The programming language used for this course is Python. So if you are not familiar with python programming Google "Python Tutorial" and select the suitable source for self learning depending on your experience. Invest first some time to figure out, what materials suit best for you. (There are tons of material on Python, including books)

We also use the python based libraries like Numpy, SciPy, Matplotlib

Some useful Links

- https://www.python-course.eu/numpy.php/
- https://www.dataquest.io/blog/numpy-tutorial-python/
- https://www.datacamp.com/community/tutorials/python-numpy-tutorial
  - Online coding environment is also available.

1 Objective

The main objective of this assignment is to familiar with python, its libraries and to understand and analyze the performance and limitation of the program from the parallel computing perspective.
2 Introduction

Fundamentally the analysis of the program is done using the asymptotic analysis. The performance of the parallel program however is measured in terms of Speed up. How fast the parallel program is compared to the sequential program for the same. As a preparatory assignment for the upcoming successive assignment in the course, we in this assignment get acquainted with speed up measurement, Numpy, plotting the result in graph using matplotlib.

Python provides the built-in time module for measuring the time. The time can be measured in different ways and while measuring time one need to be careful what function they are using because some are dependent on the platform like time.time(), time.clock().

Suggested Reading

- [https://docs.python.org/3/library/time.html](https://docs.python.org/3/library/time.html)

Example Time Measuring

```python
import time
sum = 0
start = time.time()
for i in range(1000000):
    sum += i
end = time.time()
print("Sum = {}, Total time taken = {}\).format(sum, (end-start))")
```

Listing 1: Measuring Time of the program

Result

Sum = 499999500000, Total time taken = 0.19977450370788574

Plotting with matplotlib
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines

def newline(p1, p2, color):
    ax = plt.gca()
    xmin, xmax = ax.get_xbound()
    if(p2[0] == p1[0]):
        xmin = xmax = p1[0]
        ymin, ymax = ax.get_ybound()
    else:
        ymax = p1[1]+(p2[1]-p1[1])/(p2[0]-p1[0])*(xmax-p1[0])
        ymin = p1[1]+(p2[1]-p1[1])/(p2[0]-p1[0])*(xmin-p1[0])
    l = mlines.Line2D([xmin,xmax], [ymin,ymax],color=color)
    ax.add_line(l)
    return l

n= 100 ; x = np.linspace(0.0, 1.0, n)
y_line = -2*x + 3 # original line
# generate a cloud of points randomly away from the original line:
y = y_line + np.random.normal(0, 0.55, n)
A = np.array([x, np.ones(n)])
A = A.transpose()
# Solve a least squares problem to find best constants to define a fitting line:
result = np.linalg.lstsq(A, y, rcond=None)
# print (result)
a, b = result[0]
p=[(x[i],y[i]) for i in range(len(x))]
p0 = (0,a*0 + b); p1 = (1,a*1 + b)

plt.figure(1)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Blue - original line; red - fitted line: ')
plt.legend(['Legend'])
plt.plot(x, y, 'r.')
newline(p0,p1,'blue')
newline((0,3),(1,1),'red')
plt.show()
3 Assignment

Fibonacci numbers

Fibonacci numbers were known already to Indian mathematicians centuries before, but they got named after Leonardo of Pisa, known as Fibonacci, who mentioned them in a book he wrote in the 1200’s. He considers the growth of a rabbit population, where

- Month 0 starts with just one pair of rabbits.
- In the first month, the first pair gives birth to another pair.
- In the second month, both pairs of rabbits have another pair, and the first pair dies.
- In general, each pair of rabbits has 2 pairs in its lifetime, and dies.
Let the population at month $n$ be $f_n$. At this time, only rabbits who were alive at month $n-2$ are fertile and produce offspring, so $f_{n-2}$ pairs are added to the current population of $f_{n-1}$. Thus the total is $f_n = f_{n-1} + f_{n-2}$. The recursion equation

$$f_n = f_{n-1} + f_{n-2}; \quad n > 1; \quad f_1 = 1; \quad f_0 = 0;$$

define the Fibonacci sequence. The terms of the sequence are Fibonacci numbers.

**Task 1 (5 Points):** Implement the recursive and non-recursive version of Fibonacci and measure the performance to find out which one is slow. Also write down the formula for getting approximate number of steps for recursive implementation. *Hint: Golden Ratio*

**Matrix-theoretic Algorithm**

There is a sublinear algorithm to replace this exponential algorithm. Consider the matrix

$$
\begin{pmatrix}
0 & 1 \\
1 & 1
\end{pmatrix}
$$

**Lemma.** For each $n > 0$, we have

$$F_n = \begin{pmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$$

Proof: The case $n = 1$ follows from the definition. Assume that $F_k = \begin{pmatrix} f_{k-1} & f_k \\ f_k & f_{k+1} \end{pmatrix}$, for some $k > 1$. We have

$$F_{k+1} = \begin{pmatrix} f_{k-1} & f_k \\ f_k & f_{k+1} \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} f_k & f_{k-1} + f_k \\ f_{k+1} & f_k + f_{k+1} \end{pmatrix}$$

The claim follows by induction.

It is possible to adapt the repeated squaring algorithm (below) to compute $F_n$. Since this has complexity, $O(\log n)$, this algorithm for computing
$f_n$ has complexity $O(\log n)$.

**Repeated squaring algorithm**

The basic idea is very simple. For input you have, say, a number $x$ and an integer $n > 0$. Assume $x$ is fixed, so we are really only interested in an efficient algorithm as a function of $n$.

We start with an example. Example. Compute $x^{13}$. First compute $x$ (0 steps), $x^4$ (2 steps, namely $x^2 = x \cdot x$ and $x^4 = x^2 \cdot x^2$), and $x^8$ (2 steps, namely $x^4$ and $x^8 = x^4 \cdot x^4$). Now (2 more steps)

$$x^{13} = x \cdot x^4 \cdot x^8.$$  

In general, we can compute $x^n$ in about $O(\log n)$ steps. Here is an implementation in Python.

```python
1 def power(x, n):
2     """
3     INPUT:
4        x - a number
5        n - an integer > 0
6     OUTPUT:
7        x^\text{n}
8     EXAMPLES:
9       >>> power(3,13)
10          1594323
11       >>> 3**(13)
12          1594323
13     """
14     if n == 1:
15         return x
16     if n%2 == 0:
17         return power(x, int(n/2))**2
18     if n%2 == 1:
19         return x*power(x, int((n-1)/2))**2
```

Listing 3: calculating n-th power of x
Task 2 (5 Points): Write a program to calculate Fibonacci number using the method of calculating power with 2X2 matrices i.e. matrices implementation of finding Fibonacci and measure the performance. Hint: you can define \( F_0 \) as: \( F = \text{np.matrix}([[0,1],[1,1]]) \)

Test Cases: Note we use additional testing to verify the correctness.

```
assert_equal(matrix_fib(0), 0)
assert_equal(matrix_fib(1), 1)
assert_equal(matrix_fib(2), 1)
assert_equal(matrix_fib(90), 2880067194370816120)
```

Listing 4: Test Cases

Example Running Tests:

```
from nose.tools import assert_equal

def checkForCorrectness(number):
    if number % 2 == 0:
        return 0
    return 1

#this will pass
def test_pass():
    assert_equal(checkForCorrectness(4), 0)
    assert_equal(checkForCorrectness(9), 1)

#this will fail
def test_fail():
    assert_equal(checkForCorrectness(4), 1)
    assert_equal(checkForCorrectness(9), 0)
```

Listing 5: Running Test Example

Command to run test: nosetests -v file.py; v – verbose
Task 3 (5 Points): What happens with matrices implementation of finding Fibonacci in case n greater than 93? Provide the reason and a fix in case of the problem.

Task 4 (5 Points): The Numpy library is designed to perform computations with arrays faster. But actually, how much improvement can one get when comparing with Python lists? Find it out and illustrate with a plot for different array lengths. You can do array operation of type $z = a \times (x + y)$, where $x$, $y$ and $z$ are arrays of length $n$ filled with random numbers and $a$ is a scalar value.

Task 5 (5 Points): Compare the three implementations of finding Fibonacci number (matrix, recursive and iterative) by plotting [Take i= 1..90].

4 Submission

Total 30 Points (5 Points for report, 25 for Tasks)
You need to submit through course submission page. If you have multiple files then you need to zip it in one file and upload. No submissions will be accepted after deadline. All the source codes must be provided where applicable, without it the answer will not be considered.
5 Linux Help

To run the code in HPC cluster, you need to have your file present there. It is difficult to work in HPC writing code as it is not having GUI access. So we suggest you to write the code in your system and move it to cluster and execute it.

rsync File Transfer

\textit{rsync options file\_to\_transfer username@remote\_host:destination}

More on rsync – \url{rsync tutorial}