CS301
Programming Assignment #2
Thomas Crow

Source code:

Matrix_multiplication.cpp

```cpp
//  Assignment #2 -- C++
//  Thomas Crow
//  Using Python, C++, and Java create three programs to multiply
//  two square matrices of random integers between 1 and 10.
//  The sizes of these matrices are: 250, 500, 1000, 1500 and 2000
//  elements.
//
//  In other words, multiply two 250x250 matrices, two 500x500
//  matrices, etc.


#include <cstdlib>
#include <iostream>

using namespace std;

class Matrix_Multiplication {

    public:

    void multiply_matrix(int matrix_size) {

        //Initialize 2D Arrays
        int matrixA[matrix_size][matrix_size];
        int matrixB[matrix_size][matrix_size];
        int matrixC[matrix_size][matrix_size];

        //Assign values to arrays
        for (int i=0; i<matrix_size; i++) {
            for (int j=0; j<matrix_size; j++) {
                matrixA[i][j] = (rand () % 10) + 1;
                matrixB[i][j] = (rand () % 10) + 1;
                matrixC[i][j] = 0;
            }
        }

        //Assign matrixA * matrixB to matrixC
        for (int i=0; i<matrix_size; i++){
            for (int j=0; j<matrix_size; j++) {
                for (int k=0; k<matrix_size; k++) {
                    matrixC[i][j]+=matrixA[i][k]*matrixB[k][j];
                }
            }
        }
    }
};

int main() {

    Matrix_Multiplication myMatrix;
    int matrix_size = 1500;
    myMatrix.multiply_matrix(matrix_size);

    return 0;
}
```

Matrix_Multiplication.java

EXPLORER

ASSIGNMENT #2

- ~$ow_Thomas_PA2.docx U
- Crow_Thomas_PA2.docx U
- Matrix_Multiplication.cpp M
- Matrix_Multiplication.java M
- Matrix_Multiplication.py M

Matrix_Multiplication.java M | Matrix_Multiplication.cpp M | Matrix_Multiplication.py M

Matrix_Multiplication.java > Matrix_Multiplication > main(String[])

```java
//  Assignment #2 -- Java
//  Thomas Crow
//  Using Python, C++, and Java create three programs to multiply
//  two square matrices of random integers between 1 and 10.
//  The sizes of these matrices are: 250, 500, 1000, 1500 and 2000
//  elements.
//
//  In other words, multiply two 250x250 matrices, two 500x500
//  matrices, etc.


public class Matrix_Multiplication {

    public static void multiply_matrix(int matrix_size) {

        //Initialize 2D Arrays
        int [] [] matrixA = new int [matrix_size] [matrix_size];
        int [] [] matrixB = new int [matrix_size] [matrix_size];
        int [] [] matrixC = new int [matrix_size] [matrix_size];

        //Assign values to arrays
        for (int i=0; i<matrix_size; i++) {
            for (int j=0; j<matrix_size; j++) {
                matrixA[i][j] = (int) (Math.random()*10);
                matrixB[i][j] = (int) (Math.random()*10);
                matrixC[i][j] = 0;
            }
        }

        //Assign matrixA * matrixB to matrixC
        for (int i=0; i<matrix_size; i++){
            for (int j=0; j<matrix_size; j++) {
                for (int k=0; k<matrix_size; k++) {
                    matrixC[i][j]+=matrixA[i][k]*matrixB[k][j];
                }
            }
        }
    }

    Run | Debug
    public static void main(String[] args) {

        int matrix_size = 1500;
        multiply_matrix(matrix_size);
    }
}
```

Matrix_Multiplication.py

EXPLORER

ASSIGNMENT #2

- ~$ow_Thomas_PA2.docx U
- Crow_Thomas_PA2.docx U
- Matrix_Multiplication.cpp M
- Matrix_Multiplication.java M
- Matrix_Multiplication.py M

Matrix_Multiplication.java M | Matrix_Multiplication.cpp M | Matrix_Multiplication.py M

Matrix_Multiplication.py > main

```python
#  Assignment #2 -- Python
#  Thomas Crow
#
#  Using Python, C++, and Java create three programs to multiply
#  two square matrices of random integers between 1 and 10.
#  The sizes of these matrices are: 250, 500, 1000, 1500 and 2000
#  elements.
#
#  In other words, multiply two 250x250 matrices, two 500x500
#  matrices, etc.

import random


def multiply_matrix(matrix_size):

    #Initialize 2D Arrays
    matrixA=[]
    matrixB=[]
    matrixC=[]
    for i in range(matrix_size):
        matrixA_column_elements=[]
        matrixB_column_elements=[]
        matrixC_column_elements=[]
        #Assign values to arrays
        for j in range(matrix_size):
            matrixA_column_elements.append(random.randint(1, 10))
            matrixB_column_elements.append(random.randint(1, 10))
            matrixC_column_elements.append(0)
        matrixA.append(matrixA_column_elements)
        matrixB.append(matrixB_column_elements)
        matrixC.append(matrixC_column_elements)
    #Assign matrixA * matrixB to matrixC
    for i in range(matrix_size):
        for j in range(matrix_size):
            for k in range(matrix_size):
                matrixC[i][j] = matrixA[i][k] * matrixB[k][j]


def main():

    matrix_size = 1500
    multiply_matrix(matrix_size)

main()
```

Runtime Screenshots:

250x250 Matrices

C++

```
bash-3.2$ g++ Matrix_Multiplication.cpp
bash-3.2$ time ./a.out

real    0m0.362s
user    0m0.069s
sys     0m0.005s
bash-3.2$ time ./a.out

real    0m0.072s
user    0m0.066s
sys     0m0.004s
bash-3.2$ time ./a.out

real    0m0.070s
user    0m0.067s
sys     0m0.003s
bash-3.2$
```

Java

```
bash-3.2$ javac Matrix_Multiplication.java
bash-3.2$ time java Matrix_Multiplication

real    0m0.120s
user    0m0.098s
sys     0m0.022s
bash-3.2$ time java Matrix_Multiplication

real    0m0.122s
user    0m0.099s
sys     0m0.021s
bash-3.2$ time java Matrix_Multiplication

real    0m0.120s
user    0m0.097s
sys     0m0.023s
bash-3.2$
```

Python

```
bash-3.2$ time python3 Matrix_Multiplication.py

real    0m1.272s
user    0m1.161s
sys     0m0.047s
bash-3.2$ time python3 Matrix_Multiplication.py

real    0m1.205s
user    0m1.178s
sys     0m0.022s
bash-3.2$ time python3 Matrix_Multiplication.py

real    0m1.198s
user    0m1.173s
sys     0m0.020s
bash-3.2$
```

500x500 Matrices

C++

```
bash-3.2$ g++ Matrix_Multiplication.cpp
bash-3.2$ time ./a.out

real    0m0.644s
user    0m0.303s
sys     0m0.008s
bash-3.2$ time ./a.out

real    0m0.309s
user    0m0.301s
sys     0m0.006s
bash-3.2$ time ./a.out

real    0m0.309s
user    0m0.302s
sys     0m0.005s
bash-3.2$
```

Java

```
bash-3.2$ javac Matrix_Multiplication.java
bash-3.2$ time java Matrix_Multiplication

real    0m0.419s
user    0m0.401s
sys     0m0.025s
bash-3.2$ time java Matrix_Multiplication

real    0m0.426s
user    0m0.397s
sys     0m0.030s
bash-3.2$ time java Matrix_Multiplication

real    0m0.419s
user    0m0.401s
sys     0m0.026s
bash-3.2$
```

Python

```
bash-3.2$ time python3 Matrix_Multiplication.py

real    0m9.864s
user    0m9.759s
sys     0m0.092s
bash-3.2$ time python3 Matrix_Multiplication.py

real    0m9.879s
user    0m9.772s
sys     0m0.092s
bash-3.2$ time python3 Matrix_Multiplication.py

real    0m9.974s
user    0m9.859s
sys     0m0.096s
bash-3.2$
```

1000x1000 Matrices

C++

```
bash-3.2$ g++ Matrix_Multiplication.cpp
bash-3.2$ time ./a.out

real    0m2.541s
user    0m2.169s
sys     0m0.021s
bash-3.2$ time ./a.out

real    0m2.187s
user    0m2.161s
sys     0m0.024s
bash-3.2$ time ./a.out

real    0m2.180s
user    0m2.158s
sys     0m0.021s
bash-3.2$
```

Java

```
bash-3.2$ javac Matrix_Multiplication.java
bash-3.2$ time java Matrix_Multiplication

real    0m1.287s
user    0m1.252s
sys     0m0.032s
bash-3.2$ time java Matrix_Multiplication

real    0m1.289s
user    0m1.259s
sys     0m0.030s
bash-3.2$ time java Matrix_Multiplication

real    0m1.285s
user    0m1.260s
sys     0m0.022s
bash-3.2$
```

Python

```
bash-3.2$ time python3 Matrix_Multiplication.py

real    1m21.366s
user    1m20.719s
sys     0m0.557s
bash-3.2$ time python3 Matrix_Multiplication.py

real    1m21.951s
user    1m21.233s
sys     0m0.596s
bash-3.2$ time python3 Matrix_Multiplication.py

real    1m21.759s
user    1m21.099s
sys     0m0.618s
bash-3.2$
```

1500x1500

C++

```
bash-3.2$ g++ Matrix_Multiplication.cpp
bash-3.2$ time ./a.out

real    0m7.999s
user    0m7.588s
sys     0m0.020s
bash-3.2$ time ./a.out

real    0m7.588s
user    0m7.562s
sys     0m0.023s
bash-3.2$ time ./a.out

real    0m7.604s
user    0m7.583s
sys     0m0.021s
bash-3.2$
```

Java

```
bash-3.2$ javac Matrix_Multiplication.java
bash-3.2$ time java Matrix_Multiplication

real    0m4.614s
user    0m4.585s
sys     0m0.039s
bash-3.2$ time java Matrix_Multiplication

real    0m4.548s
user    0m4.513s
sys     0m0.044s
bash-3.2$ time java Matrix_Multiplication

real    0m4.558s
user    0m4.530s
sys     0m0.037s
bash-3.2$
```

Python

```
bash-3.2$ time python3 Matrix_Multiplication.py

real    4m37.001s
user    4m36.580s
sys     0m0.361s
bash-3.2$ time python3 Matrix_Multiplication.py

real    4m36.865s
user    4m36.446s
sys     0m0.390s
bash-3.2$ time python3 Matrix_Multiplication.py

real    4m39.043s
user    4m38.292s
sys     0m0.702s
bash-3.2$
```

2000x2000 Matrices

C++

```
bash-3.2$ g++ Matrix_Multiplication.cpp
bash-3.2$ time ./a.out

real    0m20.947s
user    0m20.269s
sys     0m0.163s
bash-3.2$ time ./a.out

real    0m20.123s
user    0m19.949s
sys     0m0.166s
bash-3.2$ time ./a.out

real    0m20.245s
user    0m20.104s
sys     0m0.141s
bash-3.2$
```

Java

```
bash-3.2$ javac Matrix_Multiplication.java
bash-3.2$ time java Matrix_Multiplication

real    0m12.370s
user    0m12.275s
sys     0m0.094s
bash-3.2$ time java Matrix_Multiplication

real    0m12.166s
user    0m12.043s
sys     0m0.133s
bash-3.2$ time java Matrix_Multiplication

real    0m12.543s
user    0m12.421s
sys     0m0.128s
bash-3.2$
```

Python

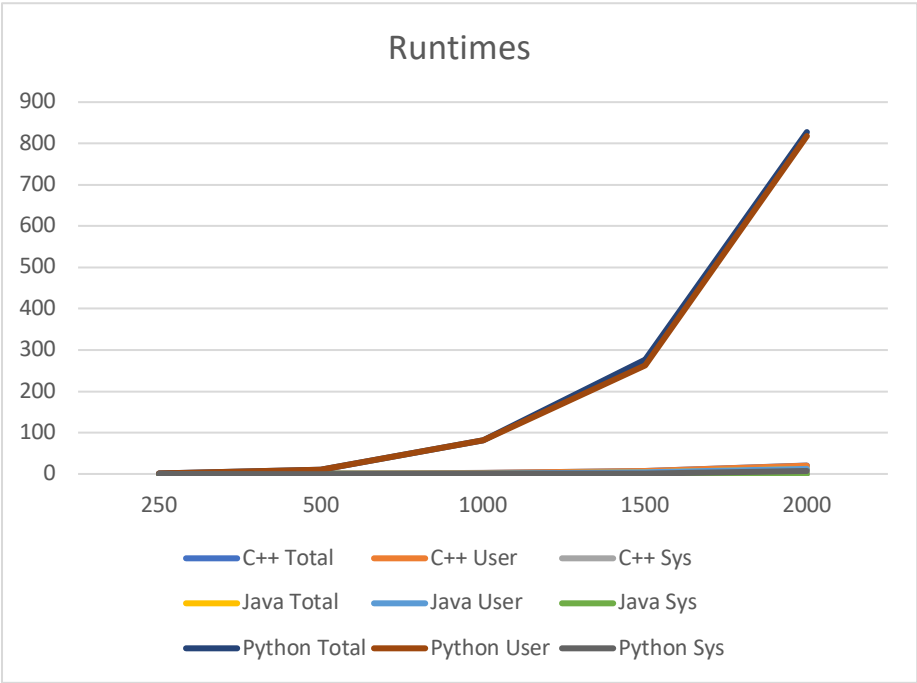
```
bash-3.2$ time python3 Matrix_Multiplication.py

real    13m53.888s
user    13m39.971s
sys     0m8.344s
bash-3.2$ time python3 Matrix_Multiplication.py

real    14m6.246s
user    13m58.965s
sys     0m6.939s
bash-3.2$ time python3 Matrix_Multiplication.py

real    13m22.469s
user    13m14.773s
sys     0m6.553s
bash-3.2$
```

Runtime graph:



Conclusion:
Java and C++ were by far faster in these tests than Python. As you can see in the graph, Python poor performance makes the other two languages virtually indistinguishable. This is likely due to Java and C++ being compiler-based languages versus the interpreter-based Python. None of the languages when a library wasn't in use took advantage of parallel processing. Utilizing parallel processing, either built into the language or via a library, would greatly increase performance. As would optimized code that using a library like numpy would provide.