Lab 2 Report

Andres Calderon

acald013@ucr.edu

October 18, 2016

1 Part 1

1.1 Matrix 10000x10000

1. **ijk and jik**
   a) Matrix A: The pattern access for this matrix is row-wise. As the line size is 10, every 10^{th} element will be a cache miss but the remaining elements will have a free ride and they will be on cache when they will be requested. So, each 10^{th} element will have n (10000) cache misses, the remaining element will have 0.
   b) Matrix B: As this matrix is accessed in column-wise fashion all its elements will not be on cache. The line size is too short to load the next element, so each element access will count as a cache miss. Indeed the number of cache misses for each element is n (10000).
   c) Matrix C: This matrix is accessed in a fixed-position pattern. Each element is read just one time but after that the cache has to read all the element from matrices A and B. The cache size is too short to load all those elements and keep the previous C element. So each element in matrix C will count 1 read cache miss.
2. **kij and ikj**
   a) Matrix A: For these cases matrix A follows a fixed-position pattern access. Each element is read just one time but as cache size is too short for keeping track of elements for the inner loop, each access will be a miss on cache. The number of read cache misses for each element of A is 1.
   b) Matrix B: Matrix B follows a row-wise access pattern, so for a line size of 10 just every 10^{th} element will be a cache miss. The number of cache misses for every 10^{th} element is n (10000) and 0 for the remaining elements.

c) Matrix C: The patter access for Matrix C is similar than previous for Matrix B. Just every 10^{th} element will be a cache miss. So, the number of cache misses for every 10^{th} element is n (10000) and 0 for the remaining elements.

3. **jki and kji**

a) Matrix A: Matrix A has a column-wise access pattern in these cases. As it was mentioned before, the line size is too short to prevent cache misses and every access will count as one. The number of read cache misses of every element of A is n (10000).

b) Matrix B: This matrix is accessed in a fixed-position fashion. Each element is read just one time but, as the cache size is too short, every of them will count as a cache miss. The total number of read cache misses is 1 for each element.

c) Matrix C: Similar that matrix A, this matrix as a column-wise access pattern. So, the number of read cache misses of every element of C is n (10000).

1.2 Matrix 10x10

Given the size of the line and the cache, the three matrices will fit on the cache at the same time. The pattern access will not have effect such there will not be cache misses. However, it has to be noted that the first access will not be on cache. Given the line size of 10, the first every 10^{th} element load will count as a miss. In conclusion, for each algorithm, for each matrix, the 10^{th} element will have 1 read cache miss, the initial access to the cache, and all the remaining element will have 0.

2 Part 2

2.1 Matrix 10000x10000

Note that using block of 10x10 will allow to fit blocks from each matrix on cache without troubles. Given the settings each block reading will cost just 10 read misses for the initial load. After that, all the access in the inner loops will have a free ride. So, this algorithm will iterate the matrices block by block in similar fashion as in the previous part. The difference will be than the number of iterations will be reduced by the size of the block. The number iterations will be given by $\frac{n}{b}$, where is the size of the block. In our particular case, with a block size of 10, the algorithm will perform $\frac{10000}{10} = 1000$ iterations.

1. **ijk and jik**

a) Matrix A: This matrix follow a row-wise access pattern similar that in previous part. Given the line size of 10, every 10^{th} element it will have a new load to the cache. However, thanks to blocking the number of read cache misses for each 10^{th} element is reduced to $\frac{n}{b} = \frac{10000}{10} = 1000$. The remaining elements will count 0 read cache misses.

b) Matrix B: As this matrix follow a column-wise access pattern each of the elements will count as a read cache miss. Even the use of the blocks, the line size is still too short. However, similar than in the previous case, the number of read cache misses is reduced by the size of the block. This matrix will count 1000 read cache misses for each element.

c) Matrix C: This matris follow a fixed-position access pattern. Each element of the matrix is computed just one time. However, the blocked version algorithm allows to keep blocks of this matrix on cache. As a result, the number of read cache misses is just 1 for every 10^{th} element. The remaining elements will count 0 read cache misses.

2. **kij and ikj**

a) Matrix A: This matrix follows a fixed-position access pattern. The number of read cache misses is similar to matrix C of the previous algorithm. Each 10^{th} element will have 1 read cache miss, the remaining elements 0.

b) Matrix B: This matrix follows a row-wise access pattern equal to the matrix A of the previous algorithm. The number of read cache misses is 1000 in each 10^{th} element. The remaining elements will not count read cache misses.

c) Matrix C: As this matrix also follows a row-wise access pattern, the number of read cache misses is the same that the previous matrix.

3. **jki and kji**

a) Matrix A: This matrix follow a column-wise access pattern. The number of read cache misses is similar to matrix B of the first algorithm. There will be 1000 read cache misses for each element of this matrix.

b) Matrix B: This matrix follows a fixed-position access pattern. The number of read cache misses is similar to matrix C of the first algorithm. There will be 1 read cache miss for each 10^{th} element. The remaining elements will not have any misses.

c) Matrix C: As this matrix follows the same access pattern that matrix A of this algorithm, the number of read cache misses will be the same (1000 read cache misses for each element).

3 Part 3

4 Part 4

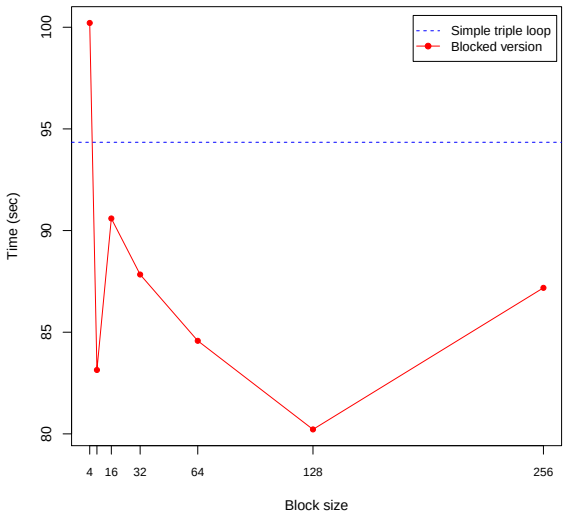

Figure 1: Matrix multiplication performance.

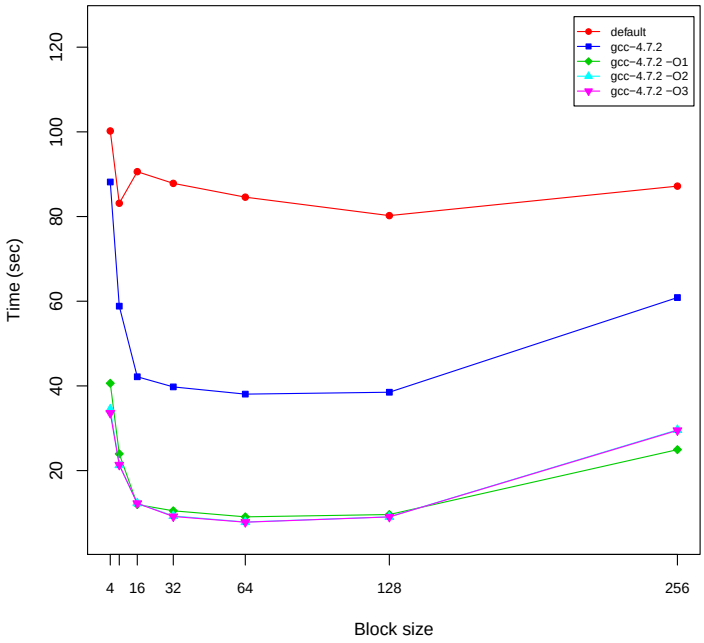

Figure 2: Matrix multiplication performance.