Benchmark of matrix multiplication 3

Abstract

Code optimization allows consumption of fewer resources. (i.e. CPU, Memory), which results in faster running machine code. Optimized code also uses memory efficiently. The optimization must be correct, it must not, in any way, change the meaning of the program.

I'll try various ways of optimizing matrix multiplication with random values, then we will implement and test a way of compressing matrixes into CRS and CSS to try and multiply them while compressed.

Introduction

I'll optimize the previous matrix multiplication code to improve the use of the resources and the speed of execution.

To optimize the matrix multiplication code, I though about 2 possible options:

-Tiled matrix multiplication

-Transposed matrix multiplication

After reading about both, I found more interesting Tiled matrix function, so I implemented this one into Java.

I'll also implement the compression of matrixes into CRS, CSS and multiplication of compressed matrix in CSS and CRS mode.

I will be doing running time calculation on an AMD Ryzen 5 5600H.

Experiments

First, I created random matrixes with 2000x2000 dimension that will be used to compress, multiply after being compressed and try the new matrix multiplication methods such as tiled multiplication and the use of threads.

I created a SparseMatrix class that will be used to create the compressed matrix.

I started testing the random 2000x2000 matrix, which compressed in just 12 milliseconds.

Once the matrix was compressed into CRS and CSS, I proceeded to multiply them (CRS matrix x CSS matrix) and check running time.

Random matrix compressed multiplication: 64 milliseconds

As we can see it makes the multiplication really fast, even faster than threaded tiled multiplication, as we will see later.

Later I compressed and multiplied the test matrix, which was imported from a MTX file and has 525825 rows and columns. This matrix has much bigger dimensions and will obviously take longer to compress and multiply.

TEST matrix compression by rows: 34.556 seconds

TEST matrix compression by columns: 34.568 seconds

TEST compressed multiplication: 6525.595 seconds (108.75 minutes)

As we can see it would probably take too long to make this multiplication in a normal way, since it has such a high number of values, even though it has a big percentage of sparsity.

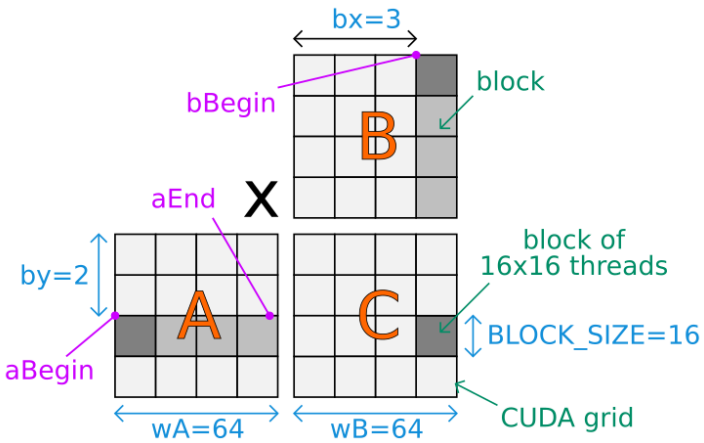
Lastly, I will try the new implemented ways of matrix multiplication to check running time.

First, we check the normal multiplication time to compare it to the new ways implemented,

Normal Multiplication running time: 42.532 seconds

As we can see it takes way too long to multiply with the normal way, this could be improved by a lot by optimizing the code by not only multiplying in a different way but also using more of the processor power by making use of threads.

Now we'll check the speed of Tiled matrix multiplication, which works this way:



Tiled Multiplication running time: 28.489 seconds.

As we can see Tiled multiplication makes a 32% running time improvement over the normal multiplication, which it's an impressive upgrade since we are only changing the way of multiplying the matrixes.

Lastly, I did the same Tiled multiplication but this time with 20 threads. This made an amazing running time improvement; however, it obviously makes use of more CPU.

Tiled multiplication with threads running time: 3.047 seconds.

However, these implementations weren't as good as multiplying them after being compressed, as we saw at the start of the experiment, taking them under 90 milliseconds to be compressed and multiplied.

Conclusion

Compressing matrixes when they have big sparsity is very important if they size of them is very big.

It also seems to improve the running time of the multiplication, since it was way faster done when compressed than making use of 20 threads in a tiled matrix multiplication.

Lastly, matrixes with big dimensions such as the test matrix, are way too big to be used without compressing them.