Programming assignement 1

1. Availability

The code is available in https://github.com/HegePI/cloud-and-edge-computing/tree/master/pa1. Repository contains both python files for matrix multiplications and plotting the A matrix. Repository also has produced .png files.

2. Programming Language

I chose python for programming language because I found it is easy to implement the exercise with it. Other Programmign languages I tried included Rust and C, but I found it to be too much overhead to try to implement the exercise with those.

3. Methodology

Matrices were created with uniform function, which is in numpy's random module https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html. In documentation it says that values are sampled such that values are equal or greater to low and values are lower than high. Because value range is defined as (0,1), low is set to 0.0000001.

The problem in the exercise is that multiplying matrix A and Matrix B with dimensions A=(10^6,10^3) and B=(10^3,10^6) results to matrix with dimensions of AB=(10^6,10^6). For any normal computer storing matrix with this size takes a lot of space, almost 8Tb. So other solution was needed.

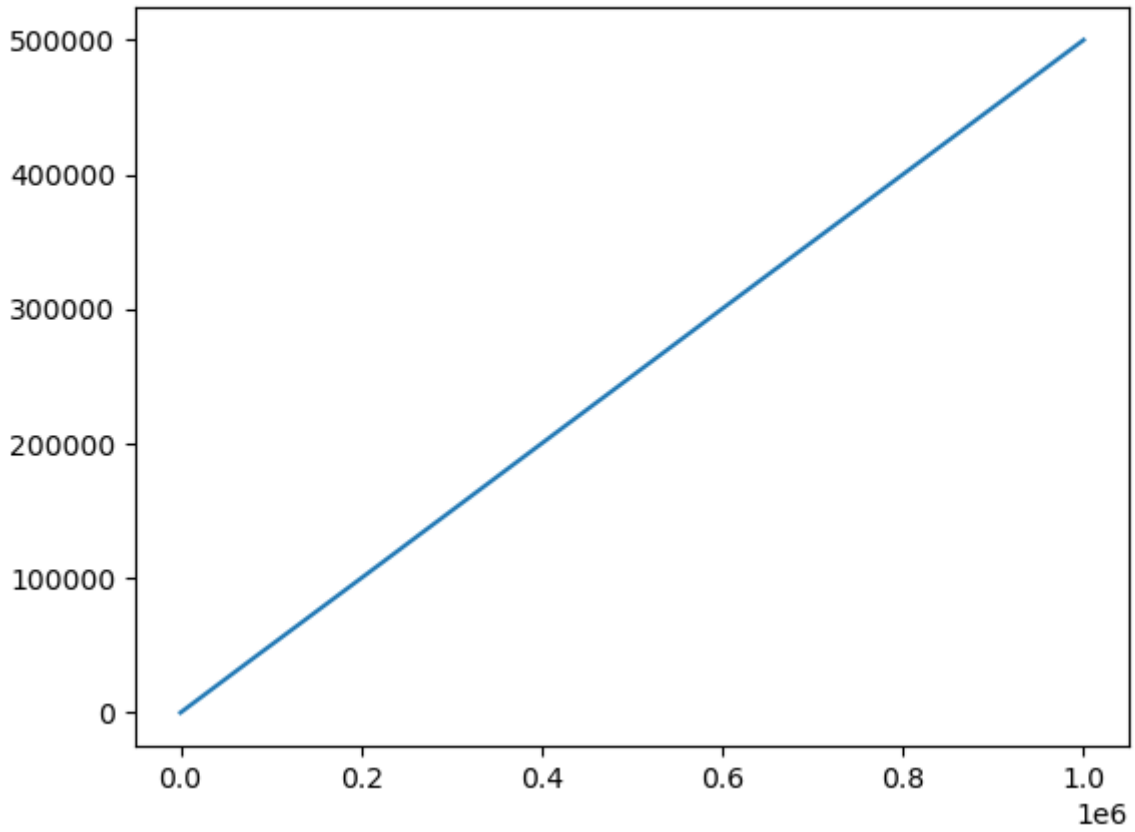
```
cloud-and-edge-computing on  master [!] via  v22.11.1 at ⎈  minikube (default)
➜ ipython
Python 3.8.13 (default, Oct 21 2022, 23:50:54)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.8.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import numpy as np

In [2]: A = np.random.uniform(0,1,(1000000,1000000))
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
Cell In[2], line 1
----> 1 A = np.random.uniform(0,1,(1000000,1000000))

File mtrand.pyx:1093, in numpy.random.mtrand.RandomState.uniform()

File _common.pyx:558, in numpy.random._common.cont()

MemoryError: Unable to allocate 7.28 TiB for an array with shape (1000000, 1000000) and data type float64

In [3]:
```

The solution was to use rule of associativity in matrix multiplication https://en.wikipedia.org/wiki/Matrix_multiplication#Associativity. So instead of calculating (AxB)xC = D we can calculate Ax(BxC) = D. This way the matricies in the calculation are only of size BC=(10^3,1) and that is multiplied with A, (10^6,10^3)x(10^3,1)=(10^6,1). This requires significantly less memory, and therefore can be easily calculated.

To measure the memory usage and CPU usage I used psrecord from the terminal and used its --plot plot.png argument to plot the results into plot.png. The command to record the usage is `python pa1.py & psrecord $(pgrep -f "python pa1.py") --plot plot.png`. Psrecord needs the PID of the process it's recording, so pgrep is used to get the PID of started python process.

4. Dataset

Plotting the matrix A is done in separate file cdf.py. Matrix A is created with seed in both pa1.py and cdf.py files, so that matrix A is same in both of them. Because plotting takes also so much memory only every 1000th value in matrix A is used in the CDF. Ths still gives a good picture about the values of A.

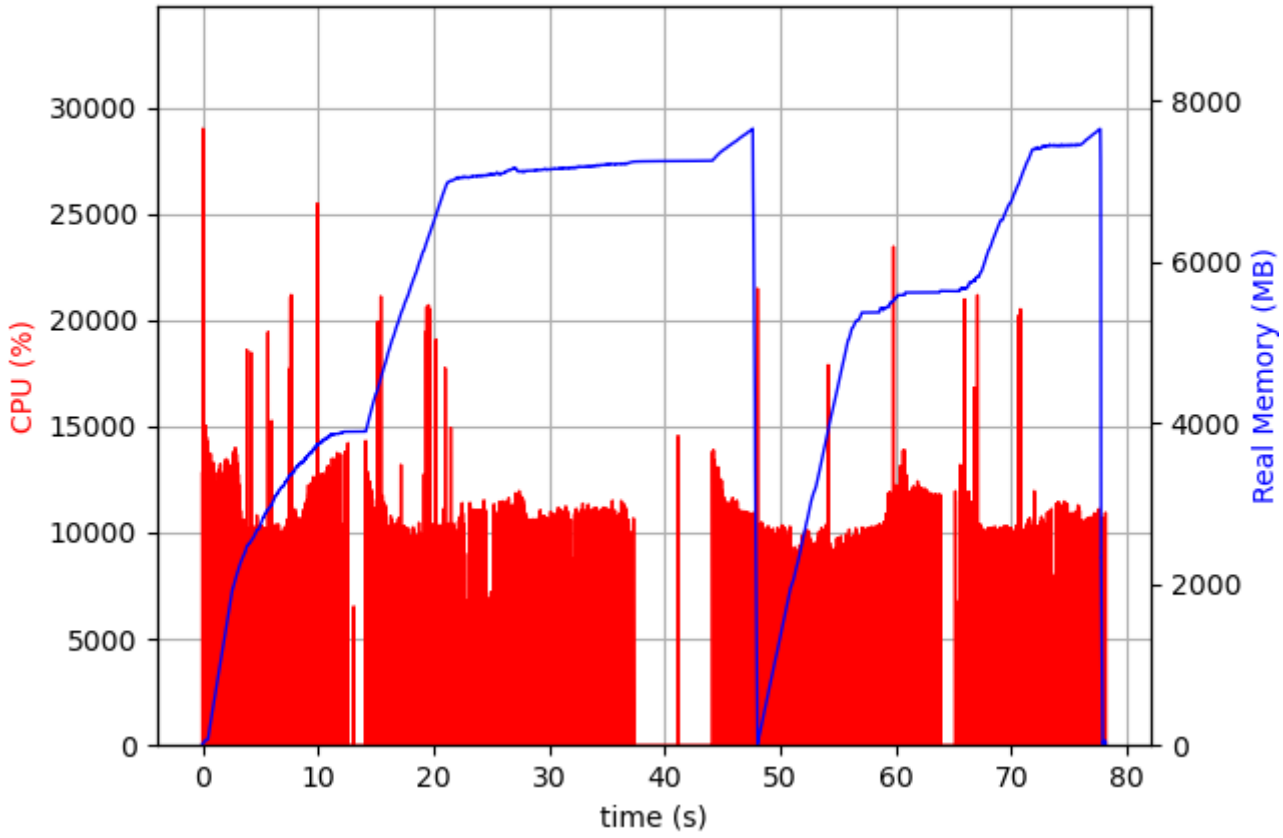


Plotting the CDF plots a straight line, meaning there are close to none outliers in the matrix A. This is expected, when sampling uniformly from range (0,1).

5. Evaluation

Looking at the memory usage (blue line) we can see that in first climb and peak matricies B and C are created and multiplied together resulting to the memory usage of around 8Mb. After the multiplication memory usage drops down to almost 0. Then matrix a is created resulting the second climb. Then multiplying A and BC results to the last climb and after multiplication memory usage drops down to almost 0 again. Plateus in memory usage could be explained as some wait time of the process.

There are some peaks in CPU usage, but otherwisecpu CPU usage is almost the same. Peaks are probably some heavy calculations in matrix multiplication. This could be investigated further.



6. Discussion

First trying to solve this exercise I tried to multiply AxB in batches. Because the memory usage was too high to calculate the matrix, I was trying to calculate first the n first columns of matrix AB and then calculate ABxC with n columns and store results to the C matrix. This solved the memory problem, but this aproach would take way too much time to finish. with batch size of 1000 running in virtual machine I calculated that it would take around 2 weeks for the process to finish.

After releasing to use associativity, matrix multiplication was quite easy. This approach still takes around 8Mb of memory, but most computers can handle that.

Other problem was plotting the CDF of matrix A values. For plotting I used matplotlib pyplot library. Because matrix A contains 10^9 values, pyplot used too much memory to plot that CDF. The solution was to choose every 1000th value from the matrix and plot those values CDF. This CDF is calculated from 10^6 values, so it is still tells about the value distribution of matrix values. Other way to save space for plotting was to delete matricies BC and D after they were calculated.