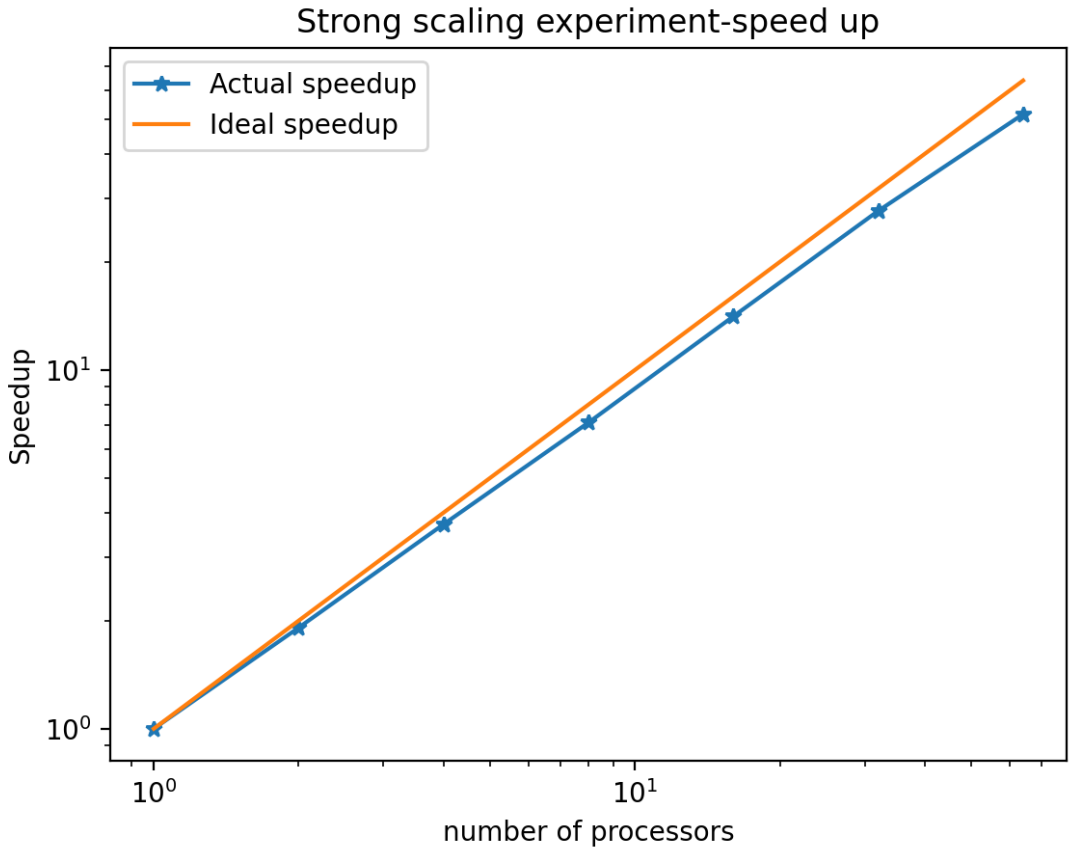
In [1]:
```
import pandas as pd
%matplotlib notebook
%pylab
```

```
Using matplotlib backend: nbAgg
Populating the interactive namespace from numpy and matplotlib
```

Task 1

(1)

In [2]:
```
#Uploading strong scaling data
d1 = pd.read_csv('task1_strong.csv')

#Calculating Speedup
T1 = d1.elapsed_time[0]
Tp = d1.elapsed_time
speedup = T1/Tp
ideal_speedup =d1.nproc

#Calculating efficiency
efficiency = speedup/d1.nproc
ideal_efficiency = speedup/speedup

# speedup plot
figure(1)
loglog(d1.nproc,speedup,'*-',label = 'Actual speedup')
loglog(d1.nproc,ideal_speedup,label = 'Ideal speedup')
title('Strong scaling experiment-speed up')
xlabel('number of processors')
ylabel('Speedup')
legend()
show()
figure(2)
semilogx(d1.nproc,efficiency,'*-',label = 'Actual efficiency')
semilogx(d1.nproc,ideal_efficiency,label = 'Ideal efficiency')
title('Strong scaling experiment-efficiency')
xlabel('number of processors')
ylabel('Efficiency')
legend()
grid()
show()
```



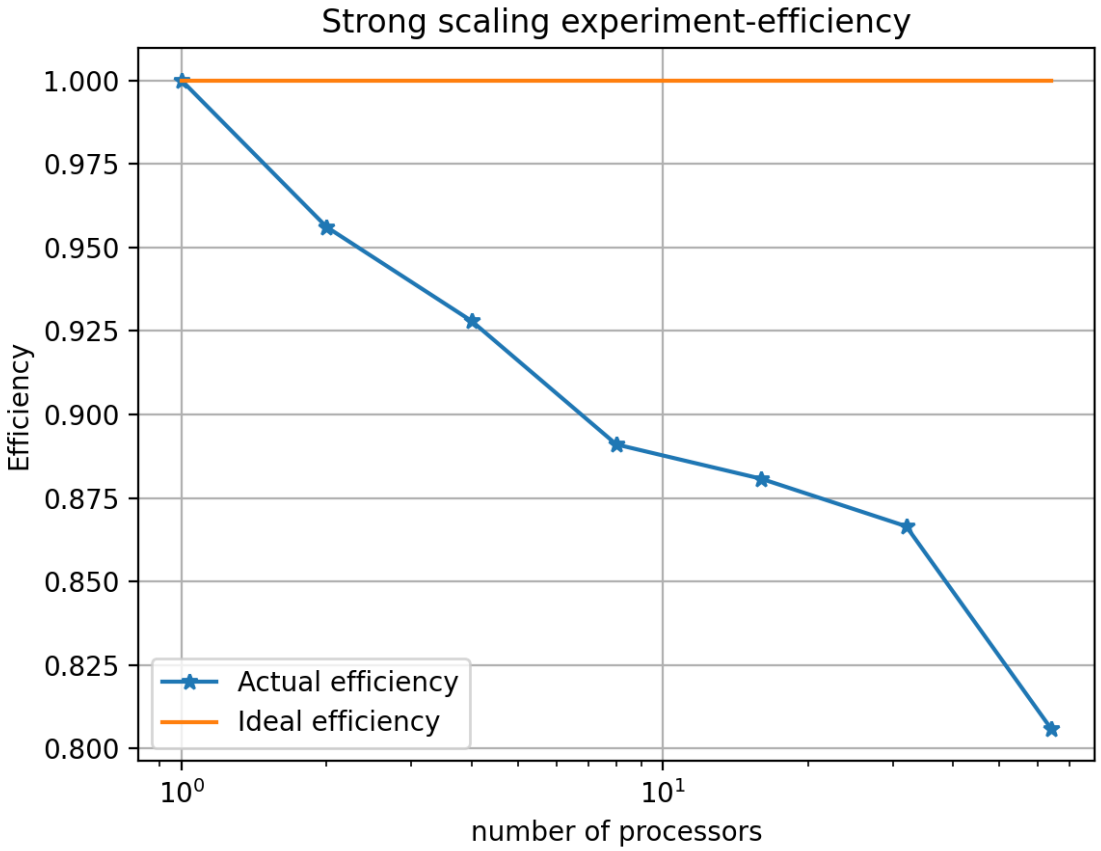


Figure 1 above shows a linear actual speedup which increases with increase in the number of

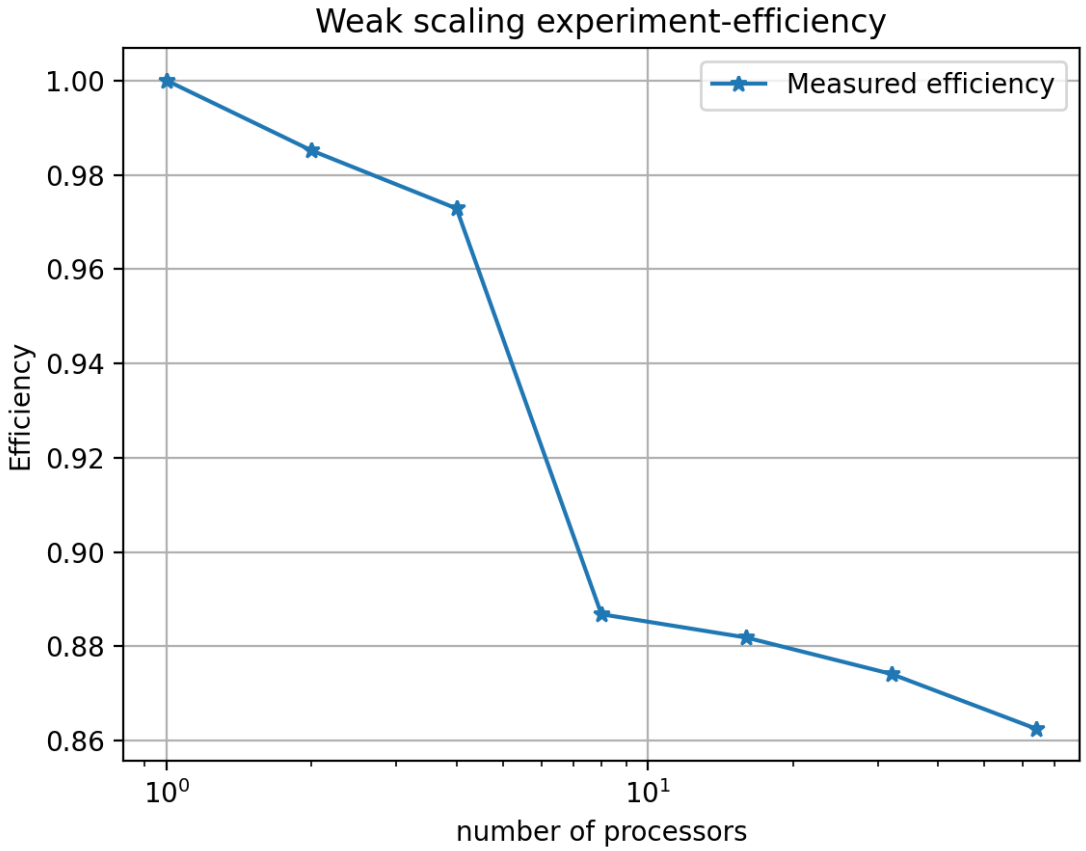
processors. It is observed that for number of processors equal to 2 and above, the actual speedup tends to diverge away from the Ideal speedup and since the divergence from the ideal speed up is not very small, it means that this experiment has a good strong scalability.

Figure 2 above shows that the efficiency of the strong scaling experiment decreases with increase in the number of processors by propotions between 0.025 and 0.075 and it decreases upto almost 80% for 64 processors. Although the computation time is decreased with increase in number of processors, the time for communication between the processors increases as the number of processors increases thus the decrease in efficiency. Therefore the parallel efficiency decreases with increase in number of processors.

(2)

In [3]:

```
#Uploading weak scaling data
d2 = pd.read_csv('task1_weak.csv')


#calculating efficiency
efficiency1 = d2.elapsed_time[0]/d2.elapsed_time

# efficiency plot
figure(3)
semilogx(d2.nproc,efficiency1,'*-',label = 'Measured efficiency')
title('Weak scaling experiment-efficiency')
xlabel('number of processors')
ylabel('Efficiency')
legend()
grid()
show()
```



In figure 3 above, it is observed that the efficiency of the weak scaling experiment decreases with increase in both the number of processors and problem size by propotions of ≤ 0.02 except for 8 processors where the efficiency decreases by almost 0.08. The efficiency decreases upto almost 86.2 for 64 processors which is better than that of the strong scaling experiment. The decrease in efficiency can be due to increase in communication time between the processors as the number of processors as well as the problem size increase.

Task 2

In [4]:
```
#Uploading strong scaling data
d = pd.read_csv('task_2.csv')
print('Standard deviation from Task 1 =' )
print(d1.std_dev[0])

print('Standard deviation from Task 2 =' )
print(d.std_dev[0])
```

```
Standard deviation from Task 1 =
0.288667
Standard deviation from Task 2 =
0.288667
```

Therefore the Welford's serial program gives the same result as that in Task 1.

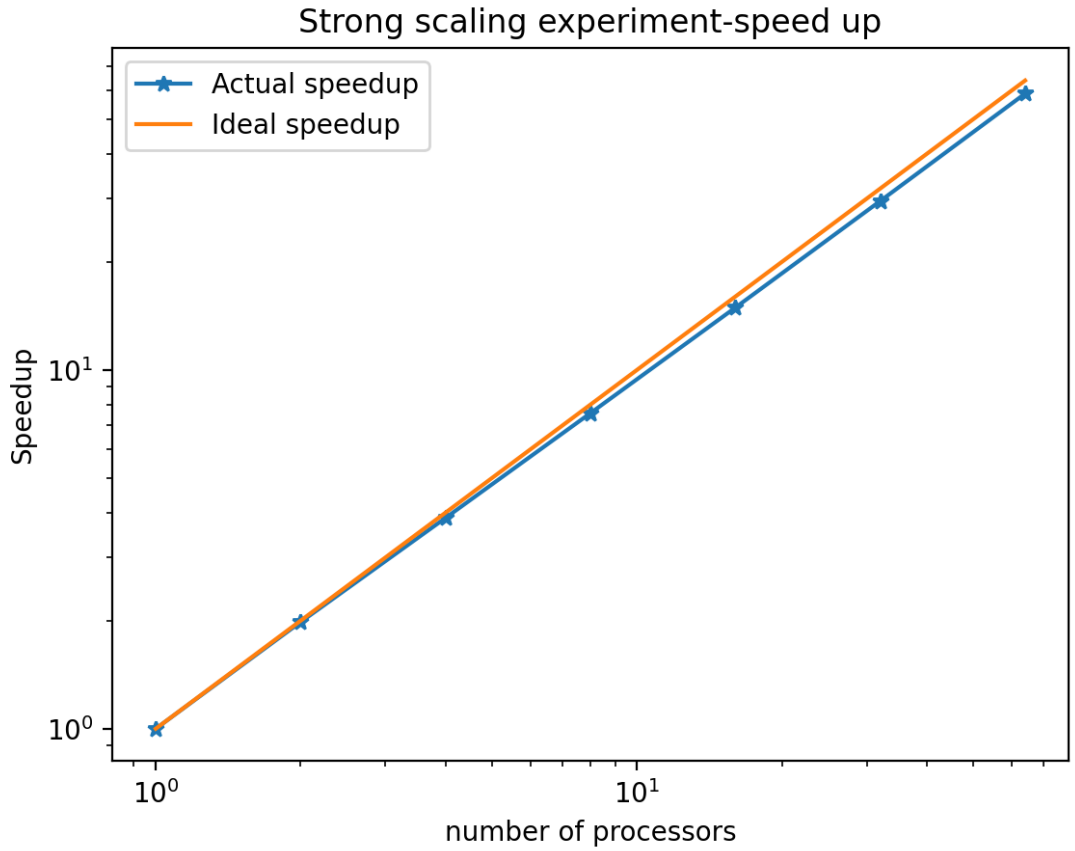
Task 3

(1)

(3)

In [5]:

```
#Uploading strong scaling data
d3 = pd.read_csv('task3_strong.csv')


#Calculating Speedup
T11 = d3.elapsed_time[0]
Tpp = d3.elapsed_time
speedup3 = T11/Tpp
ideal_speedup3 =d3.nproc

#Calculating efficiency
efficiency3 = speedup3/d3.nproc
ideal_efficiency3 = speedup3/speedup3

#speedup plot
figure(4)
loglog(d3.nproc,speedup3,'*-',label = 'Actual speedup')
loglog(d3.nproc,ideal_speedup3,label = 'Ideal speedup')
title('Strong scaling experiment-speed up')
xlabel('number of processors')
ylabel('Speedup')
legend()
#efficiency plot
figure(5)
semilogx(d3.nproc,efficiency3,'*-',label = 'Actual efficiency')
semilogx(d3.nproc,ideal_efficiency3,label = 'Ideal efficiency')
title('Strong scaling experiment-efficiency')
xlabel('number of processors')
ylabel('Efficiency')
legend()

grid()
show()
```



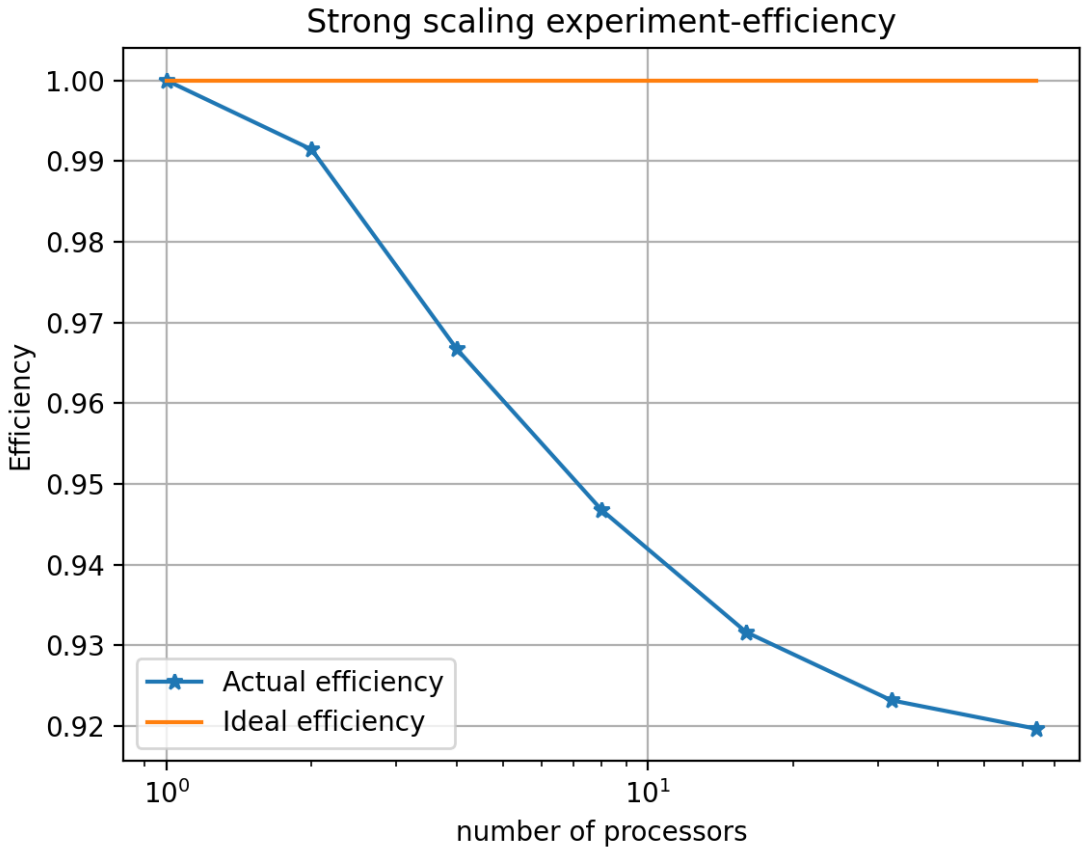


Figure 4 above shows a linear actual speedup which increases with increase in the number of

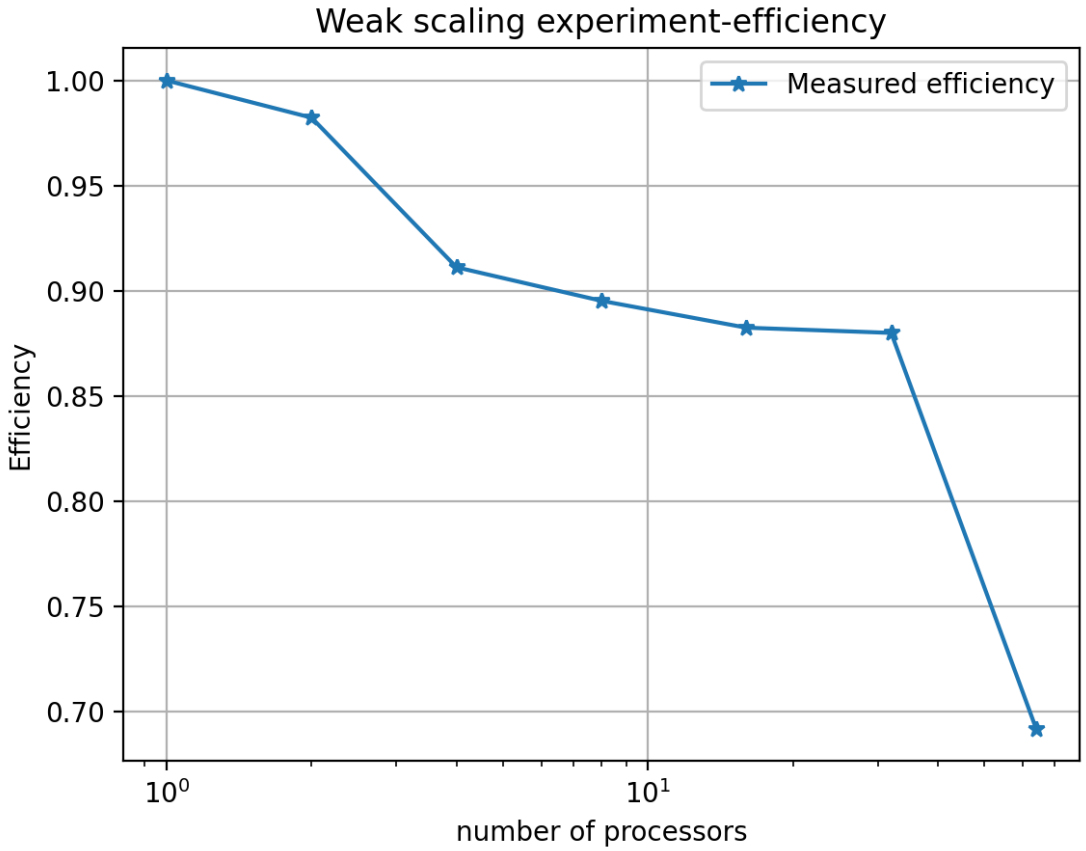
processors. It is observed that for number of processors equal to $1, 2$ and 3, the actual speedup is equal to the ideal speedup and thereafter, the actual speedup tends to diverge away from the Ideal speedup but since the divergence is very small, it means that this experiment has a very strong scalability and it is better than that in Task 1 shown by figure 1 above.

Figure 5 above shows that the efficiency of the strong scaling experiment decreases with increase in the number of processors and the efficiency decreases upto almost 92% for 64 processors. Similarly the efficiency decreases due to the increase in communication time as the number of processors increases. It is also observed that the strong scaling experiment efficiency for this task is better than that of Task 1.

(4)

In [6]:

```
#Uploading Weak scaling data
d4 = pd.read_csv('task3_weak.csv')

#calculatin efficiency
efficiency4 = d4.elapsed_time[0]/d4.elapsed_time

#efficiency plot
figure(6)
semilogx(d4.nproc,efficiency4,'*-',label = 'Measured efficiency')
title('Weak scaling experiment-efficiency')
xlabel('number of processors')
ylabel('Efficiency')
legend()
grid()
show()
```



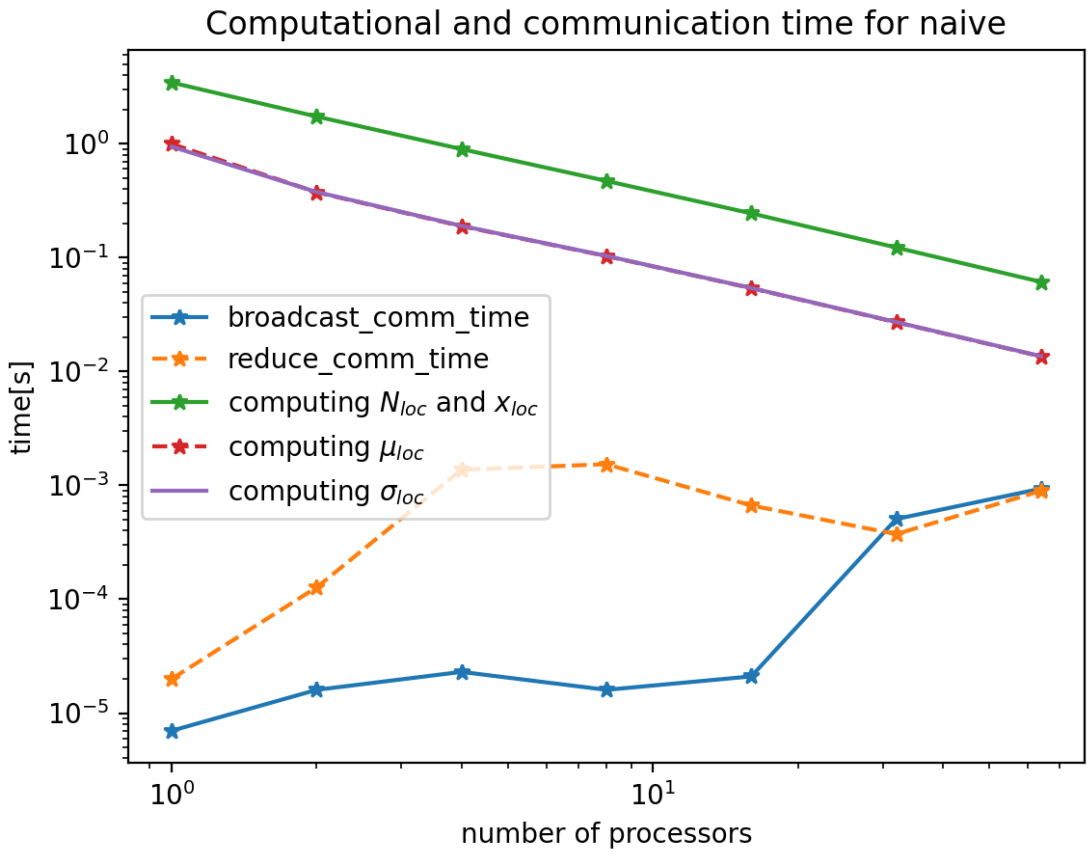
In figure 6 above, it is observed that the efficiency of the weak scaling experiment decreases with increase in both the number of processors and problem size up to 69% for 64 processors. And it is observed that the weak scaling experiment efficiency for this task is poorer than that of Task 1.

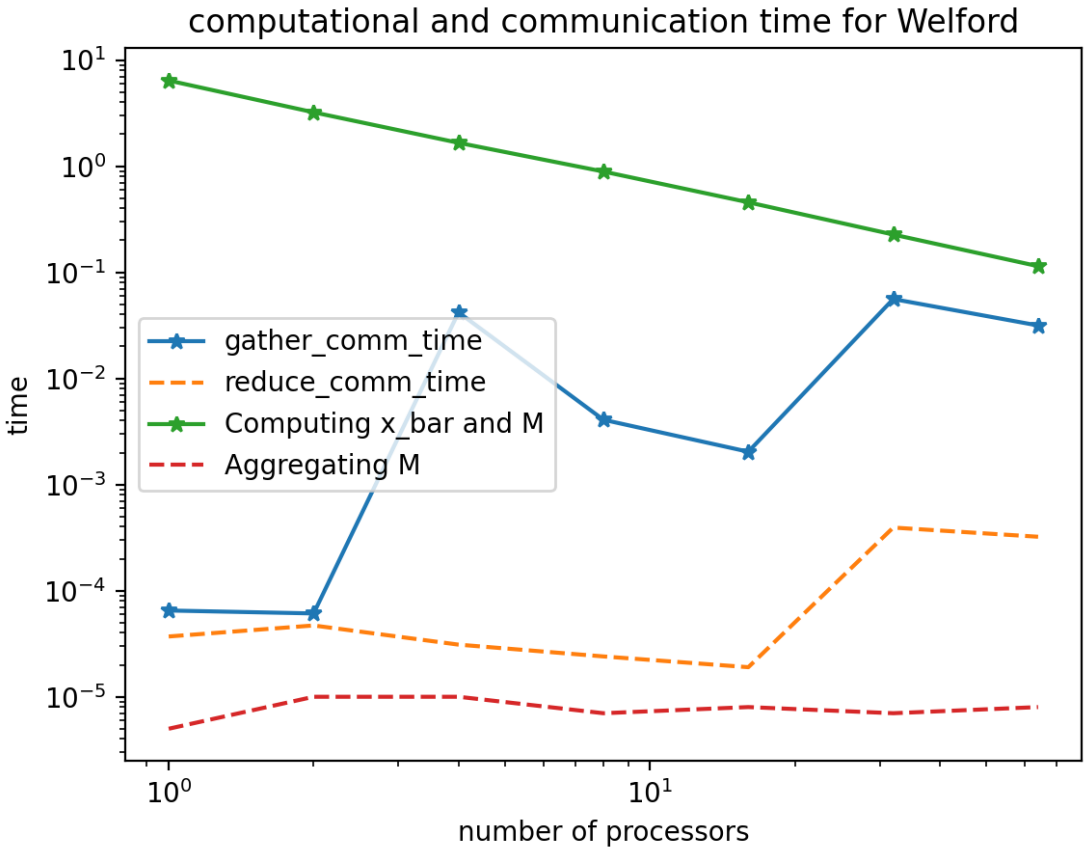
Mastery question

In [31]:

```
#Uploading task1 data for mastery question
d5 = pd.read_csv('task1_mastery.csv')
#Uploading task1 data for mastery question
d6 = pd.read_csv('task3_mastery.csv')

#ploting different computational and communication time
figure(7)
loglog(d1.nproc,d5.broadcast_comm_time,'*-',label = 'broadcast_comm_time')
#loglog(d5.nproc,d5.allreduce,'*-',label = 'broadcast_comm_time')
loglog(d1.nproc,d5.reduce_comm_time,'*--',label = 'reduce_comm_time')
loglog(d1.nproc,d5.comput_timer1,'*-',label = 'computing $N_{loc}$ and $x_{loc}$
loglog(d1.nproc,d5.comput_timer2,'*--',label = 'computing $\mu_{loc}$')
loglog(d1.nproc,d5.comput_timer3,label = 'computing $\sigma_{loc}$ ')

title('Computational and communication time for naive')
xlabel('number of processors')
ylabel('time[s]')
legend()


figure(8)
```

```
loglog(d6.nproc,d6.gather_comm_time,'*-',label = 'gather_comm_time')
loglog(d6.nproc,d6.reduce_comm_time,'--',label = 'reduce_comm_time')
loglog(d6.nproc,d6.comput_timer1,'*-',label = 'Computing x_bar and M')
loglog(d6.nproc,d6.comput_timer2,'--',label = 'Aggregating M')
title('computational and communication time for Welford')
xlabel('number of processors')
ylabel('time')
legend()

# Comparing computation and communication time for both naive and Welford algori
figure(9)
loglog(d6.nproc,d6.communication_time,'*-',label = 'communication "Welford"')
loglog(d6.nproc,d6.computational_time,'*--',label = 'computational "Welford"')
loglog(d6.nproc,d5.communication_time,'*-',label = 'communication "naive"')
loglog(d6.nproc,d5.computational_time,'*--',label = 'computational "naive"')
title('Computation and communication time for both naive and Welford')
xlabel('number of processors')
ylabel('time')
legend()
grid()
show()
```





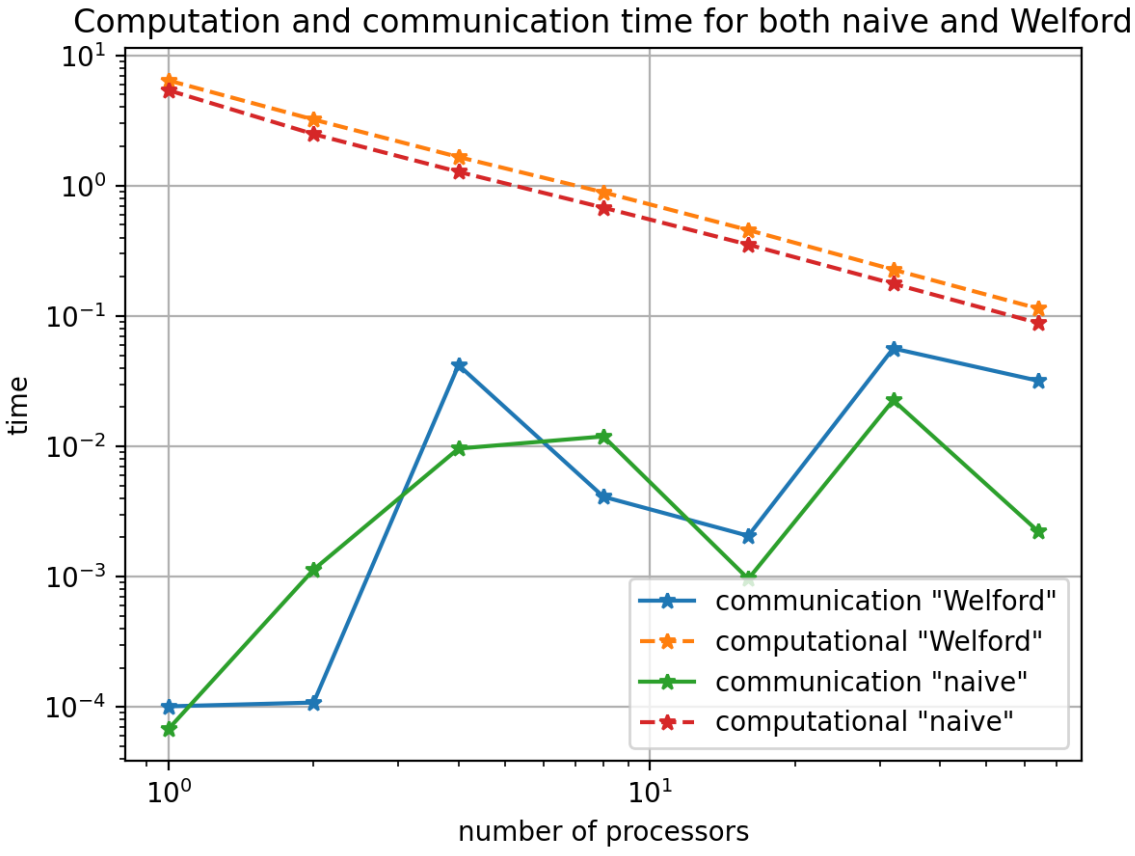


Figure 9 show that the computation times for the strong scaling experiment in both the naive

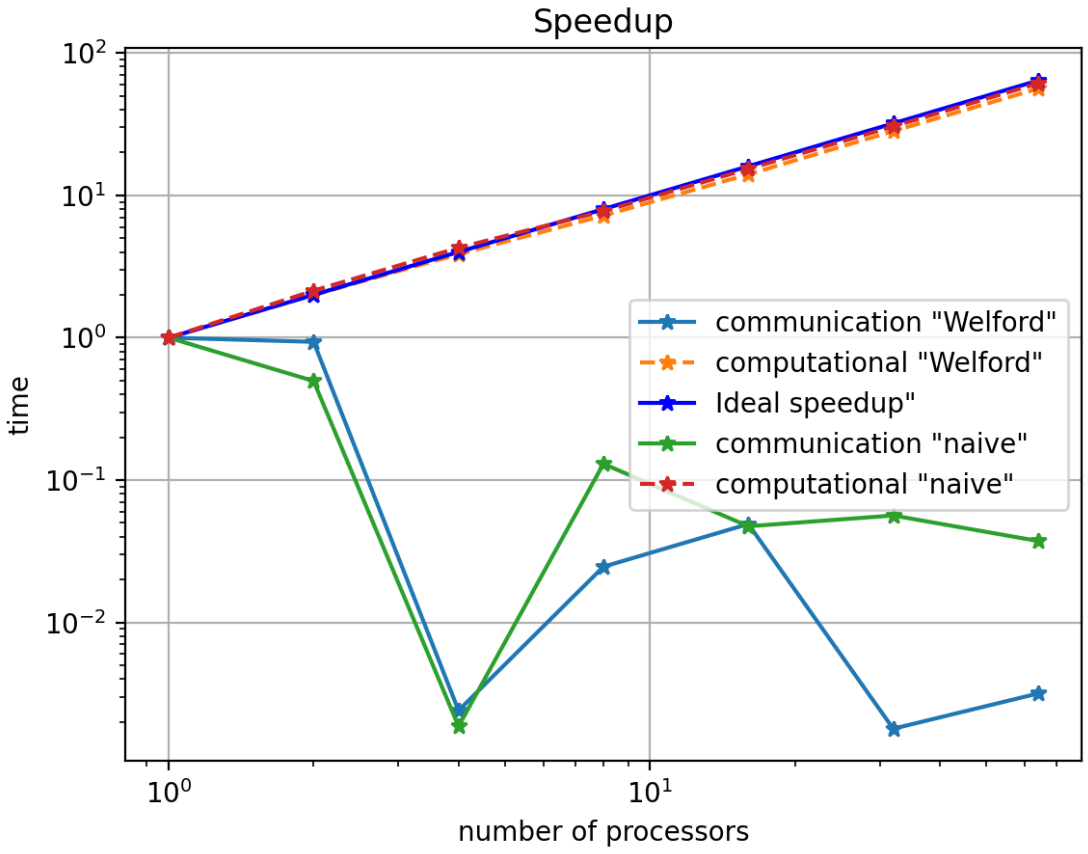
and Welford's programs decrease linearly with the increase in number of processors. However, the computation time for the Welford's program is slightly higher than that of the naive program and this is because of the computations for $\bar{x}$ and M which take up more time than that for aggregating M as shown in figure 8 above. The computations for N_{loc} and x_{loc} in the naive program take up more time than the other computations and this can be seen in figure 7 above.

The communication times for both the naive and Welford programs have increasing trends as the number of processors increase although the communication time for the Welford program is higher than that of the naive for most number of processors.

The communication time for the naive program increases for number of processors 2 and 4 with a sharp increase at 4 processors and then a sharp decrease for 8 processors and a slight increase thereafter. The communication time for the naive program is mostly affected by the allreduce comunication time as shown in the figure 7 above.

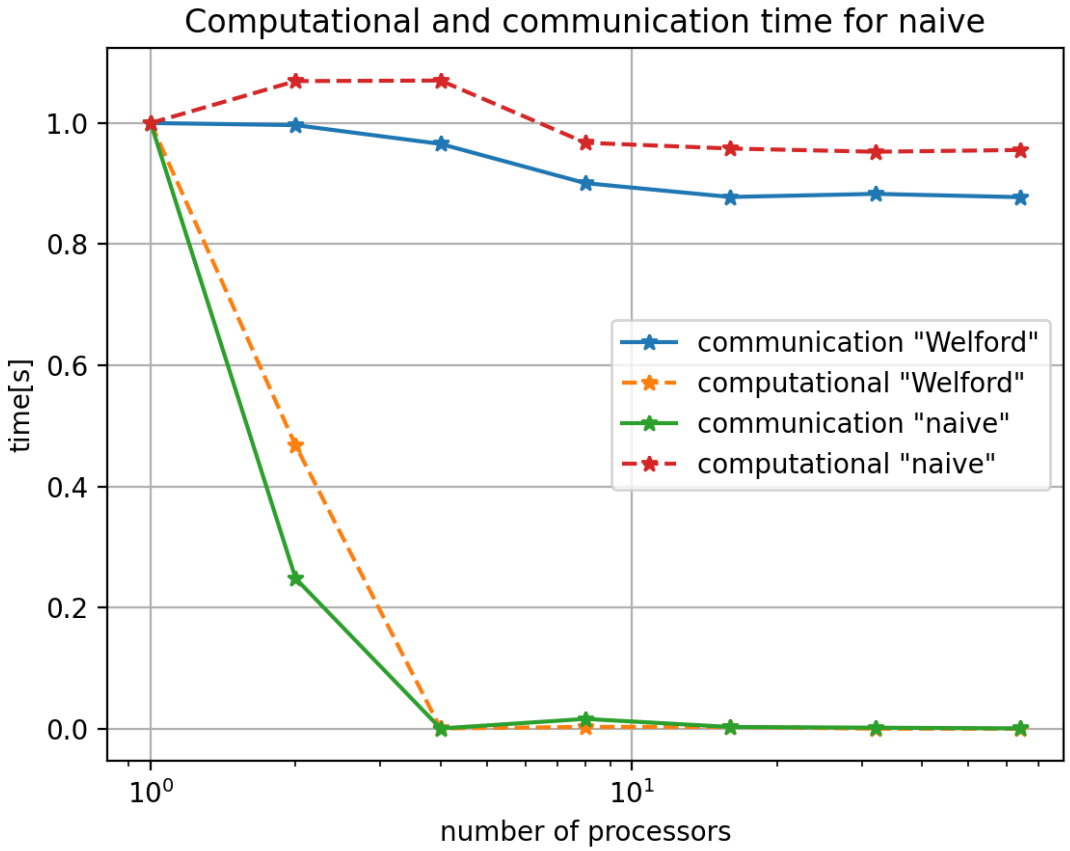
The communication time for the Welford's program increases for number of processors upto 8 with a sharp decrease at 16 processors and then a sharp increase for 32 processors and a decrease for 64 processors. The communication time for the Welford's program is mostly affected by the gather comunication time as shown in the figure 8 above.

In [9]:
```
#Calculating speedup
speedup_comm_naive = d5.communication_time[0]/d5.communication_time
speedup_comp_naive = d5.computational_time[0]/d5.computational_time
speedup_comm_Welford = d6.communication_time[0]/d6.communication_time
speedup_comp_Welford = d6.computational_time[0]/d6.computational_time
figure(10)
loglog(d6.nproc,speedup_comm_Welford,'*-',label = 'communication "Welford"')
loglog(d6.nproc,speedup_comp_Welford,'*--',label = 'computational "Welford"')
loglog(d6.nproc,d6.nproc,'b*-',label = 'Ideal speedup"')
loglog(d6.nproc,speedup_comm_naive,'*-',label = 'communication "naive"')
loglog(d6.nproc,speedup_comp_naive,'*--',label = 'computational "naive"')
title('Speedup')
xlabel('number of processors')
ylabel('time')
legend()
grid()
show()
```



In [10]:

```
#Calculating efficiency
efficiency_comm_naive = speedup_comm_naive/d6.nproc
efficiency_comp_naive = speedup_comp_naive/d6.nproc
efficiency_comp_Welford  = speedup_comp_Welford /d6.nproc
efficiency_comm_Welford  = speedup_comm_Welford /d6.nproc

figure(11)
semilogx(d6.nproc,efficiency_comp_Welford ,'*-',label = 'communication "Welford"
semilogx(d6.nproc,efficiency_comm_Welford,'*--',label = 'computational "Welford"
semilogx(d6.nproc,efficiency_comm_naive,'*-',label = 'communication "naive"')
semilogx(d6.nproc,efficiency_comp_naive ,'*--',label = 'computational "naive"')
title('Efficiency')
xlabel('number of processors')
ylabel('efficiency')
legend()
grid()
show()
```


Computational and communication time for naive
time[s]
number of processors
communication "Welford"
computational "Welford"
communication "naive"
computational "naive"
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1

In [ ]: