Cache Miss Estimation for Matrix Multiplication

Example 1

Consider a cache of size 64K words and block of size 8 words, and arrays of dimension 512×512. The arrays are stored in row-major order. Perform cache miss analysis for the ijk form of matrix multiplication considering (i) direct-mapped, and (ii) fully-associative caches. To simplify the analysis, ignore misses from 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, a quarter of the 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. If the inner loop accesses an array by column, by the time half the columns are 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 about the number of 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 the block size is 8 words, A[0][0]...A[0][7] will occupy memory block 0, and map to the cache set 0. Elements A[0][8]...A[0][15] map to the 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 64 blocks, A[2][0] must map to the set $2 \times 64 = 128$. Generalizing, element A[k][0] will map to set [$(k \times 64)$mod 8K], since the number of sets in a cache with a capacity of 64K words and block size 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 $k \times 64$ 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[1][0]...A[1][7].

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];
```

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

(a) 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

(b) Fully-associative cache

Table 1: Cache miss analysis for the ijk form.

Array A: For a fixed i and j, as k is varied, row i will be repeatedly accessed, resulting in N/B cold misses (one miss for every B accesses). As j is varied, the same row will be accessed in cache, resulting in hits. Similar costs are incurred 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, subsequent 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 summary of the cache miss analysis results is shown in Table 1. Each entry in the table represents a multiplier, for the associated 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 there is no possibility of conflict misses, and there are 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).