Matrix Vector Multiplication on a Vector Processor

M.G. v/d Horst and R. Mak

December 4, 2007

Abstract

We show how the implementation of a matrix vector multiplication on a vector processor can be accomplished using only a multiply-accumulate (MAC) and a rotation operation. The correctnes of this implementation is shown in a mathematical proof. The implementation is especially suited for cases in which the multiplication occurs often and with a constant matrix. The only real requirement is that the matrix used in the multiplication is square.

1 Introduction

The multiplication of a matrix with a vector is a common operation, in signal processing especially. When implementing this operation on a vector processor the question rises what the best implementation would be and which of the processor's vector operations are usefull. We cannot answer this question for every possible vector processor, but in this paper we will show that only two vector operations are necessary.

For implementing matrix vector multiplication (MVM), the MAC operation and the rotation operation are the only two required vector operations. We hope this information will help designers in choosing the right vector processor for the device they are designing. And that it will help vector processor designers to choose which operations to optimize.

The next section contains the definitions we will use in this report. Section 3 gives the proof and pseudo-code for the implementation. For now we will assume that the vector processor will is able to handle vectors of any size, in section 4 we will consider how to handle the real vector processors for which this assumption does not hold. The conclusion can be found in section 5.

2 Definitions

This section contains the mathematical definitions that we will use throughout this paper.

2.1 Common definitions

The matrix vector multiplication we want to implement will be denoted as:

$$r = Ca$$

where C is a $N \times N$ matrix and a and r are both vectors of length N. We construct them from their individual elements like this:

$$\begin{array}{ll} C = [c_{i,j}]_{i,j} & 0 \leq i,j < N \\ a = [a_i]_i & 0 \leq i < N \\ r = [r_i]_i & 0 \leq i < N \end{array}$$

So $c_{0,0}$ is the top-left element of matrix C, a_0 is the first element of a, etc.

Because of the multiplication $r = Ca$, the following relationship holds:

$$r_i = (\sum j : 0 \leq j < N : c_{i,j} a_j) \text{ for } 0 \leq i < N \tag{1}$$

2.2 Vector operations

First we define the point wise product of two vectors of length N:

$$(z = x \otimes y) \Rightarrow (z = [x_i y_i]_i \text{ for } 0 \leq i < N)$$

So the MAC operation can now be denoted as: $z := x \otimes y + z$.

The rotation operation can be described as the multiplication with the rotation matrix R. The $N \times N$ matrix R and its inverse R^{-1} have the following form (elements not shown are all zero):

$$R = \begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ 1 & & & \end{pmatrix} \quad R^{-1} = \begin{pmatrix} & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{pmatrix}$$

So multiplying R with a matrix A will result in the rows of matrix A being reordered in such a way that each row has is now one step closer to the top, and the first row of A has become the last row of the result. The rotation operation can therefore be described as a multiplication of R with the operand.

Note that this also holds for multiplication with vectors. The effect on a vector can be described as follows:

$$R^i [z_j]_j = [z_{j+i \bmod N}]_j \text{ for } 0 \leq j < N \tag{2}$$

3 Implementation

To implement the MVM we will have to describe the multiplication with the help of the MAC and rotate operations. The first paragraph of this section will be used to prove the following lemma:

Lemma 1 $r = Ca = (\sum i : 0 \leq i < N : R^i(\tilde{c}_i \otimes a))$ *where* $\tilde{c}_i = diag(R^{-i}C)$.

Before we start with the proof we will explain why we chose this representation. It does contain the rotate and MAC operations, but it is not very clear how the implementation will work out. Note however, that lemma 1 can also be rewritten to the following form:

$$r = Ca = \tilde{c}_0 \otimes a + R(\tilde{c}_1 \otimes a + R(\ldots + R(\tilde{c}_{N-1} \otimes a)\ldots))$$

In this form the MAC operations and rotation operations are easier to see. If $\tilde{c}_i$ can be calculated off-line these are also the only operations needed. So instead of using matrix C in the program we use the $\tilde{c}_i$ vectors. But we should not jump ahead, we will prove the lemma first.

3.1 Proof

First let us examine $\tilde{c}_i$, it is a vector that is the diagonal of a rotation of matrix C. The elements of $\tilde{c}_i$ can be defined as follows:

$$(\tilde{c}_i)_j = c_{(j-i) \bmod N, j} \text{ for } 0 \leq j < N$$

This allows us to use the following derivation:

$$
\begin{array}{ll}
& r = (\sum i : 0 \leq i < N : R^i(\tilde{c}_i \otimes a)) \\
= & \quad \{ \text{ Definition of } \otimes \text{ and } 0 \leq j < N \} \\
& r = (\sum i : 0 \leq i < N : R^i[(\tilde{c}_i)_j a_j]_j) \\
= & \quad \{ \text{ Use rotation (2) } \} \\
& r = (\sum i : 0 \leq i < N : [(\tilde{c}_i)_{j+i \bmod N} a_{j+i \bmod N}]_j) \\
= & \quad \{ \ r = [r_j]_j \ \} \\
& r_j = (\sum i : 0 \leq i < N : (\tilde{c}_i)_{j+i \bmod N} a_{j+i \bmod N}) \\
= & \quad \{ \text{ Definition of } (\tilde{c}_i)_j \ \} \\
& r_j = (\sum i : 0 \leq i < N : c_{((j+i \bmod N)-i) \bmod N, j+i \bmod N} a_{j+i \bmod N}) \\
= & \quad \{ \text{ Arithmetic } \} \\
& r_j = (\sum i : 0 \leq i < N : c_{j \bmod N, j+i \bmod N} a_{j+i \bmod N}) \\
= & \quad \{ \ 0 \leq j < N \ \} \\
& r_j = (\sum i : 0 \leq i < N : c_{j, j+i \bmod N} a_{j+i \bmod N}) \\
= & \quad \{ \text{ Dummy change: } i := i - j \bmod N \ \} \\
& r_j = (\sum i : 0 \leq i - j \bmod N < N : c_{j,i} a_i) \\
= & \quad \{ \text{ Arithmetic } (0 \leq i < N) \text{ and summation is commutative } \} \\
& r_j = (\sum i : 0 \leq i < N : c_{j,i} a_i) \\
= & \quad \{ \text{ Switch dummies (i,j:=j,i) } \} \\
& r_i = (\sum j : 0 \leq j < N : c_{i,j} a_j) \\
= & \quad \{ \text{ Definition of matrix vector multiplication (1) } \} \\
& \mathit{true}
\end{array}
$$

Therefore lemma 1 holds.

3.2 Pseudo-code

To implement the MVM we will choose as an invariant:

$$r = (\sum i : N - n \leq i < N : R^{n-N+i}(\tilde{c}_i \otimes a))$$

For $n = N$ this wil result in the desired postcondition described by lemma 1 and for $n = 0$ we can achieve the invariant with the expression $r := 0$.

To maintain the invariant under $n := n + 1$ we have to do the following:

$$
\begin{array}{ll}
 & (\sum i : N - n \leq i < N : R^{n-N+i}(\tilde{c}_i \otimes a))(n := n + 1) \\
= & \quad \{ \text{ Substitution } \} \\
 & (\sum i : N - (n+1) \leq i < N : R^{(n+1)-N+i}(\tilde{c}_i \otimes a)) \\
= & \quad \{ \text{ Split off } i = N - (n+1) \} \\
 & (\sum i : N - n \leq i < N : R^{(n+1)-N+i}(\tilde{c}_i \otimes a)) + R^{(n+1)-N+(N-(n+1))}(\tilde{c}_{N-n-1} \otimes a) \\
= & \quad \{ \text{ Arithmetic } \} \\
 & (\sum i : N - n \leq i < N : RR^{n-N+i}(\tilde{c}_i \otimes a)) + (\tilde{c}_{N-n-1} \otimes a) \\
= & \quad \{ \text{ Multiplication distributes over addition } \} \\
 & R(\sum i : N - n \leq i < N : R^{n-N+i}(\tilde{c}_i \otimes a)) + (\tilde{c}_{N-n-1} \otimes a) \\
= & \quad \{ \text{ Invariant } \} \\
 & Rr + (\tilde{c}_{N-n-1} \otimes a) \\
= & \quad \{ \text{ Commutativity of addition } \} \\
 & (\tilde{c}_{N-n-1} \otimes a) + Rr
\end{array}
$$

This formula clearly shows the rotation and MAC operation. So the pseudo code for the implementation becomes:

```
r := 0;
n := 0;
do (n != N) ->
  rotate(r);
  MAC(r, c(n), a);
  n := n + 1;
od
```

The notation is straightforward. The rotate operation $r := Rr$ is denoted by `rotate(r)`, the MAC operation $r := (c(n) \otimes a) + r$ by `MAC(r, c(n), a)` and `c(n)` denotes $\tilde{c}_{N-n-1}$.

Note that we assume that $\tilde{c}_i$ can just be retrieved from memory. This should pose no problem, since something similar has to happend with other implementations of the MVM. Each implementation needs to retrieve the all elements of C. It just so happens that our implementation does not need them in a row by row or column by column order.

Retrieving elements of a matrix row by row, or column by column is usually easier than our retrieval scheme. However, we could construct a new matrix C' with $\tilde{c}_i$ forming the rows or columns; then those same retrieval schemes can be used.

This is a drawback of our implementation, however our implementation may still be favorable over others. This is, for example, the case if the MVM has to be executed several times with the same matrix; the construction only has to take place once and so it's cost can then be divided. Of course, if the component that provides C can be easily redesigned to provide C' then there is no construction cost at all.

4 Vector Size

FIXME

5 Conclusion

We have shown how a matrix vector multiplication can be implemented for a square matrix. This implementation requies only two basic vector operations, namely the MAC operation and the rotate operation.

The actual efficiency of this implementation depends on the processor on which it is implemented, and how the matrix is provided. We believe this to be a good implementation in many cases because it uses few operations and its operational complexity is $O(N)$.

Also, when matrix C is a banded matrix, then the operational complexity becomes $O(B)$, where B is the width of the band. (FIXME: explain this)

Acknowledgements

The idea for this implementation method came from Rudolf Mak.