking Kalman filter.

Acknowledgements

Thanks to Bob Mattheyses and Kishan Baheti for reading and commenting on earlier versions of the manuscript.

References

[1] R. S. Baheti. Efficient approximation of kalman filter for target tracking. *IEEE Transactions on Aerospace and Electronic Systems*, AES-22(1):8–14, January 1986.

[2] R. S. Baheti and D. R. O'Hallaron. Efficient parallel implementation of target tracking kalman filter. In *27th IEEE Conference on Decision and Control*, IEEE, Austin, TX, December 1988.

[3] C. M. Fiduccia. *Multiplication by a Symmetric Stochastic Matrix*. Technical Report 79CRD006, General Electric Corporate Research and Development Center, March 1979.

[4] C. M. Fiduccia. *On the Algebraic Complexity of Matrix Multiplication*. PhD thesis, Brown University, Providence, R.I., 1973.

[5] D. R. O'Hallaron and R. S. Baheti. Fast mapping of a kalman filter on warp. In *Proceedings of SPIE Symposium, Real-Time Signal Processing XI*, Society of Photo-Optical Instrumentation Engineers, San Diego, CA, August 1988.