1. For each of the following loop nests:

```plaintext
for k = 2,N-1
    for i = 2,N-1
        for j = 2,N-1
            A(i,j-1,k-1) = A(i,j,k) - A(i-1,j-1,k+1)
        end for
    end for
end for

for i = 2,N-1
    for j = 2,N-1
        A(i,j) = (A(i-1,j-1)+A(i+1,j-1)+A(i-1,j+1)+A(i+1,j+1))/4
    end for
end for
```

(a) List all data dependences, stating the kind of dependence and the distance vector.
(b) What loop permutations are valid? Why?
(c) Which loops can be unrolled? Why?
(d) What tiling is possible? Why?

**Solution:**

For the first loop nest, there are two possible flow/output dependence vectors: one because of the pair of references \([A(i,j-1,k-1), A(i,j,k)]\) and another because of the pair \([A(i,j-1,k-1), A(i-1,j-1,k+1)]\). Considering the first, let the write occur in iteration \((kw,iw,jw)\) and the read in iteration \((kr,ir,jr)\). For the same array element to be accessed in both, we must have: \(iw=ir\); \(jw-1=jr\); \(kw-1=kr\). If there is a flow dependence, the write must occur before the read, and the dependence distance vector would be \((kr-kw,iw-ir,jw-jr)\), i.e. \((-1,0,-1)\). But this is not lexicographically positive, which means that a flow dependence does not exist for this pair. If an anti-dependence exists, the read must occur before the write, and the dependence vector would be \((kw-kr,iw-ir,jw-jr)\), or \((1,0,1)\).

Considering the pair \([A(i,j-1,k-1), A(i-1,j-1,k+1)]\), for the same element to be written at iteration \((kw,iw,jw)\) and read at iteration \((kr,ir,jr)\), we must have \(iw=ir-1\); \(jw-1=jr\); \(kw=kr+1\), i.e. \(iw-ir=-1\), \(jw-jr=0\), and \(kw-kr=2\). This implies that we have no valid flow dependence \((kr-kw,iw-ir,jw-jr)\) but we do have an anti-dependence \((kw-kr,iw-ir,jw-jr)\) = \((2,-1,0)\).
Clearly, there cannot be any output dependence.

The valid permutations are kji and jki. The (2, -1, 0) dependence makes the permuted dependence vector lexicographically negative for the other three permutations.

For unrolling by an arbitrary factor, the sufficient condition is that the permutation that moves the unrolled loop innermost (leaving all other loops in the same relative order) be valid. Unrolling the innermost loop is of course always valid. Unrolling the middle "i" loop is also valid, since the two permuted dependence vectors stay positive. But unrolling the outer loop is problematic because the permuted dependence vector is now (-1, 0, 2), which is lexicographically negative.

Tiling a set of loops is valid when that set of loops is fully permutable. So 3D tiling is not valid, since the kij loops are not fully permutable. But tiling just the ij loop pair is valid, since no lexicographically negative vectors result. Also, tiling of just the innermost j loop with respect to the outer k loop is valid - the resulting permuted vector is (0, 2, -1).

For the second loop nest, four pairs are to be tested for flow/anti dependence: \([A(i,j), A(i-1, j-1)]\), \([A(i,j), A(i+1, j-1)]\), \([A(i,j), A(i-1, j+1)]\), and \([A(i,j), A(i+1, j+1)]\).

\([A(i,j), A(i-1, j-1)]\): \(iw=ir-1\) and \(jw=jr-1\); So \(ir-iw=1\) and \(jr-jw=1\), i.e. read follows write, or flow dependence (1, 1).

\([A(i,j), A(i+1, j-1)]\): \(iw=ir+1\) and \(jw=jr-1\); So \(ir-iw=1\) and \(jr-jw=-1\), i.e. read precedes write, or anti dependence (1, -1).

\([A(i,j), A(i-1, j+1)]\): \(iw=ir-1\) and \(jw=jr+1\); So \(ir-iw=1\) and \(jr-jw=-1\), i.e. read follows write, or flow dependence (1, -1).

\([A(i,j), A(i+1, j+1)]\): \(iw=ir+1\) and \(jw=jr+1\); So \(ir-iw=1\) and \(jr-jw=1\), i.e. read precedes write, or anti dependence (1, 1). Due to the (1, -1) dependence, loop interchange is invalid. The (1, -1) dependence also means that unrolling of the outer loop is not to be done. The (1, -1) dependence also disallows tiling.

2. Consider the following three forms of matrix multiplication: \(ijk\), \(ikj\), and \(jik\).

(a) Assuming that we have a direct-mapped cache of size 64K words and linesize 8 words, what is the number of cache misses you would expect for each of the forms (assume row-major ordering of the elements of 2D arrays), if the matrices are all 512 x 512?

(b) Estimate the number of cache misses for each case if the cache is fully associative.

To simplify your analysis, ignore misses due to cross-interference between elements of different arrays (i.e. do the analysis as if each array had its own cache).

**Solution:**

The cache has a capacity of 64K words while each array has 256K elements. Thus only a quarter of any array can fit into the cache. This means that elements (i, j) and
`(i+128,j)` will map to the same cache block in a direct mapped cache. Thus, if the inner loop accesses an array by column, by the time half the column has been accessed, the elements in the first quarter column would have been removed from the cache due to conflict misses.

```
for i = 1,N
  for j = 1,N
    for k = 1,N
      C(i,j) = C(i,j) + A(i,k)*B(k,j)
    end for
  end for
end for
```

Array C will have total temporal and spatial reuse; the only misses will be initial cold misses for both direct and associative caches. So total number of misses = `N*N/B`.

Array A: for a fixed `i` and `j`, as `k` is varied, row-`i` will be accessed, occupying `N/B` adjacent blocks in the cache. As `j` is varied, the same row will be repeatedly accessed in cache. So again, only the initial compulsory misses will occur, i.e. total number of misses = `N*N/B`.

Array B: For fixed `i` and `j`, as `k` is varied, elements in a column of `B` are accessed. When `j` is changed by one, the adjacent column of `B` is accessed, but will incur misses for a direct mapped cache (hits for a fully associative cache since only `N/B` blocks would be used). However, no temporal reuse is possible, even with a fully associative cache since there is insufficient capacity to hold all of `B` till the outer loop `i` changes. So misses for direct-mapped cache will be `N*N*N`, and `N*N*N/B` for a fully associative cache.

```
for i = 1,N
  for k = 1,N
    for j = 1,N
      C(i,j) = C(i,j) + A(i,k)*B(k,j)
    end for
  end for
end for
```

Array A will have total temporal and spatial reuse; the only misses will be initial cold misses for both direct and associative caches. So total number of misses = `N*N/B`.

Array B will have complete spatial reuse since it is accessed by row in the innermost loop. But no temporal reuse is possible since there is insufficient capacity to hold all of `B` till the outer loop `i` changes. So misses for both direct and associative cache will be `N*N*N/B`.

3
Array C: for a fixed \( i \) and \( k \), as \( j \) is varied, row-\( i \) will be accessed, occupying \( N/B \) adjacent blocks in the cache. As \( k \) is varied, the same row will be repeatedly accessed in cache. So only the initial compulsory misses will occur, with either direct-mapped or fully associative cache, i.e. total number of misses = \( N*N/B \).

\[
\begin{align*}
\text{for } j = 1,N \\
\quad &\text{for } i = 1,N \\
\quad &\quad \text{for } k = 1,N \\
\quad &\quad \quad C(i,j) = C(i,j) + A(i,k)*B(k,j) \\
\quad &\quad \text{end for} \\
\quad &\text{end for} \\
\text{end for}
\end{align*}
\]

Array C: There will be complete temporal reuse since the innerloop index \( k \) does not appear in \( C \)'s addressing. For a fixed \( j \), as \( i \) is varied, a column of \( C \) is accessed. This will result in conflict misses for a direct-mapped cache, but not for a fully associative cache. So the total number of misses for a direcy-mapped cache will be \( N*N \) (one miss for each element), while it will only be \( N*N/B \) for a fully associative cache.

Array A: It is accessed by row and so will have complete spatial reuse. The middle loop causes different rows to be accessed. For a fixed \( j \), the entire array is accessed once. As \( j \) is changed, the elements will have to be accessed again since the cache does not have sufficient capacity to enable any temporal reuse. So number of misses is \( N*N*N/B \) for both direct-mapped and associiative cache.

Array B: It is accessed by column in the innermost loop. The middle loop is the k-loop and causes repeated access of the same column. For a direct mapped cache, each access will be a miss, but for a fully associative cache we will only have compulsory cold misses (as long as the capacity is at least \( N \) blocks or \( N*B \) words). So total number of misses for direct-mapped cache will be \( N*N*N \), but only \( N*N/B \) for a fully associative cache.