Importing necessary modules

In [2]:

```
import findspark
findspark.init()
```

In [7]:

```
import numpy as np
from pyspark.mllib.linalg.distributed import RowMatrix
from pyspark.mllib.linalg.distributed import *

from pyspark import SparkContext
from pyspark.sql.session import SparkSession
```

Creating a sparkcontext and sparksession

In [8]:

```
sc = SparkContext("local","PySpark Word Count Exmaple")
spark = SparkSession(sc)
```

Creating 2 random arrays for multiplication

In [4]:

```
A = np.arange(1024 ** 2, dtype=np.float64).reshape(1024, 1024)
B = np.arange(1024 ** 2, dtype=np.float64).reshape(1024, 1024)
```

This function converts the np.array into blockmatrix data type We need to use blockmatrix datatype to access multiplication

In [5]:

```
def as_block_matrix(rdd, rowsPerBlock=1024, colsPerBlock=1024):
    return IndexedRowMatrix(rdd.zipWithIndex().map(lambda xi: IndexedRow(xi[1], xi[0]))).toBlockMat
rix(rowsPerBlock, colsPerBlock)
```

we are converting the A and B to block matrix and computing the product

In [9]:

```
matrixA = as_block_matrix(sc.parallelize(A))
matrixB = as_block_matrix(sc.parallelize(B))
product = matrixA.multiply(matrixB)
```

We need to convert a blockmatrix to localmatrix in order to print

In [13]:

```
print(product.toLocalMatrix())
```

```
DenseMatrix([[3.65967180e+11, 3.65967704e+11, 3.65968227e+11, ...,
              3.66501955e+11, 3.66502479e+11, 3.66503003e+11],
             [9.15186123e+11, 9.15187695e+11, 9.15189267e+11, ...,
              9.16791494e+11, 9.16793066e+11, 9.16794639e+11],
             [1.46440507e+12, 1.46440769e+12, 1.46441031e+12, ...,
              1.46708103e+12, 1.46708365e+12, 1.46708628e+12],
             ...,
             [5.61118508e+14, 5.61119579e+14, 5.61120650e+14, ...,
              5.62212121e+14, 5.62213192e+14, 5.62214264e+14],
             [5.61667727e+14, 5.61668799e+14, 5.61669871e+14, ...,
              5.62762411e+14, 5.62763483e+14, 5.62764555e+14],
```

```
       [5.62216946e+14, 5.62218019e+14, 5.62219092e+14, ...,
        5.63312700e+14, 5.63313774e+14, 5.63314847e+14]])
```

A - matrix

In [14]:

```
print(A)
```

```
[[0.000000e+00 1.000000e+00 2.000000e+00 ... 1.021000e+03 1.022000e+03
  1.023000e+03]
 [1.024000e+03 1.025000e+03 1.026000e+03 ... 2.045000e+03 2.046000e+03
  2.047000e+03]
 [2.048000e+03 2.049000e+03 2.050000e+03 ... 3.069000e+03 3.070000e+03
  3.071000e+03]
 ...
 [1.045504e+06 1.045505e+06 1.045506e+06 ... 1.046525e+06 1.046526e+06
  1.046527e+06]
 [1.046528e+06 1.046529e+06 1.046530e+06 ... 1.047549e+06 1.047550e+06
  1.047551e+06]
 [1.047552e+06 1.047553e+06 1.047554e+06 ... 1.048573e+06 1.048574e+06
  1.048575e+06]]
```

B - matrix

In [15]:

```
print(B)
```

```
[[0.000000e+00 1.000000e+00 2.000000e+00 ... 1.021000e+03 1.022000e+03
  1.023000e+03]
 [1.024000e+03 1.025000e+03 1.026000e+03 ... 2.045000e+03 2.046000e+03
  2.047000e+03]
 [2.048000e+03 2.049000e+03 2.050000e+03 ... 3.069000e+03 3.070000e+03
  3.071000e+03]
 ...
 [1.045504e+06 1.045505e+06 1.045506e+06 ... 1.046525e+06 1.046526e+06
  1.046527e+06]
 [1.046528e+06 1.046529e+06 1.046530e+06 ... 1.047549e+06 1.047550e+06
  1.047551e+06]
 [1.047552e+06 1.047553e+06 1.047554e+06 ... 1.048573e+06 1.048574e+06
  1.048575e+06]]
```

The product when computed using np method

In [16]:

```
A.dot(B)
```

Out[16]:

```
array([[3.65967180e+11, 3.65967704e+11, 3.65968227e+11, ...,
        3.66501955e+11, 3.66502479e+11, 3.66503003e+11],
       [9.15186123e+11, 9.15187695e+11, 9.15189267e+11, ...,
        9.16791494e+11, 9.16793066e+11, 9.16794639e+11],
       [1.46440507e+12, 1.46440769e+12, 1.46441031e+12, ...,
        1.46708103e+12, 1.46708365e+12, 1.46708628e+12],
       ...,
       [5.61118508e+14, 5.61119579e+14, 5.61120650e+14, ...,
        5.62212121e+14, 5.62213192e+14, 5.62214264e+14],
       [5.61667727e+14, 5.61668799e+14, 5.61669871e+14, ...,
        5.62762411e+14, 5.62763483e+14, 5.62764555e+14],
       [5.62216946e+14, 5.62218019e+14, 5.62219092e+14, ...,
        5.63312700e+14, 5.63313774e+14, 5.63314847e+14]])
```

In [ ]: