Using Blocking to Initialize a Matrix with bigger dimensions in lesser time

Venkatesh Prabhanjan

Abstract

This Project is designed to have a greater understanding about the spatial arrangements and temporal locality. The objective is to improve the initialization code such that execution time is minimal for the initialization of the matrix and to write an optimal code that initializes and transposes the same matrix. Accessing the matrix row wise is faster than accessing the matrix column wise and using blocking method after finding the optimal block size contributes towards a good execution time.

Introduction

While initializing a matrix, specially of higher dimensions the way we access the elements of matrix are important in terms of efficiency. In the given situation we have two kinds of initialization of matrix Row wise and Column wise. The Row wise initialization is more efficient for the fact that it will access the adjacent elements successively and makes the best use of the contents of block. However conversely the Column wise initialization would not access the adjacent elements successively and hence this is not making efficient use of the block brought from memory. For us to get efficient initialization of matrix we will need to make Row wise initialization. Spatial locality is a concept in which a block when moved to cache is completely used and this is turn makes no need to revisit a block once used. To achieve Spatial locality while transposing a matrix we need to use a concept called blocking, and need to determine the optimal block size for the same.

Part I  
After executing the modified the program for 5 times, we observe the following:  
-The row wise execution time is greatly lesser than the column wise execution time  
-The column wise execution time remains the same through all 7 tests  
-The row wise execution time reduces as we iterate through tests  
-The column wise execution time remains the same because every time we initialize matrix elements are accessed column wise meaning the content brought from Primary memory is not accessed efficiently.  
-The row wise execution time reduces because it will access the adjacent elements successively and make the best use of the content brought from primary memory.

Part II  
Strategy to minimize the execution time of program:  
-To minimize the execution time of initialization of matrix we will have to initialize the matrix row wise, as accessing the matrix row wise is quicker than accessing the matrix column wise for the fact that row wise initialization will access the adjacent elements successively

-To minimize the execution time of initialization and transposition of a matrix we will need to use a concept called blocking, in which we virtually create blocks and iterate through the blocks and not the dimension of the matrix, a block is accessed at once and elements are completely used(transposed). This method will help us reduce the execution time greatly.  
-Optimal block size is when we are able to store the elements of an entire row or an entire column(Transpose) in a block and hence we choose a block size of 40000.

-with having an optimal block size (40000) we will be able to transpose elements per block and we will not be getting the whole matrix into the cache which makes it difficult (takes longer time) to access the elements, but just the elements in a block which is quite lesser than all the elements in the matrix.

Collection and analysis of result and how they match our actions:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Test 1** | **Test 2** | **Test 3** | **Test 4** | **Test 5** | **Test 6** | **Test 7** |
| **Maximum(seconds)** | 8.55 | 8.55 | 8.55 | 8.55 | 8.55 | 8.55 | 8.55 |
| **Minimum(seconds)** | 8.55 | 5.39 | 5.39 | 5.39 | 5.39 | 5.37 | 5.37 |
| **Average(seconds)** | 8.55 | 6.97 | 6.64 | 6.33 | 6.21 | 6.07 | 5.97 |

-Initially, first iteration to initialize and transpose a matrix system takes about 8.55 seconds as it must get the content from primary memory to cache.  
-Since we have used the concept of blocking to access elements of a huge matrix, the elements are accessed in blocks and due to spatial locality of elements, the hit ratio will increase when looking for an element therefore the execution time reduces.

-Please find above the report of execution time variation as we iterate through the matrix using the concept blocking.

Conclusion

In conclusion, accessing a matrix with higher dimensions iterating through all the elements and if the access order is not row wise i.e., if the adjacent elements are not accessed successively the process of iterating takes a lot of time. Hence pattern of accessing the elements of a matrix with huge dimension is important. However, when we need to transpose a matrix with bigger dimension we will have to access elements that are not adjacent to the current element quite every other time and hence execution time will increase. In order to reduce the execution time of accessing the matrix even while we transpose a matrix with such big dimension we use a concept of blocking in which we divide the matrix into smaller blocks and access each block at a time, perform transpose with the elements of the block and not revisit the block anytime again, This will help us reduce the execution time greatly.