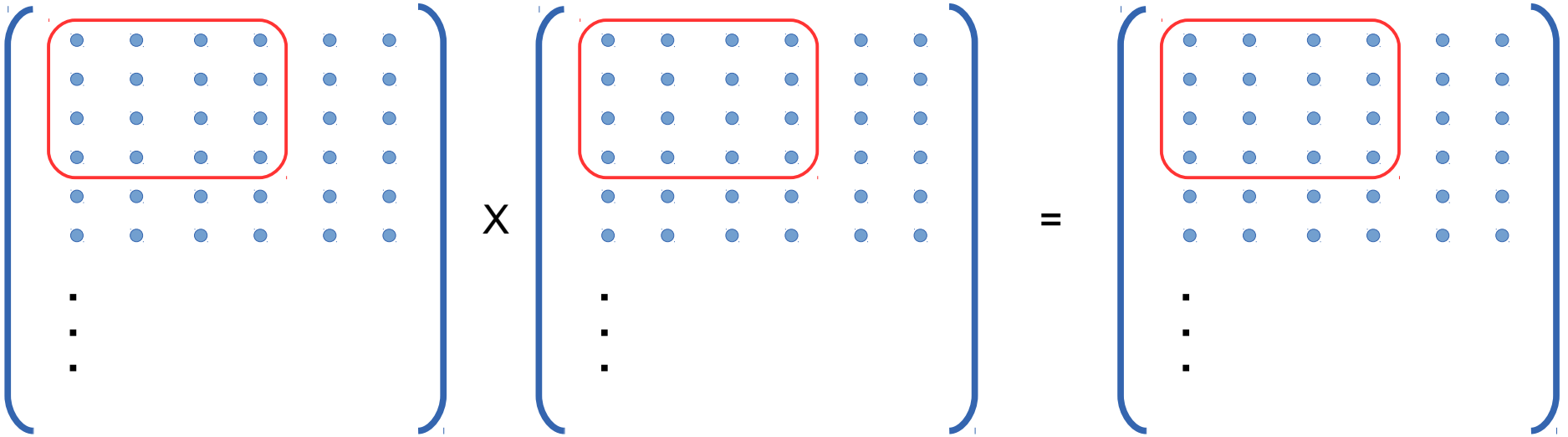
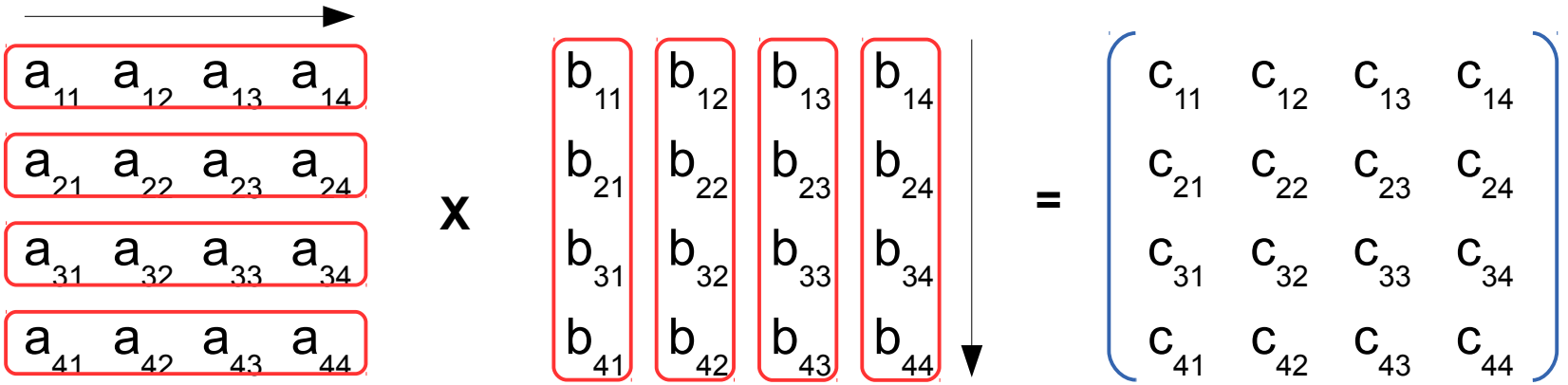
Matrix Multiplication – Blocked SSE

Since each SSE vector is 4 floats in size, we choose block size of 4 (i.e. we operate on a 4x4 block at a time). This means that each time we load four vectors and finish operating on them before moving onto next block (next four vectors).



Matrix Multiplication – Blocked SSE

Simple matrix multiplication algorithm:



$$c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} + a_{14} \cdot b_{41}$$

$$c_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22} + a_{13} \cdot b_{32} + a_{14} \cdot b_{42}$$

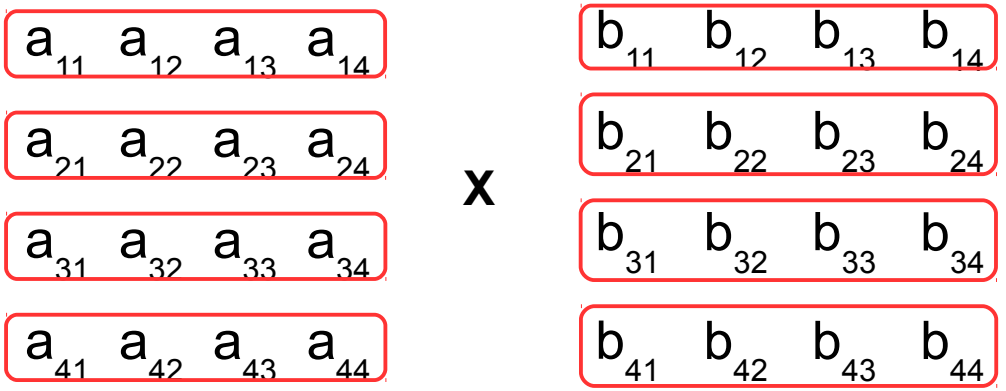
.
.

$$c_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + a_{23} \cdot b_{31} + a_{24} \cdot b_{41}$$

.
.

Matrix Multiplication – Blocked SSE

Problem with vectorized load is that we get a row (of four floats) each time we perform a load, whereas we need to have a column from the second matrix for the algorithm to work correctly.



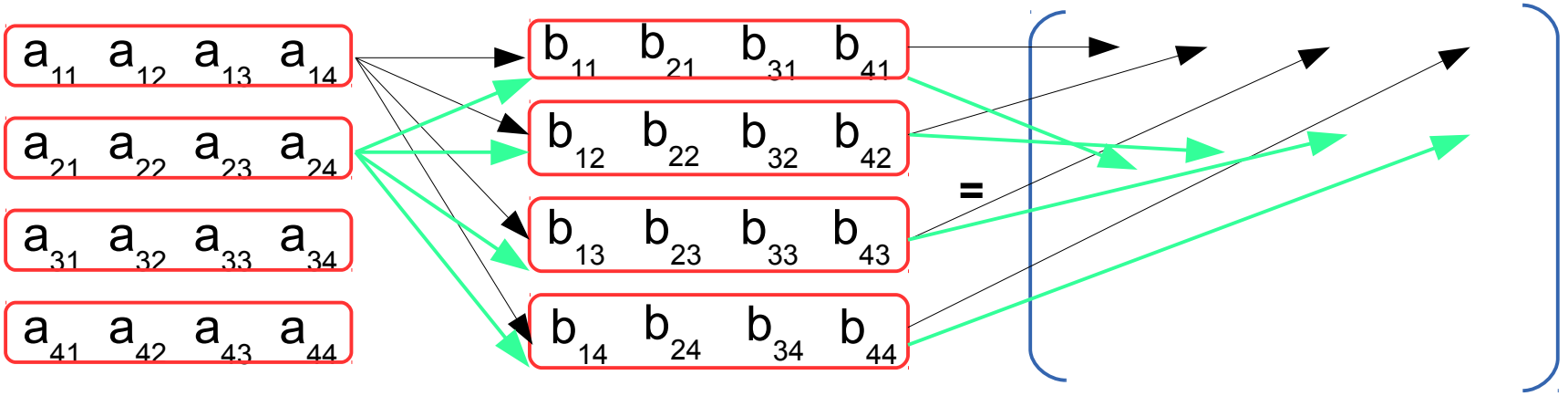
Solution would be to transpose the second block (use __MM_TRANSPOSE4_PS).

(you can read the specification for __MM_TRANSPOSE4_PS at

http://msdn.microsoft.com/en-us/library/5hah127h(v=vs.90).aspx

Matrix Multiplication – Blocked SSE

The second block will look like below after the transpose, which will allow us to perform vectorized multiplications row-wise and get correct results.



Note that the arrows on the left represent reduction operation on each pair of corresponding vectors (dot product, which produces a scalar value out of two vectors). Such a reduction operation could be implemented using dot-product instruction (_mm_dp_ps). Such a reduction operation could also be implemented using normal vector multiplication followed by another reduction operation. Can you think of such a reduction operation?

Please refer to lab manual for a list of vector instructions. You can also find plenty of information including good examples online, such as Microsoft MSDN library.

Matrix Multiplication – Blocked SSE

Note that when we are updating the result matrix after doing the computations for each row, we need to add the newly computed partial value to the current row value in result matrix, rather than overwriting the old value.

Also note that since we store into memory using a vectorized store command (four floats), we need to compute all the reduction operations for each row and prepare a single vector containing a reduced value for each element in the row, since vectorized store command requires a vector to write into memory (4 floats).