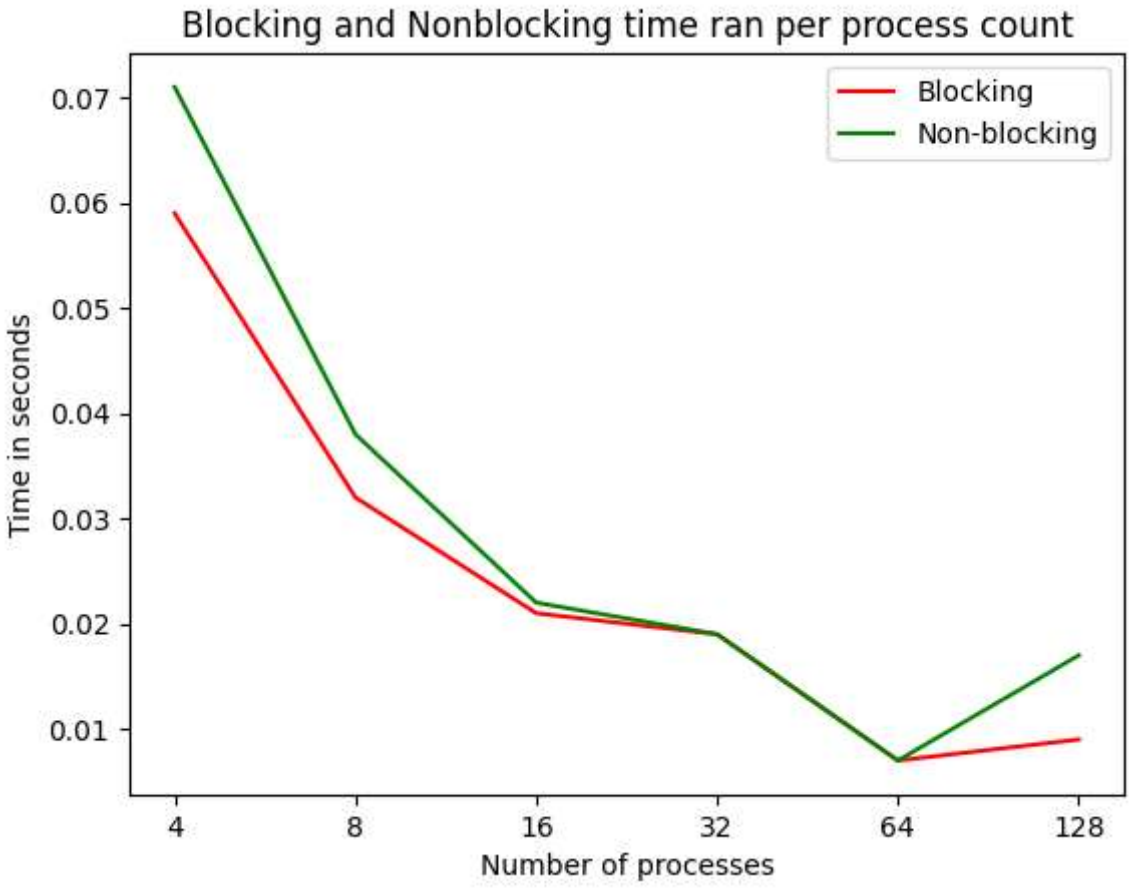
WriteUp

Graph

The following graph is performance result of running the two parellel version implemented on inputfile life.512x512 with 500 iterations.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame(np.array([[0.059,0.071],[0.032,0.038],[0.021,0.022],[0.019,0.019]
df = df.set_axis(['4','8','16','32','64','128'],axis = 'index')

plt.title("Blocking and Nonblocking time ran per process count")
plt.plot(df.index, df['Blocking'], color ='red', label='Blocking')
plt.plot(df.index, df['Non-blocking'], color ='green', label ='Non-blocking')
plt.ylabel("Time in seconds")
plt.xlabel("Number of processes")
plt.legend()
plt.show()
```



Data Distribution

For both Nonblocking and blocking, the data distribution was done the same way. In order to utilize mpi scatter which expectcs contiguous memory as send data and recieve data, I had to modify both send data and recieve data from the given serial code. For the send data, I've modified the read_input_file function given from serial code and converted 2d coordinate from csv file to 1d index of 1-d array to flatten 2-d coordinates to a 1-d array which is now contiguous in memory. For recieve data, in-order to utilize already implemented compute function in serial,which takes 2d array. I dynmically created 2-d array allocating contiguous memory. After send and recieve data was prepared, I did 1-d decomposition of rows by scatter by giving each processes (total row count/number of processes) rows.

Ensuring blocking version does not deadlock

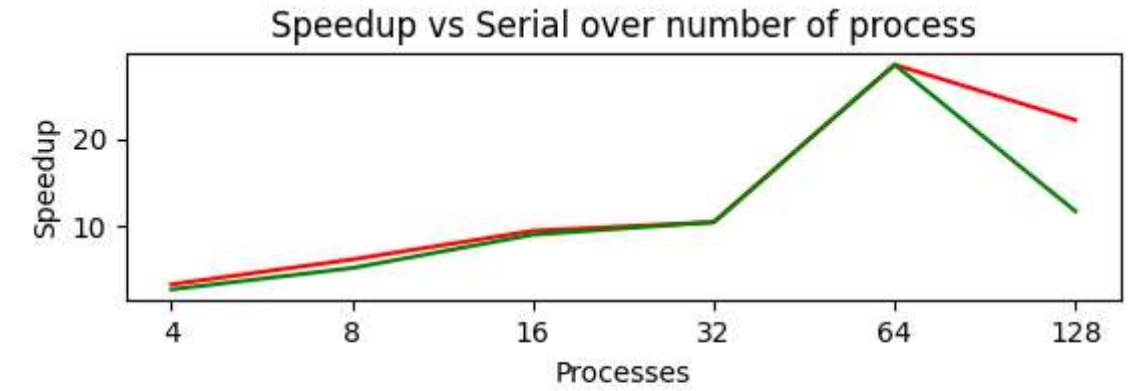
In order for blocking version to not deadlock, I have done all computation after the message passing and data exchange has been thorugh each iteration so that no one process sits on data and other waiting for signal. Also, there are no circular exchange of data; each process will always exchange with the same process every iteration beeing less likely for data to be exchanged safely.

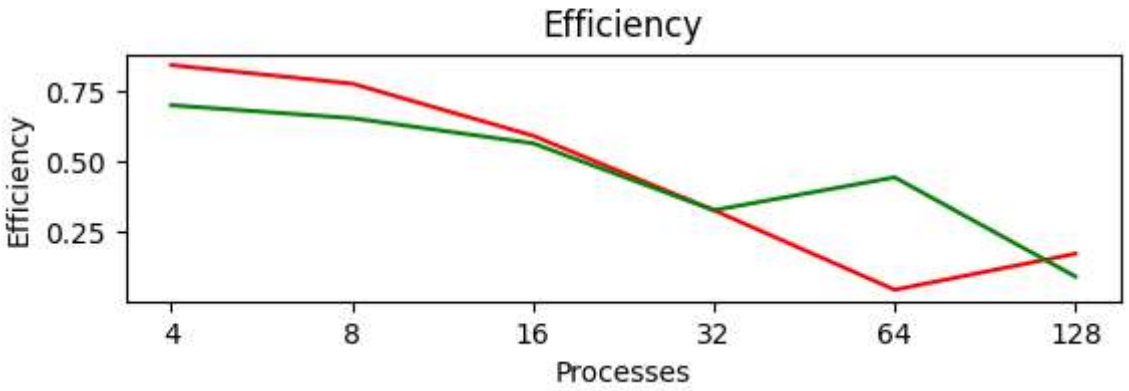
Performance results

The performance results are close to the given target. See below for speedupchart.

In [ ]:
```
df2 = pd.DataFrame(np.array([[0.199/0.059,0.199/0.071],[0.199/0.032,0.199/0.038],[0
df2 = df2.set_axis(['4','8','16','32','64','128'],axis = 'index')
df3 = pd.DataFrame(np.array([[0.199/(0.059*4),0.199/(0.071*4)],[0.199/(0.032*8),0.1
df3 = df3.set_axis(['4','8','16','32','64','128'],axis = 'index')

fig, (ax1,ax2) = plt.subplots(2)
plt.subplots_adjust(hspace = 1)

ax1.plot(df2.index, df2['Blocking'], color ='red', label='Blocking')
ax1.plot(df2.index, df2['Non-blocking'], color ='green', label='Non-blocking')
ax1.set_title("Speedup vs Serial over number of process")
ax1.set(xlabel ='Processes', ylabel='Speedup')


ax2.plot(df3.index, df3['Blocking'], color ='red', label='Blocking')
ax2.plot(df3.index, df3['Non-blocking'], color ='green', label='Non-blocking')
ax2.set_title("Efficiency")
ax2.set(xlabel ='Processes', ylabel='Efficiency')
```

```
fig.show()
```





The Performance results was mix of expeted result and unexpected reuslt for me.
I did expect that efficiency drop over the processes due to overhead.
On the otherhand, I did not expecet that unblocked would perform similar to blocked. I expected unblocked to perform noticeably better but It seems that they perform very similar. My guess is that the computation required in the project was very light that barely any wait time between processes such that performing calculation during waittime leads to no significant improvement.