ABFT Fault Tolerance Design

1. Project Summary

In this project, you will be implementing a program that does matrix multiplication and uses ABFT(algorithm-based fault tolerant) to ensure fault tolerance.

You will implement two versions of this program. Both versions can expect the same input and should give the same output, defined as follows:

Input:

The program will read in two matrices of size (1024x1024). These will be provided as *.csv files.

Output:

It will then calculate the product of these two matrices and print the result to a file. (also in the .csv format)

Version 1

This matrix multiplication program should also use ABFT to provide error detection. It should tell the user if an error is detected and how long it took to detect the error. This will also require you to devise a method to inject an error to detect.

Version 2

Similar to version 1, we want to use ABFT to provide error detection, and the program should tell the user if an error is detected and how long it took to detect the error. However, this version of the program should be done in a distributed manner. After the input files are parsed, the matrix multiplication should be partitioned and distributed among a 4x4 array of containers or virtual machines (or alternatively a simulation tool such as opnet or mininet). Each of those subproblems should be computed also using ABFT checks for matrix multiplication. The results should then be brought back together and the result as a whole should also be checked with ABFT.

2. Core principle for ABFT

In this project, I'm going to use ABFT to provide error detection, it should utilize the ABFT methodology and matrix multiplication to tell the user if an error is detected and how long it took to detect the error.

At first, I have two matrices, I call them A and B, then the product of matrix, I call it C matrix.

There are many ABFT for detect an error in matrix, my solution is to use checksum for each rows and columns of the product of the two original matrices (C matrix).

To get the product of two original matrices, I use "numpy" package in Python which aims to finish matrix operations.

After get the product, I calculate the checksum for each rows and each columns, then store these checksums in two array, one for columns checksum, one for rows checksum.

To inject an error, I designed two random integers between (0~1023), use this to inject an error at the any matrix, for example, if two random integers are 378 and 896, and the fault happened in the C matrix, then the program will change the value of C[378][896] and regard it as an error. Then the program will get the new checksums of C matrix and compare them with old twos, if an error occurs, the two new checksums will not equal to the two old checksum, this is the core thought.

In version 2 program, the core principle apply the same methodology, however, I split two 1024*1024 matrices(A & B matrices) into many(65536) 4*4 matrices. Then dot product of this all small matrices, then I can get the C matrix which has 65535 4*4 matrices, however, comparing with version 1, each value is much more smaller than the version 1, which may increase the speed of the operation.

Below is the figure to show "Row + Column checksum locates and corrects single error", my program do not add checksum before multiplication, but calculate checksum the C matrix.

Figure 1. Row + Column checksum locates and corrects single error

3. Techniques details

3.1 Computer Environment & IDE & Language & Package & Files

Environment: As Figure 2

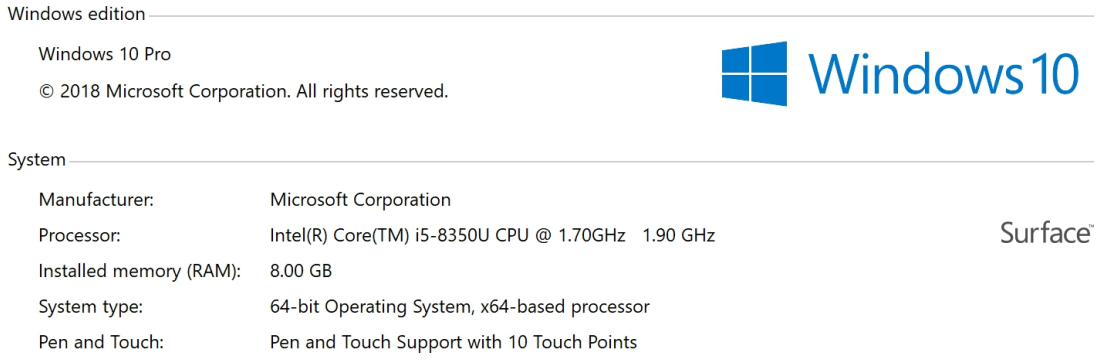


Figure 2. Basic Configuration

IDE:
PyCharm: the Python IDE for Professional Developers
Official Web: https://www.jetbrains.com/pycharm/

Language:
Python 3.8.0
Official Web: https://www.python.org/downloads/release/python-380/

Package:
Numpy, pip & setuptools as figure 3 shows.

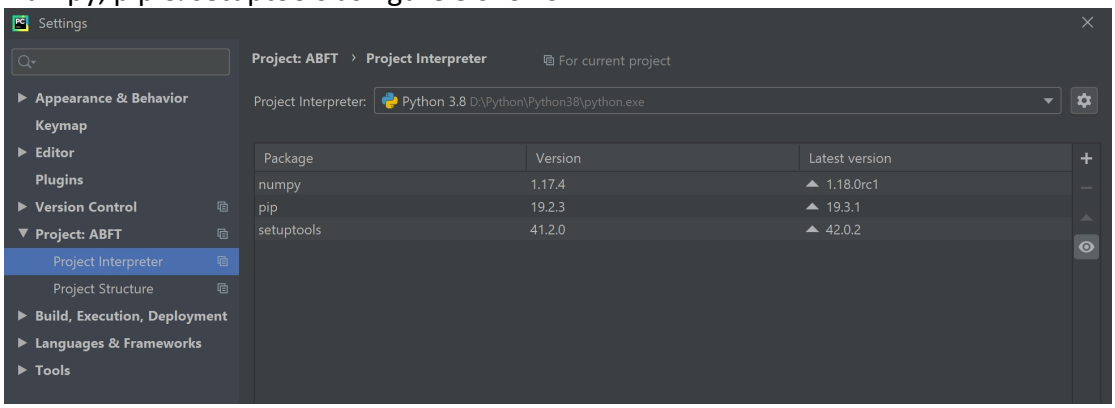


Figure 3. Setting of Interpreter

Files:

I store original csv files in disk C, and the output file will generate and store in disk D, as below:

```
csvfile1 = open("C:\\matrix1.csv","r",encoding='utf-8-sig')
csvfile2 = open("C:\\matrix2.csv","r",encoding='utf-8-sig')
with open("D:\\matrix3.csv","w+") as my_csv:
```

Because disk C is administration disk which requires admin's authorization.

3.2 Version 1

In this part, I try to elaborate my codes in details and even line by line.

My code is written in Python, so at the beginning of my code, I import packages, just as below:

```
import numpy
import random
import csv
import sys
import timeit
```

After that, I offer a question that let user to generate an error or not, if yes, you can choose any matrix or random matrix to generate, as below:

```
generateFault = input("Please select a matrix to generate a fault : 1. A matrix; 2. B matrix. 3. C matrix (The product of the first two matrix) 4. Any matrix 5. Nothing")
if generateFault == "4":
    generateFault = str(random.randint(1,4))
```

Then I start to calculate the time, as below:

```
start = timeit.default_timer()
```

Then I read the *.csv files and store the information in two 2D matrix, as below:

```
firstMatrix = []
csvfile1 = open("C:\\matrix1.csv","r",encoding='utf-8-sig')
reader1 = csv.reader(csvfile1)
for line in reader1:
    eachline = []
    for i in line:
        eachline.append(int(i))
    firstMatrix.append(eachline)

secondMatrix = []
csvfile2 = open("C:\\matrix2.csv","r",encoding='utf-8-sig')
reader2 = csv.reader(csvfile2)
for line in reader2:
    eachline = []
    for i in line:
        eachline.append(int(i))
    secondMatrix.append(eachline)
```

Then I use numpy to transform the data to integer format, as below:

```
x = numpy.array(firstMatrix, dtype='int64')
y = numpy.array(secondMatrix, dtype='int64')
```

Then get the product of two matrices:

```
print ("The product of matrices is : ")
fullMultiplication = numpy.dot(x,y)
print (fullMultiplication)
```

Write the output and generate the *.csv file and store it in disk D:

```
with open("D:\\matrix3.csv","w+") as my_csv:
    csvWriter = csv.writer(my_csv,delimiter=',')
    csvWriter.writerows(fullMultiplication)
```

The next step is to calculate the checksum for each rows and columns of C matrix, as below:

```
checkSumRow = []
for i in range(0,1024):
    sum = 0
    for j in range(0,1024):
```

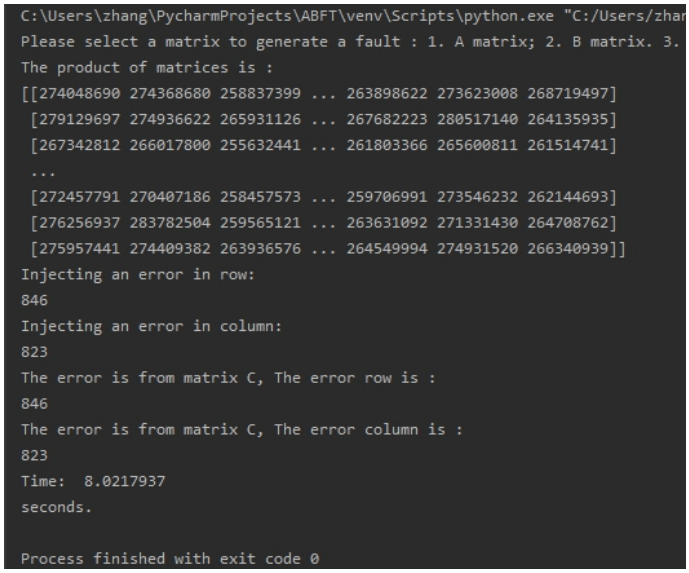
```
        sum += fullMultiplication[i][j]
    checkSumRow.append(sum)

checkSumColumn = []
for i in range(0,1024):
    sum = 0
    for j in range(0,1024):
        sum += fullMultiplication[j][i]
    checkSumColumn.append(sum)
```

Then in inject an Error in C matrix:

```
faultRow = random.randint(0,1023)
faultColumn = random.randint(0,1023)

if generateFault != "5":
    print("Injecting an error in row: ")
    print(faultRow)
    print("Injecting an error in column: ")
    print(faultColumn)

if generateFault == "3":
    fullMultiplication[faultRow][faultColumn] += 5


if generateFault == "1":
    firstMatrix[faultRow][faultColumn] += 5

if generateFault == "2":
    secondMatrix[faultRow][faultColumn] += 5

x = numpy.array(firstMatrix, dtype='int64')
y = numpy.array(secondMatrix, dtype='int64')
fullMultiplication2 = numpy.dot(x,y)
```

Here it generate an Error randomly, and if you want to inject an error in A or B matrix, the value of `firstMatrix[faultRow][faultColumn]` and the `secondMatrix[faultRow][faultColumn]` also changed.

Eventually, I can get the result by comparing the checksum and finally print the information on console panel, including execution time and fault information, like below:



3.3 The result of Version 1

When execute the program, the console panel will show the C matrix(product of two original matrices) and write it to Matrix3.csv in your D disk, then it shows where the error injected, then it will use checksum to detect this error and pinpoint it, as shown in figure 4.

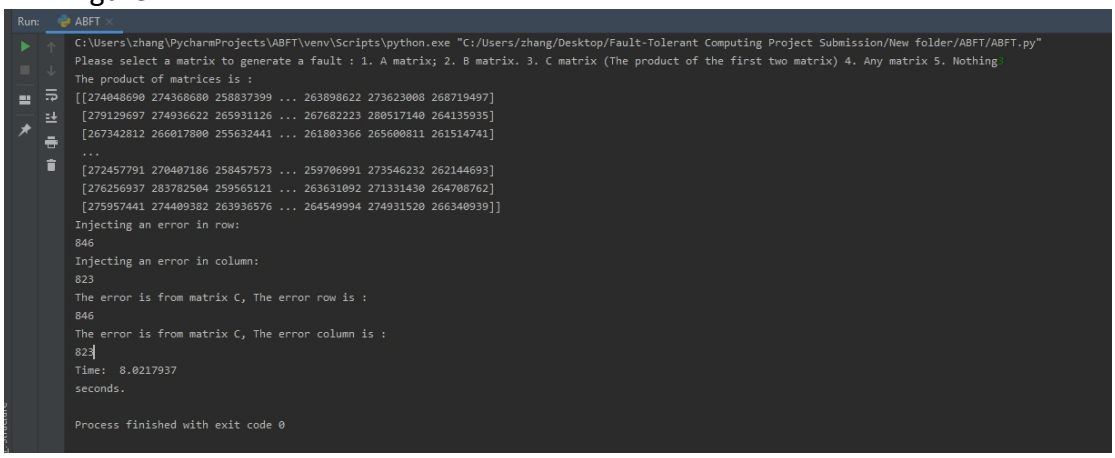


Figure 4. Output of Version 1 Program

The matrix3.csv file is in your D disk, shown as figure 5:

Figure 5. Part of Matrix3.csv file

3.4 Version 2

In this part, version 2 program basic principle also utilize the matrix multiplication, however, it is in distributed manner. So the result of this version has 65536 4*4 matrix, and 65536*4*4 == 1048576 == 1024 * 1024, which match my expect.

Some part of version 2 is similar to the version 1, as below:

(1) Open .csv file and read data from this file.

(2) Dot product for 65536 4*4 matrices from A matrix and 65536 4*4 matrices from B matrix to get 65536 4*4 matrices to C matrix.

(3) Inject an Error to A or B or C matrix.

However, because of its distributed manner, so I need to split the original matrices into smaller matrices, as below:

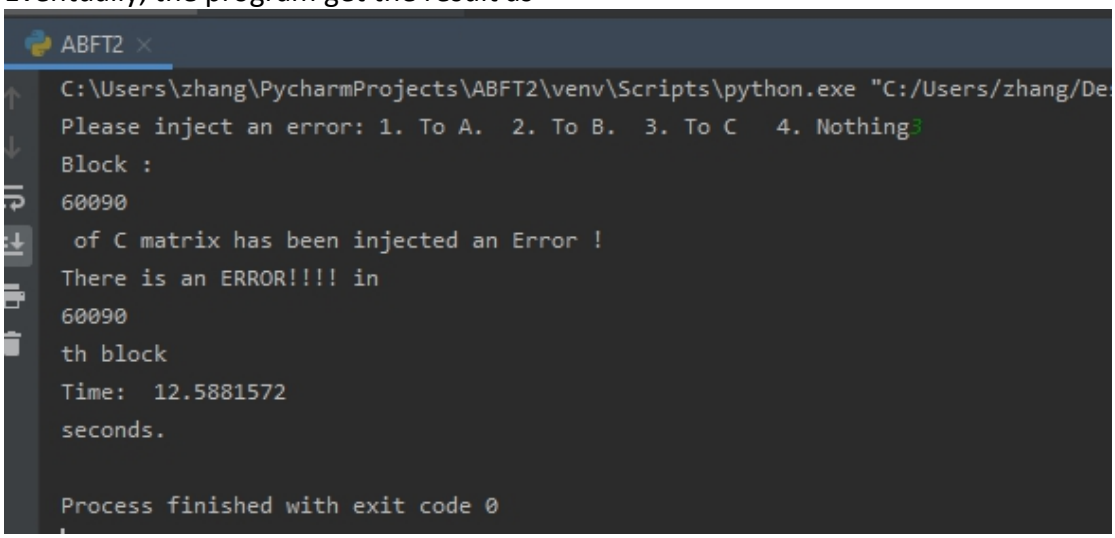
```
block_array = []
previous_row = 0
for row_block in range(256):
    previous_row = row_block * p
```

```
    previous_column = 0
    for column_block in range(256):
        previous_column = column_block * q
        block = x[previous_row:previous_row+p,previous_column:previous_column+q]
        block_array.append(block)

block_array = numpy.array(block_array)
```

And then multiply them to get C:

```
fullMultiplication = []
for i in range(len(block_array)):
    fullMultiplication.append(numpy.dot(block_array[i], block_array2[i]))
```

Eventually, the program get the result as



3.5 The result of Version 2

When execute the program, the console panel will show the C matrix(product of two original matrices) and write it to Matrix4.csv in your D disk, then it shows where the error injected, then it will use checksum to detect this error and pinpoint it, as shown in figure 6.

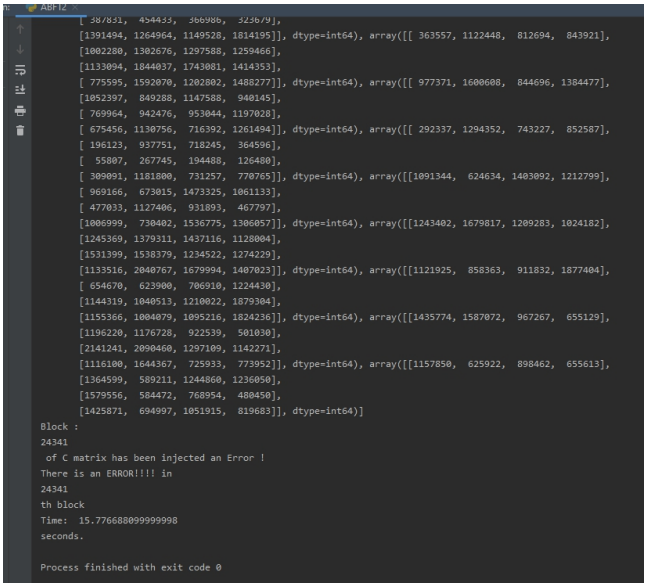
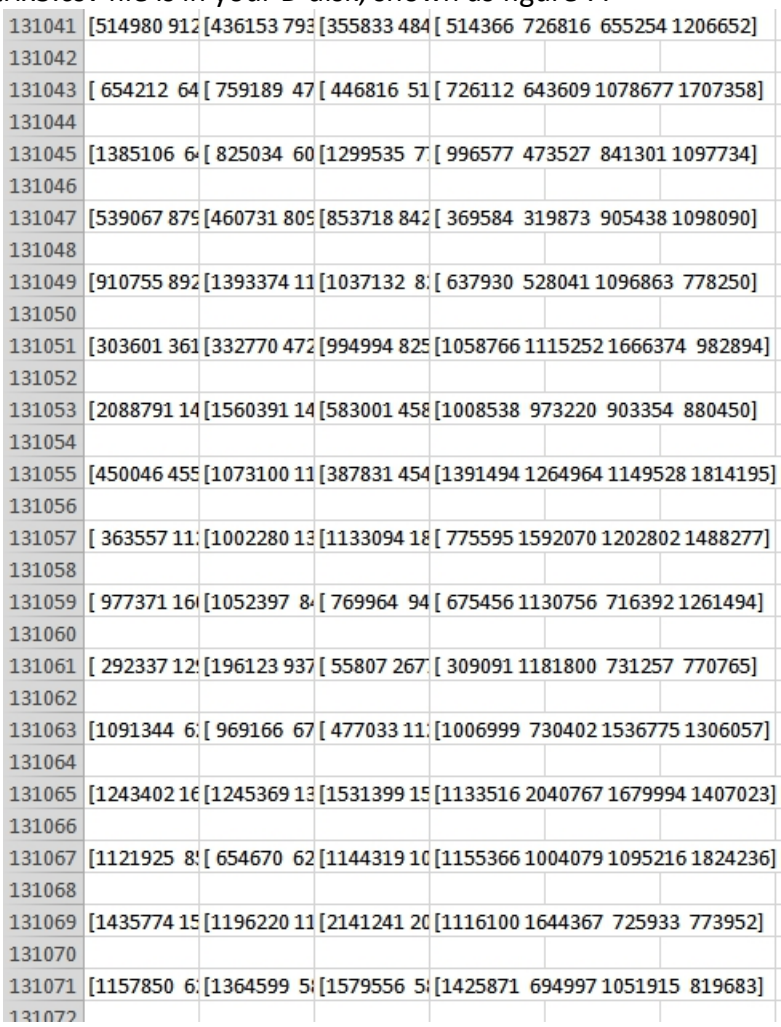


Figure 6. Output of Version 2 Program

The matrix3.csv file is in your D disk, shown as figure 7:



131041	[514980 912 [436153 793 [355833 484 [514366 726816 655254 1206652]
131042	
131043	[654212 64 [759189 47 [446816 51 [726112 643609 1078677 1707358]
131044	
131045	[1385106 6 [825034 60 [1299535 7 [996577 473527 841301 1097734]
131046	
131047	[539067 879 [460731 809 [853718 842 [369584 319873 905438 1098090]
131048	
131049	[910755 892 [1393374 11 [1037132 8 [637930 528041 1096863 778250]
131050	
131051	[303601 361 [332770 472 [994994 825 [1058766 1115252 1666374 982894]
131052	
131053	[2088791 14 [1560391 14 [583001 458 [1008538 973220 903354 880450]
131054	
131055	[450046 455 [1073100 11 [387831 454 [1391494 1264964 1149528 1814195]
131056	
131057	[363557 11 [1002280 13 [1133094 18 [775595 1592070 1202802 1488277]
131058	
131059	[977371 16 [1052397 8 [769964 94 [675456 1130756 716392 1261494]
131060	
131061	[292337 12 [196123 937 [55807 267 [309091 1181800 731257 770765]
131062	
131063	[1091344 6 [969166 67 [477033 11 [1006999 730402 1536775 1306057]
131064	
131065	[1243402 16 [1245369 13 [1531399 15 [1133516 2040767 1679994 1407023]
131066	
131067	[1121925 8 [654670 62 [1144319 10 [1155366 1004079 1095216 1824236]
131068	
131069	[1435774 15 [1196220 11 [2141241 20 [1116100 1644367 725933 773952]
131070	
131071	[1157850 6 [1364599 5 [1579556 5 [1425871 694997 1051915 819683]
131072	

Figure 7. Part of Matrix4.csv file

3.6 Comparison of two versions

Then let's analyze these two versions programs

We can see at matrix3.csv file, each element in the product matrix C is about 250,000,000 to 280,000,000, while in matrix4.csv file, each elements is about 500,000 to 3,000,000. It means that when break the total matrix into pieces, it can largely decrease the arithmetic operation, which can generate a algorithm with lower time complexity(but may cause more space complexity).

Appendix :

1. Version 1 Code

```
import numpy
import random
import csv
import sys
import timeit


generateFault = input("Please select a matrix to generate a fault : 1. A matrix;
2. B matrix. 3. C matrix (The product of the first two matrix) 4. Any matrix 5. Nothing")
if generateFault == "4":
    generateFault = str(random.randint(1,4))

start = timeit.default_timer()

firstMatrix = []
csvfile1 = open("C:\\matrix1.csv","r",encoding='utf-8-sig')
reader1 = csv.reader(csvfile1)
for line in reader1:
    eachline = []
    for i in line:
        eachline.append(int(i))
    firstMatrix.append(eachline)

secondMatrix = []
csvfile2 = open("C:\\matrix2.csv","r",encoding='utf-8-sig')
reader2 = csv.reader(csvfile2)
for line in reader2:
    eachline = []
    for i in line:
        eachline.append(int(i))
    secondMatrix.append(eachline)

x = numpy.array(firstMatrix, dtype='int64')
y = numpy.array(secondMatrix, dtype='int64')

print ("The product of matrices is : ")
fullMultiplication = numpy.dot(x,y)
print (fullMultiplication)

with open("D:\\matrix3.csv","w+") as my_csv:
    csvWriter = csv.writer(my_csv,delimiter=',')
    csvWriter.writerows(fullMultiplication)

# Calculate the checkSum for each rows and columns of C matrix

checkSumRow = []
for i in range(0,1024):
    sum = 0
    for j in range(0,1024):
        sum += fullMultiplication[i][j]
    checkSumRow.append(sum)

checkSumColumn = []
for i in range(0,1024):
    sum = 0
    for j in range(0,1024):
        sum += fullMultiplication[j][i]
    checkSumColumn.append(sum)

# Gernerate a fault in C matrix
```

```
faultRow = random.randint(0,1023)
faultColumn = random.randint(0,1023)

if generateFault != "5":
    print("Injecting an error in row: ")
    print(faultRow)
    print("Injecting an error in column: ")
    print(faultColumn)

if generateFault == "3":
    fullMultiplication[faultRow][faultColumn] += 5


if generateFault == "1":
    firstMatrix[faultRow][faultColumn] += 5

if generateFault == "2":
    secondMatrix[faultRow][faultColumn] += 5

x = numpy.array(firstMatrix, dtype='int64')
y = numpy.array(secondMatrix, dtype='int64')
fullMultiplication2 = numpy.dot(x,y)



# Detect a fault in C matrix

for i in range(0,1024):
    sum = 0
    for j in range(0,1024):
        sum += fullMultiplication[i][j]
    if sum != checkSumRow[i]:
        print("The error is from matrix C, The error row is : ")
        print(i)


for i in range(0,1024):
    sum = 0
    for j in range(0,1024):
        sum += fullMultiplication[j][i]
    if sum != checkSumColumn[i]:
        print("The error is from matrix C, The error column is : ")
        print(i)
        stop = timeit.default_timer()
        print("Time: " , stop - start)
        print("seconds.")
        sys.exit()


for i in range(0,1024):
    sum = 0
    for j in range(0,1024):
        sum += fullMultiplication2[i][j]
    if sum != checkSumRow[i]:
        print("\nThere is an ERROR !!!!! \nThe error is from matrix A or B")
        break

stop = timeit.default_timer()

print("Time: " , stop - start)
print("seconds.")
```

2. Version 2 Code

```
import numpy
import random
import csv
import sys
import timeit

selectMatrixToInject = input("Please inject an error: 1. To A.  2. To B.  3. To C
4. Nothing")

start = timeit.default_timer()

firstMatrix = []
csvfile1 = open("C:\\matrix1.csv","r",encoding='utf-8-sig')
reader1 = csv.reader(csvfile1)
for line in reader1:
    eachline = []
    for i in line:
        eachline.append(int(i))
    firstMatrix.append(eachline)

secondMatrix = []
csvfile2 = open("C:\\matrix2.csv","r",encoding='utf-8-sig')
reader2 = csv.reader(csvfile2)
for line in reader2:
    eachline = []
    for i in line:
        eachline.append(int(i))
    secondMatrix.append(eachline)

x = numpy.array(firstMatrix, dtype='int64')
y = numpy.array(secondMatrix, dtype='int64')



m = x.shape[0]  #image row size
n = x.shape[1]  #image column size

p = 4     #block row size
q = 4     #block column size

block_array = []
previous_row = 0
for row_block in range(256):
    previous_row = row_block * p
    previous_column = 0
    for column_block in range(256):
        previous_column = column_block * q
        block = x[previous_row:previous_row+p,previous_column:previous_column+q]
        block_array.append(block)

block_array = numpy.array(block_array)

#print(block_array)


block_array2 = []
previous_row = 0
for row_block in range(256):
    previous_row = row_block * p
    previous_column = 0
    for column_block in range(256):
```

```
        previous_column = column_block * q
        block = y[previous_row:previous_row+p,previous_column:previous_column+q]
        block_array2.append(block)

block_array2 = numpy.array(block_array2)

#print(block_array2)

fullMultiplication = []
for i in range(len(block_array)):
    fullMultiplication.append(numpy.dot(block_array[i], block_array2[i]))


print(fullMultiplication)

with open("D:\\matrix4.csv","w+") as my_csv:
    csvWriter = csv.writer(my_csv,delimiter=',')
    csvWriter.writerows(fullMultiplication)


faultRow = random.randint(0,1023)
faultColumn = random.randint(0,1023)

if selectMatrixToInject == "1":
    firstMatrix[faultRow][faultColumn] += 5
if selectMatrixToInject == "2":
    secondMatrix[faultRow][faultColumn] += 5
x = numpy.array(firstMatrix, dtype='int64')
y = numpy.array(secondMatrix, dtype='int64')
fullMultiplication2 = numpy.dot(x,y)



block_array = []
previous_row = 0
for row_block in range(256):
    previous_row = row_block * p
    previous_column = 0
    for column_block in range(256):
        previous_column = column_block * q
        block = x[previous_row:previous_row+p,previous_column:previous_column+q]
        block_array.append(block)

block_array = numpy.array(block_array)

#print(block_array)


block_array2 = []
previous_row = 0
for row_block in range(256):
    previous_row = row_block * p
    previous_column = 0
    for column_block in range(256):
        previous_column = column_block * q
        block = y[previous_row:previous_row+p,previous_column:previous_column+q]
        block_array2.append(block)

block_array2 = numpy.array(block_array2)

#print(block_array2)
```

```
fullMultiplication2 = []
for i in range(len(block_array)):
    fullMultiplication2.append(numpy.dot(block_array[i], block_array2[i]))

errorBlock = random.randint(0,65536)
if selectMatrixToInject == "3":
    print("Block : ")
    print(errorBlock)
    print(" of C matrix has been injected an Error !")
    fullMultiplication2[errorBlock][random.randint(0,3)][random.randint(0,3)] +=
5

for i in range(len(fullMultiplication)):
    if ((fullMultiplication[i] == fullMultiplication2[i]).all()):
        continue
    else:
        print("There is an ERROR!!!! in ")
        print(i)
        print("th block")
        break

stop = timeit.default_timer()

print("Time: " , stop - start)
print("seconds.")
```