Aviv Abramovich

University of Haifa,
Department of Computer Science

Diagonal Order Matrices Multiplication

As Shai's Discusses in his article "Algorithms in HElib", he suggests 3 way to preform a matrix by vector multiplication, only by SIMD operations, so it will be applicable in HElib's encrypted arrays operations, like multiply 2 encrypted arrays, adding them, shift and rotate.
As he describe it, you can represent a matrix by vector of encrypted vectors in 3 different forms:

1. Rows order - each sub-vector is a row vector in the original matrix
2. Columns Order - each sub-vector is a column vector in the matrix
3. Diagonal Order - each sub-vector is a diagonal vector in the matrix, that start in the first (upper) row of the matrix.

For each of these representations, Shai describes how to multiply that kind of matrix (encrypted or plain text) by a single vector, and the result is always a Column vector.

For our purposes, we want to perform a matrices multiplication (matrix by matrix multiplication) on encrypted matrices. We used Shai's methods on both rows, columns and diagonal order. To adapt these methods to matrix by matrix multiplication, we just broke the second matrix to columns vectors, and preform Shai's methods in loop for each of the second matrix vectors.
This method works well, in different time for each of the 3 representation kinds, but have one disadvantage: because the matrix by vector multiplication always return the result as a column vector, the results of theses matrix by matrix multiplication will always be a columns order matrix, independently in the first matrix representation.
It not sound very bad, but after a long research and many tests, we found that rows order and columns order forms of multiplication are very very slow in compared to the diagonal order kind of multiplication, some times even more than 10 times slower. That means that we can multiply a matrix in diagonal order by a matrix in diagonal order, but the result would be a matrix in a columns order, so the next multiplication must be in columns order that very slow.

So the 2 main problems are:

1. the second matrix in each multiplication **MUST** be a columns order.
2. The result of each multiplication in a columns order matrix, so the next multiplications may be slower.

We discussed about this "problem" with Shai, and his suggestion for some N matrices multiplication, is to ordering the multiplications in specific order, when the N-1 first be in diagonal order, and the last one be in columns order, and then perform the multiplication every time on the last 2 matrices, so all the N-1 multiplication be in diagonal order form.
That solution is good, but only for really specific cases. Say we want to perform some kind of algorithm that use the multiplication not in that linear order, that solution won't be hopeful.

Our goal is to use the fastest form of multiplication, the diagonal order kind, on all the matrices, and use **ONLY** diagonal order matrices, so we won't be stuck with columns order matrices that make our algorithm slower, or even to stuck because the needed matrices is not in "suitable" representation.
In similar to Shai's multiplications methods, using only SIMD operations, I developed a diagonal order matrix by matrix multiplication (both of the matrices are in diagonal order) that it result is also a diagonal order matrix, so we can run the whole algorithm with diagonal order matrices.

let A and B be 2 squares diagonal order matrices, so A is vector of vectors, that each of these sub-vectors are a diagonal in A: $[A_0, A_1 ... A_{n-1}]$ (and same for B). Let C_i be the i-th diagonal vector in the result, so we can calculate it that way

$$C_i = \sum_{j=0}^{n-1} A_j * (B_{(i-j)\bmod n} <<< j)$$

when $vec <<< j$ stands for "rotate the vector vec j indexes left, or -j to right).
Example:
I'll prove that method works for 3*3 matrix, but it stands for any size of matrix.
Let A and B be 2 3*3 matrices:

$$A=\begin{pmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & i_1 \end{pmatrix} B=\begin{pmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & i_2 \end{pmatrix}$$

So our result should be

$$C=\begin{pmatrix} a_1a_2+b_1d_2+c_1g_2 & a_1b_2+b_1e_2+c_1h_2 & a_1c_2+b_1f_2+c_1i_2 \\ d_1a_2+e_1d_2+f_1g_2 & d_1b_2+e_1e_2+f_1h_2 & d_1c_2+e_1f_2+f_1i_2 \\ g_1a_2+h_1d_2+i_1g_2 & g_1b_2+h_1e_2+i_1h_2 & g_1c_2+h_1f_2+i_1i_2 \end{pmatrix}$$

if we disassemble A and B to diagonal vectors, they would be:

$A_0=[a_1,e_1,i_1], A_1=[b_1,f_1,g_1], A_2=[c_1,d_1,h_1]$

$B_0=[a_2,e_2,i_2], B_1=[b_2,f_2,g_2], B_2=[c_2,d_2,h_2]$

and C should be :

$$C_0=\begin{pmatrix} a_1a_2+b_1d_2+c_1g_2 \\ d_1b_2+e_1e_2+f_1h_2 \\ g_1c_2+h_1f_2+i_1i_2 \end{pmatrix}, C_1=\begin{pmatrix} a_1b_2+b_1e_2+c_1h_2 \\ d_1c_2+e_1f_2+f_1i_2 \\ g_1a_2+h_1d_2+i_1g_2 \end{pmatrix}, C_2=\begin{pmatrix} a_1c_2+b_1f_2+c_1i_2 \\ d_1a_2+e_1d_2+f_1g_2 \\ g_1b_2+h_1e_2+i_1h_2 \end{pmatrix}$$

So if we use the formula for calculate C_0 we get:

$A_0B_0+A_1(B_2<<<1)+A_2(B_1<<<2)$

$$\begin{pmatrix} a_1a_2 \\ e_1e_2 \\ i_1i_2 \end{pmatrix}+\begin{pmatrix} b_1d_2 \\ f_1h_2 \\ g_1c_2 \end{pmatrix}+\begin{pmatrix} c_1g_2 \\ d_1b_2 \\ h_1f_2 \end{pmatrix}=\begin{pmatrix} a_1a_2+b_1d_2+c_1g_2 \\ d_1b_2+e_1e_2+f_1h_2 \\ g_1c_2+h_1f_2+i_1i_2 \end{pmatrix}=C_0$$

Same for C_1, C_2:

$A_0B_1+A_1(B_0<<<1)+A_2(B_2<<<2)$

$$\begin{pmatrix} a_1b_2 \\ e_1f_2 \\ i_1g_2 \end{pmatrix}+\begin{pmatrix} b_1e_2 \\ f_1i_2 \\ g_1a_2 \end{pmatrix}+\begin{pmatrix} c_1h_2 \\ d_1c_2 \\ h_1d_2 \end{pmatrix}=\begin{pmatrix} a_1b_2+b_1e_2+c_1h_2 \\ d_1c_2+e_1f_2+f_1i_2 \\ g_1a_2+h_1d_2+i_1g_2 \end{pmatrix}=C_1$$

$A_0B_2+A_1(B_1<<<1)+A_2(B_0<<<2)$

$$\begin{pmatrix} a_1c_2 \\ e_1d_2 \\ i_1h_2 \end{pmatrix}+\begin{pmatrix} b_1f_2 \\ f_1g_2 \\ g_1b_2 \end{pmatrix}+\begin{pmatrix} c_1i_2 \\ d_1a_2 \\ h_1e_2 \end{pmatrix}=\begin{pmatrix} a_1c_2+b_1f_2+c_1i_2 \\ d_1a_2+e_1d_2+f_1g_2 \\ g_1b_2+h_1e_2+i_1h_2 \end{pmatrix}=C_2$$

For conclusion, this method let us to run our algorithms, when all the encrypted matrices are in diagonal order only, so there won't be any problem of "not suitable representation" of multiplication, and all the multiplication be fast, because that method is based on the exact operations of diagonal order matrix by vector multiplication (as represented by Shai) without any other operation (except that this method is for a whole matrix and not for a specific vector), so it base on the fastest kind of multiplication.