Comparison of running time of a program in different languages

Program:

```
s=0
for i from 1 to 9999 do:
    for j from 1 to 9999 do:
        s=s+i/j
```

In Python:

```
import time
start_time = time.clock()
s=0
for i in range(1,9999,1):
      for j in range(1,9999,1):
                    s=s+i/j
print (time.clock() - start_time, "seconds")
```

The execution time is: **20 seconds**

In R:

```
start_time <- Sys.time()
s=0
for(i in 1:9999)
  for(j in 1:9999)
    s=s+i/j
end_time <- Sys.time()
x=end_time - start_time
x
```

The execution time is: **4 seconds**

In Java:

```java
import java.util.*;
public class Time
{
	public static void main(String args[])
	{
		try
		{
			long start_time = System.currentTimeMillis( );
			long s=0;
			for(long i=1;i<10000;i++)
			{
				for(long j=1;j<10000;j++)
				{
					s=s+i/j;
				}
			}
			long end_time = System.currentTimeMillis( );
			long diff = end_time-start_time;
			System.out.println(diff/1000+" seconds");
		}
		catch (Exception e)
		{
			System.out.println("Got an exception!");
		}
	}
}
```

The execution time is: **2 seconds**

In C++:

```cpp
#include<iostream>
#include <ctime>
using namespace std;
int main ()
{
  int start_time=clock();
  float s =0;
  float i=1,j=1;
  for(i=1;i<10000;i++)
    for(j=1;j<10000;j++)
      s=s+i/j;
  int end_time=clock();
  cout<<(end_time-start_time)/double(CLOCKS_PER_SEC)<<" seconds";
  return 0;
}
```

The execution time is: **1 second**

So, we see from this example that the execution time for same code in different languages is in the order:

Python > R > Java > C++

This result may lead to a question that "**If python is so inefficient, then why most of the development in the field of machine learning and deep learning is done in python?**". Since, machine learning algorithms are computation intensive, most efficient language should be used for execution of those algorithms.

One answer that I found is – **Python performs vectorized operations easily**.
Using the CPU and GPU capabilities python performs SIMD (single instruction multiple data) instructions with just a few lines of code.

Source:
https://www.coursera.org/learn/neural-networks-deep-learning/lecture/NYnog/vectorization

Example:

```
import numpy as np
import time
a=np.random.rand(1000000)
b=np.random.rand(1000000)
start_time = time.time()
c=np.dot(a,b)
end_time=time.time()
print("Vectorized version: "+str(1000*(end_time-start_time))+" ms")
start_time =time.time()
c=0
for i in range(1000000):
        c+=a[i]*b[i]
        end_time = time.time()
        print("For loop version: "+str(1000*(end_time-start_time))+" ms")
```

The execution time is: **Vectorized version: 1 ms**
For loop version: 563 ms

From this example we see that the vectorized computation in python is **faster** than running the same program using traditional for loop.

In Java:

```
import java.util.*;
public class LoopTime
{
       public static void main(String args[]){
             try {
                int[] a = new int[1000000];
                Random rand = new Random();
                for(int i=0;i<1000000;i++)
                {
                    a[i]= rand.nextInt();
                }
                long start_time = System.currentTimeMillis( );
                long c=0;
                for(int i=0;i<1000000;i++)
                {
                   c+=a[i]*a[i];
                }
                long end_time = System.currentTimeMillis( );
                long diff = end_time-start_time;
                System.out.println(diff+" ms");
             }
            catch (Exception e) {
                System.out.println("Got an exception!");
             }
       }
}
```

The execution time is: **4 ms**

Clearly, vectorized operation in python is still more efficient than for loop in java. But the difference is not huge. Parallel implementation in Java will definitely outperform the performance of vectorized operation in python.

In C++:

```
#include<iostream>
#include <ctime>
using namespace std;
int main ()
{
  int array[1000000];
  for( int i=0;i<1000000;i++)
  {
    array[i]=rand();
  }
  int start_time=clock();
  long c=0;
  for(int i=0;i<1000000;i++)
  {
    c+=array[i]+array[i];
  }
  int end_time=clock();
  cout <<(end_time-start_time)/double(CLOCKS_PER_SEC)*1000 <<" ms";
  return 0;
}
```

The execution time is: **3 ms**

This shows that the vectorized version of python is more efficient than C++ loop version. But vectorized implementation in C++ will outperform the performance of vectorized operation in python.

Probably, implementing the vectorized execution in Java or C++ will take a lot of effort but it is made convenient for use in python which made it popular in machine learning and deep learning community.

Credits: I want to thank my advisor Dr. Ovidiu Daescu for encouraging me to run this experiment.