PARALEL COMPUTING PROGRAMMING ASSINGMENT

Recep Tayyip Erdoğan
20170808039

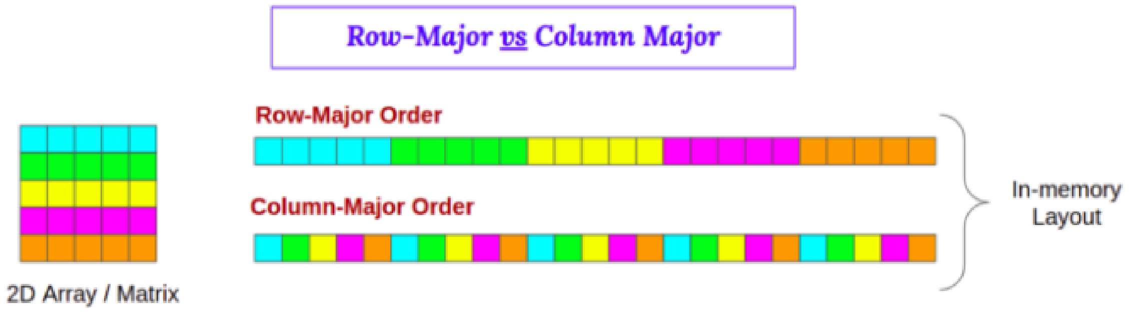
02 March 2022

Akdeniz University
Parallel Computing

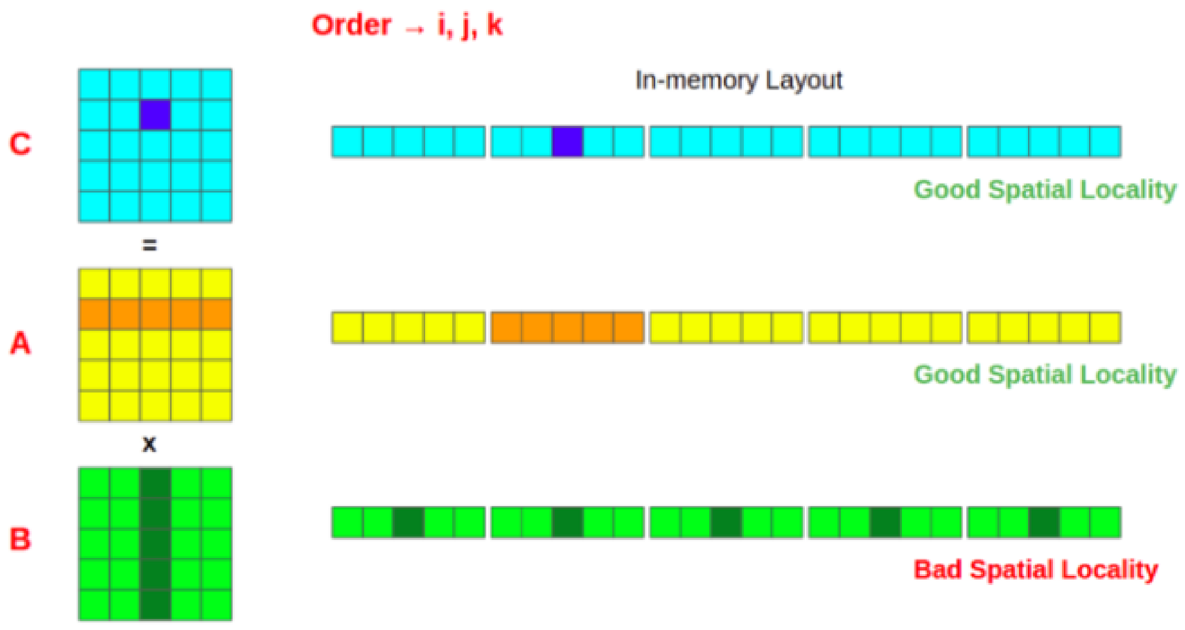
1. Problem and Solution Strategy

a. Problem:

In C programming language, two-dimensional arrays are stored in RAM as row-major (Fortran language stores it as column major).



The data to be processed is loaded from the RAM to the cache in the CPU. Since the size of the Cache in the CPU is very small compared to the RAM, the size of the data received from the RAM during the process is the size of the Cache. When the CPU needs data to perform a certain operation, the system checks whether that data is in the cache. (If the data is in the Cache, we call it CACHE HIT, if not, CACHE MISS.) If the data to be processed is not in the Cache, this data must be loaded from RAM to the Cache. If the data is in the Cache, this means a much faster data access than loading the data from the RAM into the Cache.



Let's assume that the order of our loops in standard matrix multiplication is i j and k, as in the picture above. Since the data to be processed is row-major due to the nature of the C programming language, our Cache Miss value in the A matrix will be very low because we will get the data to be processed from the A matrix in row-major order. However, when doing matrix multiplication, when we take the first matrix as row-major, we need to take the second matrix as column major (according to the matrix multiplication rule).

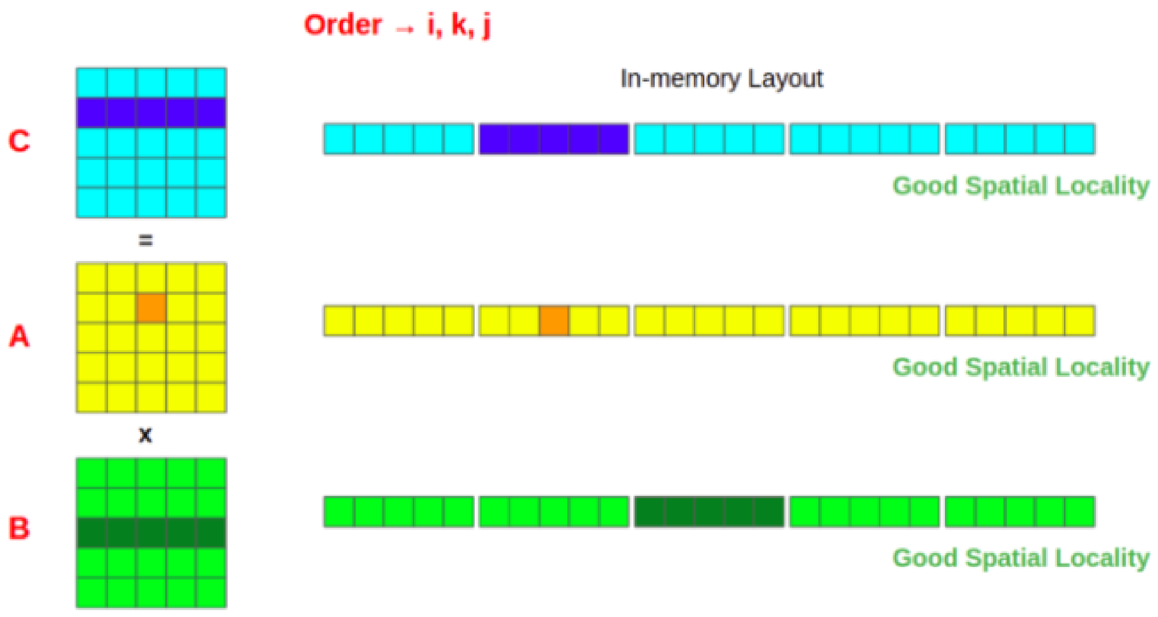
We need to take the values in the B Matrix as Column major, in this situation the CACHE MISS RATE increases a lot when multiplying the matrix. As we have just mentioned, the data to be processed in the C programming language is taken as row-major.

As a result, since the CPU will not found the data in the Cache required data for processing, data have to loaded from ram into the Cache continuously.

This situation causes a lot of time wasted during the process.

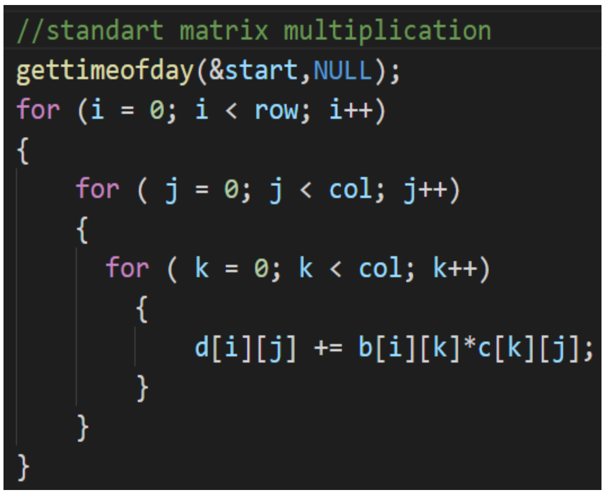
b. Solution

To solve the cache miss rate situation, we change the order of the for loops we use during matrix multiplication. Now let's examine how the matrix multiplication operation has an effect on cache and memory during the i, k and j cycles.

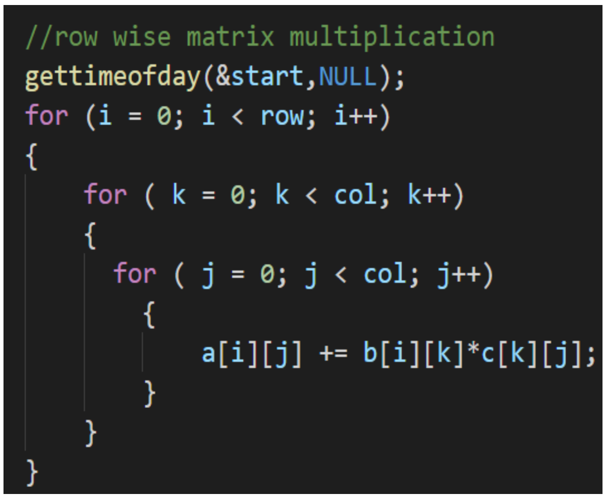


When we rearrange the for loop in this way, the data that the CPU will use will be in the Cache friendly state. As a result, our Cache Hit value will increase a lot and this will reduce the amount of data we want from RAM.

MY CODE:



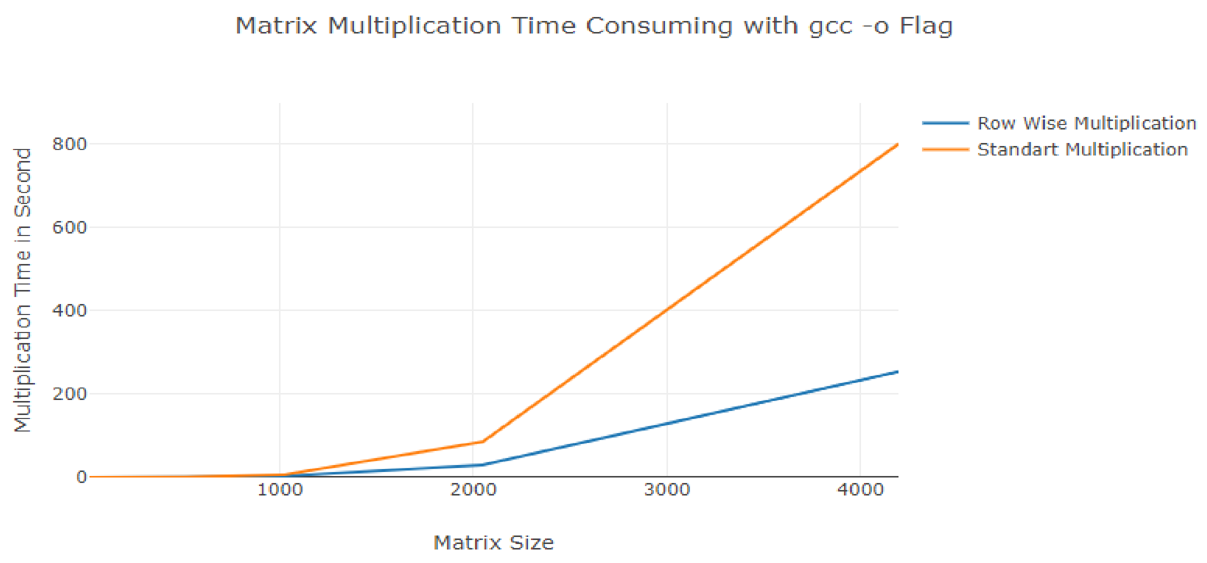
```
//standart matrix multiplication
gettimeofday(&start,NULL);
for (i = 0; i < row; i++)
{
    for ( j = 0; j < col; j++)
    {
      for ( k = 0; k < col; k++)
        {
            d[i][j] += b[i][k]*c[k][j];
        }
    }
}
```



```
//row wise matrix multiplication
gettimeofday(&start,NULL);
for (i = 0; i < row; i++)
{
    for ( k = 0; k < col; k++)
    {
      for ( j = 0; j < col; j++)
        {
            a[i][j] += b[i][k]*c[k][j];
        }
    }
}
```

2. Data Visualizations

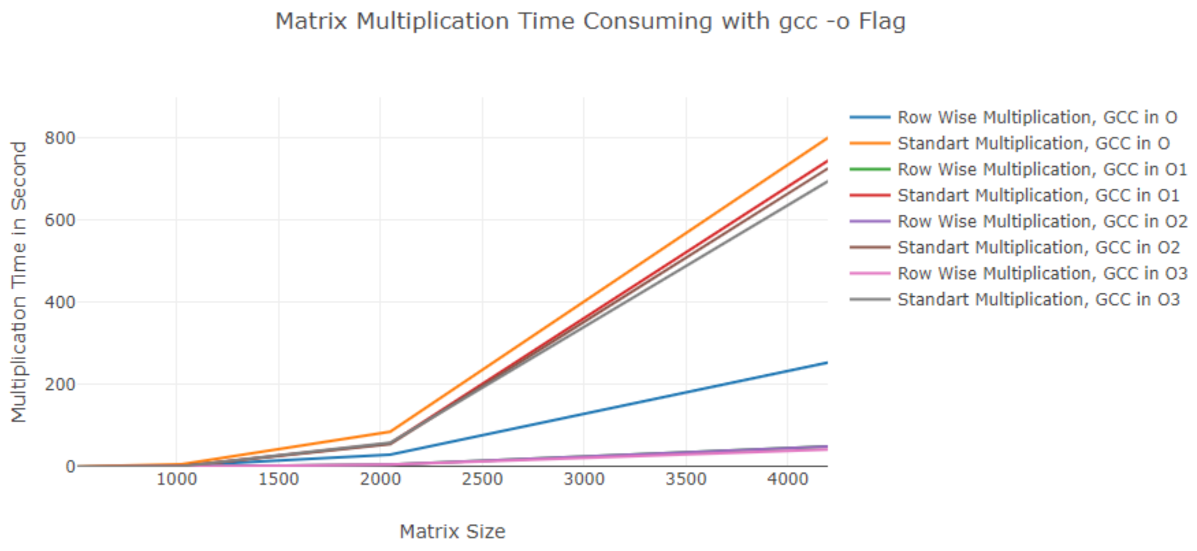
The following data are results obtained with normal execution of C code:



Matrix Size	Row Wise Multiplication Time in Second	Standart Multiplication in Second
16	0	0
32	0.0001	0.0001
64	0.0009	0.0009
128	0.0078	0.0082
256	0.0624	0.0715
512	0.4567	0.6177
1024	3.7214	6.8506
2048	29.9308	85.5447
4196	254.3844	801.4539

The following data are results obtained with different GCC flag execution of C code:

Matrix Size	Row Wise Multiplication Time in Second O1	Standart Multiplication in Second O1	Row Wise Multiplication Time in Second O2	Standart Multiplication in Second O2	Row Wise Multiplication Time in Second O3	Standart Multiplication in Second O3
512	0.0772	0.1038	0.0785	0.1071	0.0742	0.1037
1024	0.9109	1.572	0.6309	1.4844	0.6458	1.5645
2048	5.6076	57.5213	6.0299	56.0582	5.6088	58.8549
4196	49.614	745.6761	49.7354	726.695	42.5439	695.5012



3. **Conclusion:**
   The data we have clearly shows that if we write a Cache friendly code, we can achieve the result we want to achieve much faster. In addition, learning how the programming language we write works on computer hardware allows us to get a much faster solution in that programming language. If we did not know how the C programming language works with memory, we would not be able to get a faster solution.

4. **References:**

   a. https://www.rapidtables.com/code/linux/gcc/gcc-o.html
   b. https://levelup.gitconnected.com/c-programming-hacks-4-matrix-multiplication-are-we-doing-it-right-21a9f1cbf53
   c. https://cs.brown.edu/courses/cs033/lecture/18cacheX.pdf

Note: I used HTML and JavaScript for Creating Graphics and I attached to the files that I have send.