Systems for Data Science - HW1

Shubham Shetty

February 8 2021

#	**Order**	**Execution Time**
1	i k j	3.026624 s
2	k i j	3.032376 s
3	j i k	8.338146 s
4	i j k	9.566060 s
5	j k i	18.476502 s
6	k j i	19.271500 s

Table 1: Execution times for different i, j, k orders

The above table shows the execution time for matrix multiplication from the given code for different i, j, k combinations in ascending order of time. The variable i represents the row of matrix, j represents column, and k acts as an iterator across row and column.

The matrix stored in memory can be visualised as multiple cache lines, with each cache line storing a single row of the matrix. Pulling a cache line from memory is time consuming, while calculating the product of 2 numbers is not. Hence the most optimized algorithm would have the fewest number of cache line pulls. This is the concept of spatial locality, where memory which is stored closer to each other (i.e. in the same cache lines) is faster to access.

For the combination {i, j, k}, k is the innermost iterator. Hence while calculating C[i][j] += A[i][k] * B[k][j], there would be a new cache line pull for each new value of B[k][j], which is not the best implementation as it corresponds to 3 cache line pulls per element in final matrix. Same is the case for the combination {j, i, k}.

For the combination {j, k, i}, i is the innermost iterator. Hence while calculating C[i][j] += A[i][k] * B[k][j], there would be a new cache line pull for each new value of A[i][k], which is the worst implementation as there would be 9 cache line pulls per element in final matrix. Same is the case for the combination {k, j, i}.

The combination of {i, k, j} & {k, i, j} has the best execution time as it has j as the innermost iterators. This ensures that there is only a single cache line pull to calculate values of each row before moving on to the next. These combinations present the best optimised algorithm by using spatial locality of reference.

To check the improvement in performance between the original algorithm and the optimised algorithm, we can check the speed-up (S) between them -

$$S_{time} = \frac{T_{i,j,k}}{T_{i,k,j}} = \frac{9.566060}{3.026624} = 3.16 \approx 3$$