Examples of Cache Miss Estimation for Matrix Multiplication

Consider a cache of size 64K words and linesize 8 words, and arrays 512 x 512. Perform cache miss analysis for the following three forms of matrix multiplication: *ijk*, *ikj*, and *jik*, considering both direct-mapped and fully associative caches. The arrays are stored in row-major order. To simplify analysis, ignore misses due to cross-interference between elements of different arrays (i.e. perform the analysis for each array, ignoring accesses to the other arrays).

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.

Another way to come to the same conclusion regarding misses on repeated column-wise access is as follows. Without loss of generality, let us assume that array element A[0][0] maps to memory location 0. Since linesize is 8 words, A[0][0]..A[0][7] will occupy memory block 0, and map to cache set 0. A[0][8]..A[0][15] map to cache set 1, and so on. With a consecutive set of 8 elements in a row occupying a memory block, the last element in row 0, A[0][511] would map to set (512/8)-1=63. Wrapping around in row major order, element A[1][0] would map to set 64. Since the elements of row 1 would also occupy a total of 64 blocks, A[2][0] must map to set 2*64=128. Generalizing, element A[k][0] will map to set [(k*64) mod 8K], since the number of sets in a cache with capacity of 64K words and linesize of 8 words is 8K. We will have a collision and therefore eviction of A[0][1] from cache set 0 if any other element A[k][0] also maps to set 0. The smallest value of k for this to happen is when k*64 equals 8*1024, i.e., k = 128. So we can conclude that when A[128][0] is accessed, the newly loaded block into set 0 would evict the previously loaded block containing A[0][0]..A[0][7]. Similarly, the block of data containing A[129][0] will map to set 64 and cause eviction of the previously loaded block containing A[0][8]..A[0][15]. Thus only 128 out of the total of 8K sets in the cache is actually utilized and no reuse of cache lines occurs.

Analysis of *ijk* form

```
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] += A[i][k]*B[k][j];
```

Array A: For a fixed i and j, as k is varied, row-i will be accessed, resulting in N/B cold misses (one miss for every B accesses). As j is varied, the same row will be repeatedly accessed in cache, resulting in hits. The same costs recur for each outer iteration i, as different rows of A are accessed. So the total number of misses is N*N/B for both direct-mapped and fully associative caches.

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 lines would be used). However, no temporal reuse is possible for different iterations of the i-loop, 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 a direct-mapped cache will be N*N*N, and N*(N/B)*N for a fully associative cache.

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

The tabular analysis is shown below. Each entry in the table represents a multiplier with respect to that loop index, on the number of cache misses. For the innermost loop, it is the total number of cache misses for a fixed value of the outer two loops. For the middle loop, it represents how many times the inner-loop miss count will get multiplied as we run through all iterations of the middle loop, for a fixed value of the outer loop.

For example, with a direct-mapped cache, the innermost loop for the *ijk* form is k. For array A, which is indexed as A[i][k], for N iterations of k, for a fixed value of outer loops at 0, the sequence of accesses is A[0][0], A[0][1], A[0][2], ... A[0][N-1]. Since contiguous memory elements are being accessed, there will be one miss every B accesses, i.e., the table entry is N/B. The middle loop is j, which does not appear in the indexing of A. So row zero will be repeatedly scanned as j is varied. Since the cache capacity is large enough to hold N elements (in N/B consecutive sets; so no possibility of conflict misses), no additional cache misses occur for j values 1, 2, ... N-1. So the table entry for j is 1, meaning that the total cost for executing all iterations of j is just one times the cost already determined for running through all iterations of the inner-most loop. As the outermost loop (i) is varied, for each distinct value of i, different rows of A are accessed, and we have a repeat of the number of misses corresponding to i=0. Hence the table entry has a multiplier value of N. The total number of misses for A is N*1*(N/B).

Loop	A	B	C
i	N	N	N
j	1	N	N/B
k	N/B	N	1
Total	N^2/B	N^3	N^2/B

Direct-mapped cache

Loop	A	B	C
i	N	N	N
j	1	N/B	N/B
k	N/B	N	1
Total	N^2/B	N^3/B	N^2/B

Fully associative cache

Figure 1: Cache miss analysis for **ijk** form

Analysis of *ikj* form

```
for (i=0; i<N; i++)
 for (k=0; k<N; k++)
  for (j=0; j<N; j++)
   C[i][j] += A[i][k]*B[k][j];
```

Array A: It 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: It 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
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.

Loop	A	B	C
i	N	N	N
k	N/B	N	1
j	1	N/B	N/B
Total	N^2/B	N^3/B	N^2/B

Direct-mapped cache

Loop	A	B	C
i	N	N	N
j	N/B	N	1
k	1	N/B	N/B
Total	N^2/B	N^3/B	N^2/B

Fully associative cache

Figure 2: Cache miss analysis for **ikj** form

Analysis of *jik* form

```
for (j=0; j<N; j++)
 for (i=0; i<N; i++)
  for (k=0; k<N; k++)
   C[i][j] += A[i][k]*B[k][j];
```

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 associative cache.
Array B: It is accessed by column in the innermost loop, resulting in N misses for either type of cache. The middle loop is the i-loop and i does not index B. We have repeated access of the same column j. For a direct mapped cache, each iteration of i will result in repeated misses, but for a fully associative cache we will only have compulsory cold misses for i=0 and all hits for later iterations of the i loop (as long as the capacity is at least N blocks or N*B words). The outer loop j is the column index in B (B[k][j]), causing different columns to be accesses for different iterations. With a direct mapped cache, no spatial reuse is exploited due to conflict misses in accessing the elements of each column. Hence the miss cost is repeated for each distinct value of j, i.e., the table entry is N. But for a fully associative cache, the column access does not cause any evictions and therefore, as j is changed from 0 to 1, no additional misses will occur as the ik loops are fully traversed. But once every B iterations of j we will encounter new data elements during the innermost loop's column access. Therefore, the multiplier for loop j is N/B. So the total number of misses for a direct-mapped cache will be N*N*N, but only N*N/B for a fully associative cache.
Array C: There will be complete temporal reuse since the inner loop 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 direct-mapped cache will be N*N (one miss for each element), while it will only be N*N/B for a fully associative cache.

Loop	A	B	C
j	N	N	N
i	N	N	N
k	N/B	N	1
Total	N^3/B	N^3	N^2

Direct-mapped cache

Loop	A	B	C
j	N	N/B	N/B
i	N	1	N
k	N/B	N	1
Total	N^3/B	N^2/B	N^2/B

Fully associative cache

Figure 3: Cache miss analysis for **jik** form