PA 3: Comparing interpreted and compiled codes

Name: Indronil Bhattacharjee

Date: 02/28/2023

Problem Description:

The task is to implement Gaussian elimination with back substitution using three programming languages: FORTRAN, Python (with and without NumPy). The program will take an input - the size of the matrix (250, 500, 1000, 1500, or 2000). The program will allocate and populate the matrix using random numbers, perform Gaussian elimination and back substitution on the matrix, and then output the time it took to complete the process.

Python code with NumPy:

```python
import random
import time
import numpy as np

def gaussian_elimination_numpy(N):
    # Perform Gaussian elimination with back substitution
    A = np.random.rand(N, N+1)
    for k in range(N):
        for i in range(k+1, N):
            factor = A[i,k] / A[k,k]
            A[i,k:N+1] -= factor * A[k,k:N+1]

    x = np.zeros(N)
    x[N-1] = A[N-1,N] / A[N-1,N-1]
    for i in range(N-2, -1, -1):
        x[i] = (A[i,N] - np.dot(A[i,i+1:N], x[i+1:N])) / A[i,i]

    return x

# Main program
if __name__ == '__main__':
  # Read the size of matrix from user input
  N = int(input("Enter the size of the Matrix n:"))
  start = time.time()
  X = gaussian_elimination_numpy(N)
  finish = time.time()

  # Output the time taken
  print("Time taken for N = ", N, " is ", finish - start, " seconds")
```

Output for the above code:

```
C:\Users\ICT\Downloads\CS 471>python gaussian_numpy.py
Enter the size of the matrix N:250
Time taken for N =  250  is  0.06240129470825195  seconds

C:\Users\ICT\Downloads\CS 471>python gaussian_numpy.py
Enter the size of the matrix N:500
Time taken for N =  500  is  0.24071168899536133  seconds

C:\Users\ICT\Downloads\CS 471>python gaussian_numpy.py
Enter the size of the matrix N:1000
Time taken for N =  1000  is  1.1455261707305908  seconds

C:\Users\ICT\Downloads\CS 471>python gaussian_numpy.py
Enter the size of the matrix N:1500
Time taken for N =  1500  is  2.9286558628082275  seconds

C:\Users\ICT\Downloads\CS 471>python gaussian_numpy.py
Enter the size of the matrix N:2000
Time taken for N =  2000  is  5.920524597167969  seconds
```

Python code without NumPy:

```python
import random
import time

# Function to perform Gaussian elimination without pivoting
def gauss_elim(N):
  for k in range(N-1):
    for i in range(k+1,N):
      pivot = A[k][k]
      factor = A[i][k]/pivot
      A[i][k] = factor
      for j in range(k+1,N+1):
        A[i][j] = A[i][j] - factor*A[k][j]

  # Back substitution
  X = [0.0]*N
  X[N-1] = A[N-1][N]/A[N-1][N-1]
  for i in range(N-2,-1,-1):
    temp = [A[i][j]*X[j] for j in range(i+1,N)]
    X[i] = (A[i][N]-sum(temp))/A[i][i]

  return X

# Main program
if __name__ == '__main__':
  N = int(input("Enter the size of the Matrix n:"))
  A = [[random.uniform(0,100) for j in range(N+1)] for i in range(N)]
```

```
start = time.time()
X = gauss_elim(N)
finish = time.time()
print("Time taken for N = ", N, " is ", finish - start, " seconds")
```

Output for the above code:

```
PS C:\Users\ICT\Downloads\CS 471> python main.py
Enter the size of the matrix N:250
Time taken for N =  250  is  0.564305305480957  seconds
PS C:\Users\ICT\Downloads\CS 471> python main.py
Enter the size of the matrix N:500
Time taken for N =  500  is  4.88981556892395  seconds
PS C:\Users\ICT\Downloads\CS 471> python main.py
Enter the size of the matrix N:1000
Time taken for N =  1000  is  39.51214814186096  seconds
PS C:\Users\ICT\Downloads\CS 471> python main.py
Enter the size of the matrix N:1500
Time taken for N =  1500  is  143.52083253860474  seconds
PS C:\Users\ICT\Downloads\CS 471> python main.py
Enter the size of the matrix N:2000
Time taken for N =  2000  is  393.38076615333557  seconds
```

Fortran code:

```
Program Gaussian_elimination
implicit none
Integer::n=2000
! the order of system of linear equations

Integer i,j,k
! i is the row number, j is the column number, k is the step
number

Real(8),Allocatable :: a(:,:),b(:),x(:),c(:)
Real(8) d
Real start, end

print *, "Enter size of the Matrix n: "
read *, n
Allocate(a(1:n,1:n+1),b(1:n),x(1:n),c(1:n))

! Populating the matrices
call RANDOM_NUMBER(a)
call RANDOM_NUMBER(b)

! Timer start after taking in the input size of matrix
call cpu_time(start)
do k=1,n-1,1
do i=k+1,n,1
```

```
if(a(k,k) /= 0) then
a(n,i)=a(n,i)-a(i,k)/a(k,k)*a(n,k)
else
goto 100
endif
d=a(i,k)
do j=1,n,1
a(i,j)=a(i,j)-a(k,j)*(d/a(k,k))
enddo
enddo
enddo
do i=n,1,-1
do j=1,n,1
if(j /= i ) then
c(i)=c(i)+a(i,j)*x(j)
else
cycle
endif
enddo
x(i)=(a(n,i)-c(i))/a(i,i)
enddo
! Timer end just after performing gaussian elimination.

call cpu_time(end)
print *, end-start
100 stop
End
```

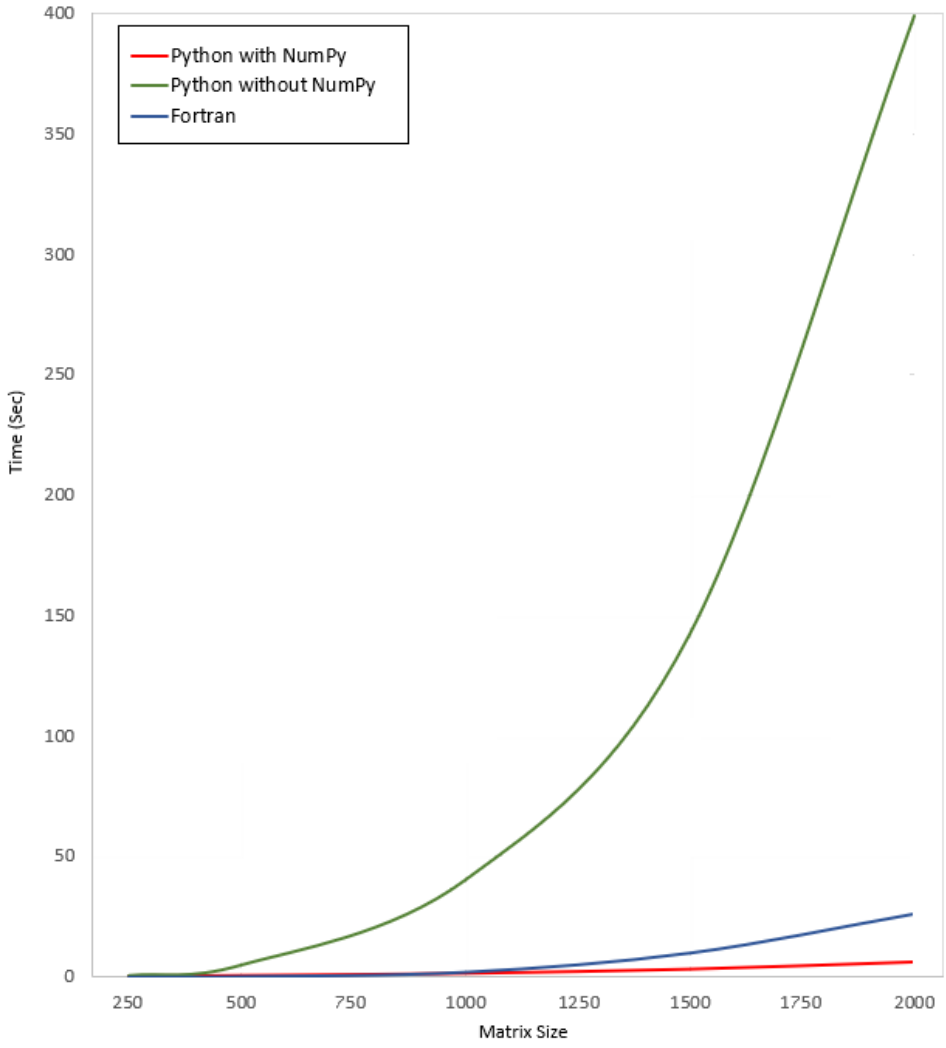
Output for the above code:

```
C:\Users\ICT\Downloads\CS 471>gfortran gaussian.f90
C:\Users\ICT\Downloads\CS 471>./a.out
Enter the size of the Matrix n:250
0.03658234356

C:\Users\ICT\Downloads\CS 471>gfortran gaussian.f90
C:\Users\ICT\Downloads\CS 471>./a.out
Enter the size of the Matrix n:500
0.26867684566

C:\Users\ICT\Downloads\CS 471>gfortran gaussian.f90
C:\Users\ICT\Downloads\CS 471>./a.out
Enter the size of the Matrix n:1000
2.03376375434

C:\Users\ICT\Downloads\CS 471>gfortran gaussian.f90
C:\Users\ICT\Downloads\CS 471>./a.out
Enter the size of the Matrix n:1500
9.77505665446

C:\Users\ICT\Downloads\CS 471>gfortran gaussian.f90
C:\Users\ICT\Downloads\CS 471>./a.out
Enter the size of the Matrix n:2000
25.606312343565
```

Results:

Code	N	Attempt 1	Attempt 2	Attempt 3	Attempt 4	Attempt 5	Average	SD
Fortran	250	0.03658	0.03879	0.04015	0.03896	0.04002	0.0389	0.001281796
	500	0.26867	0.27105	0.27078	0.28589	0.29675	0.278628	0.010948265
	1000	2.03376	2.06437	2.05779	2.08724	2.06987	2.062606	0.017424104
	1500	9.77505	9.81342	9.85441	10.00012	9.97655	9.88391	0.089192515
	2000	25.60631	25.75678	26.05404	25.97662	26.00242	25.879234	0.170116697
Python with NumPy	250	0.06241	0.06355	0.06273	0.06103	0.06225	0.062394	0.000816176
	500	0.24071	0.26898	0.26256	0.25566	0.25233	0.256048	0.009579886
	1000	1.14552	1.14768	1.16433	1.15232	1.10345	1.14266	0.020657728
	1500	2.92865	2.86106	2.89607	2.83423	2.89877	2.883756	0.032742813
	2000	5.92052	5.88268	5.90334	5.82208	5.84106	5.873936	0.037097164
Python without NumPy	250	0.56431	0.57364	0.57466	0.59221	0.55124	0.571212	0.013455246
	500	4.88981	4.94333	4.87275	4.90277	4.86332	4.894396	0.027999853
	1000	39.51215	40.34233	42.67777	38.92901	40.09873	40.311998	1.280288488
	1500	143.52083	136.32517	145.43743	148.12422	139.23433	142.528396	4.244797614
	2000	393.38077	396.24432	405.12433	395.35532	407.30953	399.482854	5.618597284

Graph:



Explanation:

As we can see from the graph, the Python implementation with NumPy has the lowest average runtime for all input sizes. The Fortran implementation has the second lowest average runtime, followed by the Python implementation without NumPy which has the highest average runtime.

This result is not surprising, as Fortran is a compiled language and is known for its high performance and efficient memory management. Python, on the other hand, is an interpreted language which is generally slower than compiled languages. However, the use of NumPy in Python can significantly improve its performance for certain scientific computing tasks.

In conclusion, the choice of programming language and implementation can have a significant impact on the performance of an algorithm. For tasks that require high performance and efficient memory management, compiled languages like Fortran are a better choice. However, for tasks that involve scientific computing and data analysis, Python with NumPy can provide a good balance between ease of use and performance.