Project_1

Name: Brian KYANJO

Class: ME 571

Date: Febraury, 18th 2021

In [1]:
```
%matplotlib notebook
%pylab
```

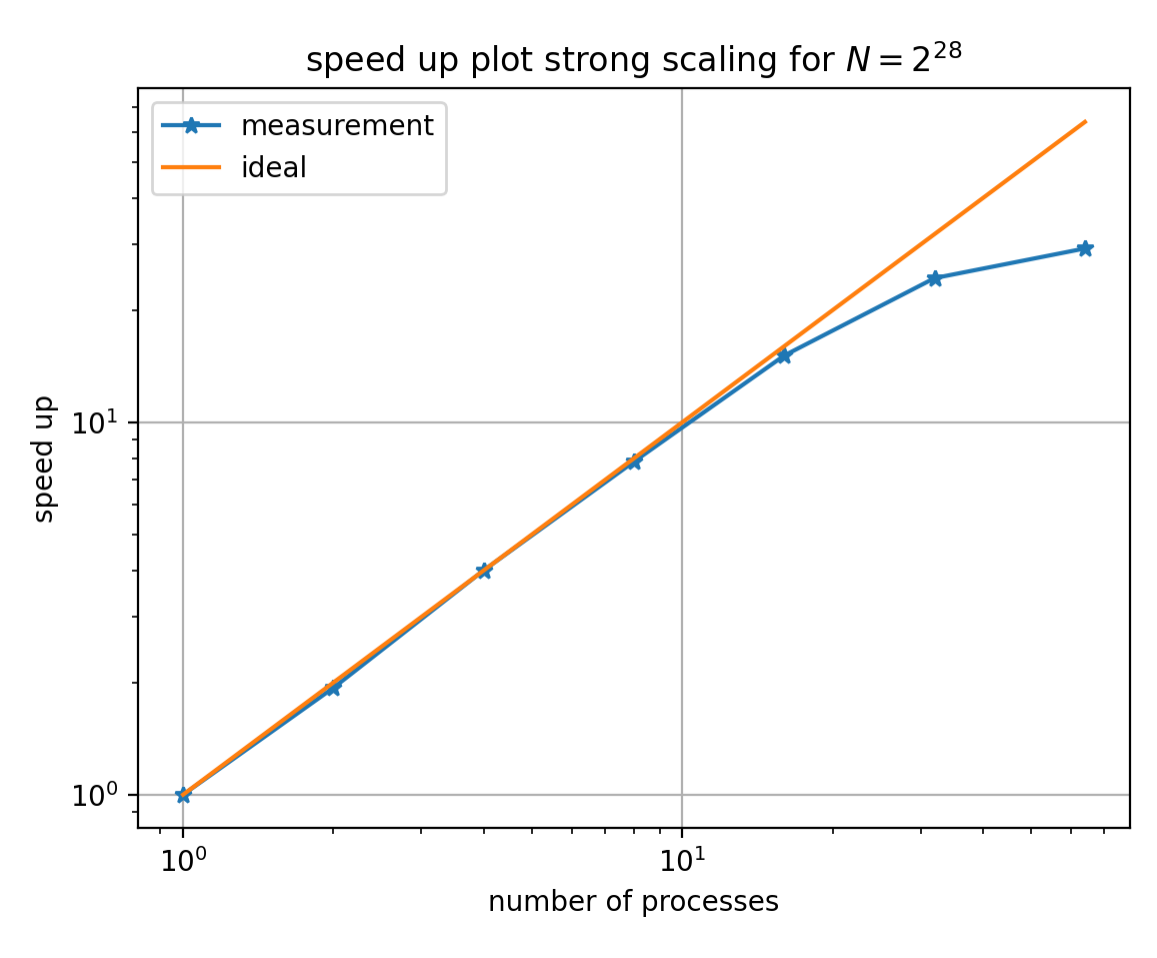
```
Using matplotlib backend: nbAgg
Populating the interactive namespace from numpy and matplotlib
```

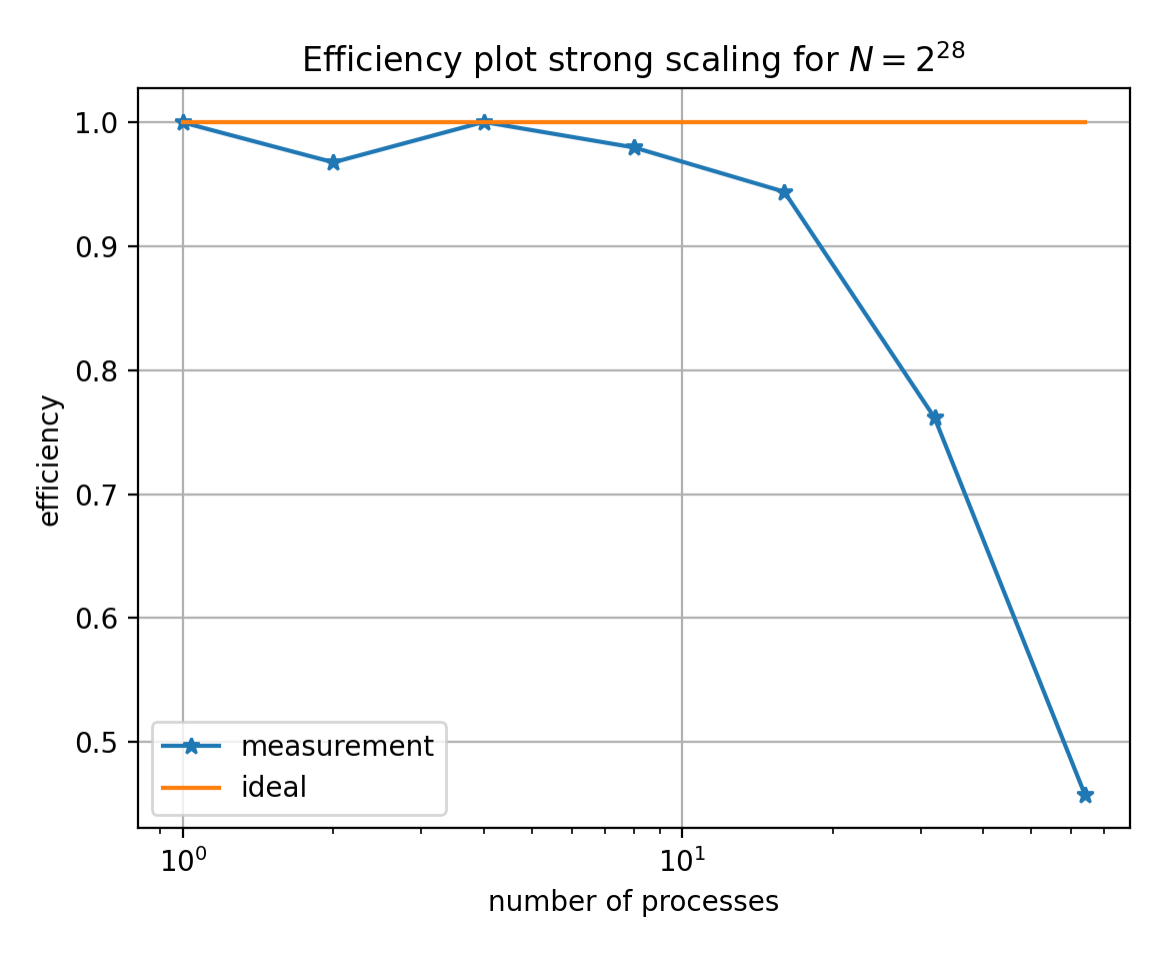
Task 1

1. Run a strong scaling experiment using nproc=1,2,4,8,16,32,64 and problem size $N = 2^{28}$. In your write-up, show the speedup and efficiency plots. Comment on what is happening. How does the parallel efficiency change with the number of processes?

In [2]:
```
#strong scaling
# number of processors
np = array([1,2,4,8,16,32,64])

#serial time
T1 = 8.902195

#parallel time
Tp = array([8.902195,4.598994,2.224560,1.135647,0.589346,0.365254,0.304099])

#speed up
s = T1/Tp

#efficiency plot
eff = s/np

e = s/s
```

In [3]:
```
figure(1)
loglog(np,s,"-*",label='measurement')
loglog(np,np,label='ideal')
ylabel("speed up")
grid()
xlabel("number of processes")
title("speed up plot strong scaling for $N=2^{28}$")
legend()
figure(2)
semilogx(np,eff,"-*",label='measurement')
semilogx(np,e,label='ideal')
title("Efficiency plot strong scaling for $N=2^{28}$")
ylabel("efficiency")
xlabel("number of processes")
grid()
legend()
show()
```





In figure 1, we obtain a linear speedup as the number of processors increases, up to a point when it trends off with the ideal measurement. This means, that the process is perfectly strongly scaled for a given number of processors, as the number grows big, scalability in turns of speedup drops

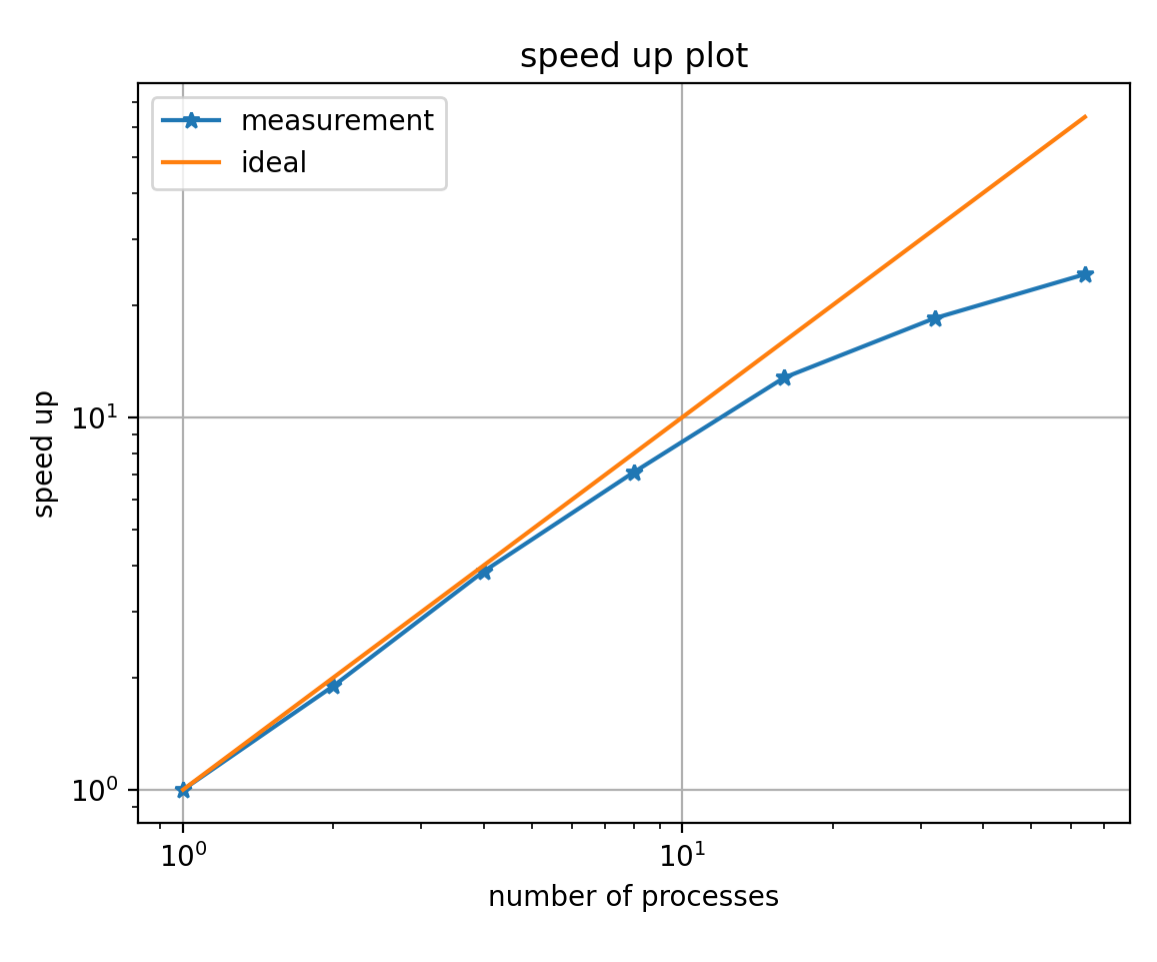
In figure 2, The Efficiency drops by larger percentages, as the number of process increases to below 50%, This is because as the number of processors increases, more time is spent on communication with different processors, as the program distributes chunks of data, to different processors.

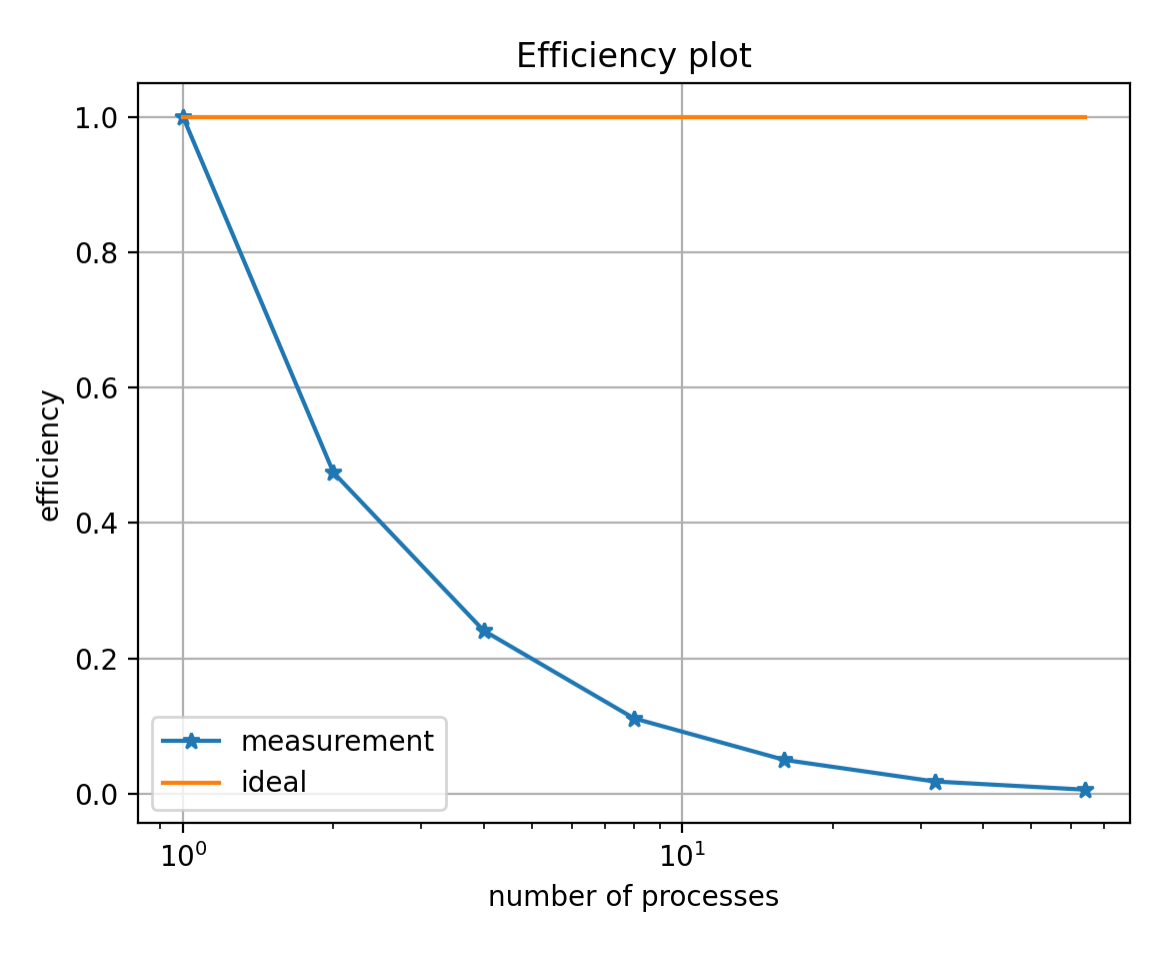
Parallel efficiency decrease with increase in number of processors

2. Run a weak scaling experiment using the same number of processors, starting with nproc=1 and $N = 2^{22}$ and doubling the problem size each time you double the number of processes. In your write-up show the weak scaling efficiency plot and comment on it.

In [4]:
```
#weak scaling
#serial time
Tw1 = 0.116381

#parallel time
Twp = array([0.116381,0.122599,0.120756,0.130732,0.145758,0.201987,0.307796])
#speed up
sw1 = np*Tw1/Twp
sw = Tw1/Twp

#efficiency plot
effw = sw/np
e = sw/sw
```

In [5]:
```
figure(3)
loglog(np,sw1,"-*",label='measurement')
loglog(np,np,label='ideal')
ylabel("speed up")
xlabel("number of processes")
legend()
grid()
title("speed up plot")
figure(4)
semilogx(np,effw,"-*",label='measurement')
semilogx(np,e,label='ideal')
title("Efficiency plot")
ylabel("efficiency")
xlabel("number of processes")
grid()
legend()
show()
```





In figure 3, as the problem size doubles with increasing number of processors, speedup raises almost linearly, this means that the total work performed varies linearly with the number of processors. Obtain that linearity i used Gustafson-s law, by scaling the speedup formular with the number of processors.

In figure 4, the parallel efficiency gradually drops to 0%, as the problem size doubles with increasing number of processors, This means that the system performs badly as the problem size and number of processors increases.

Task 2

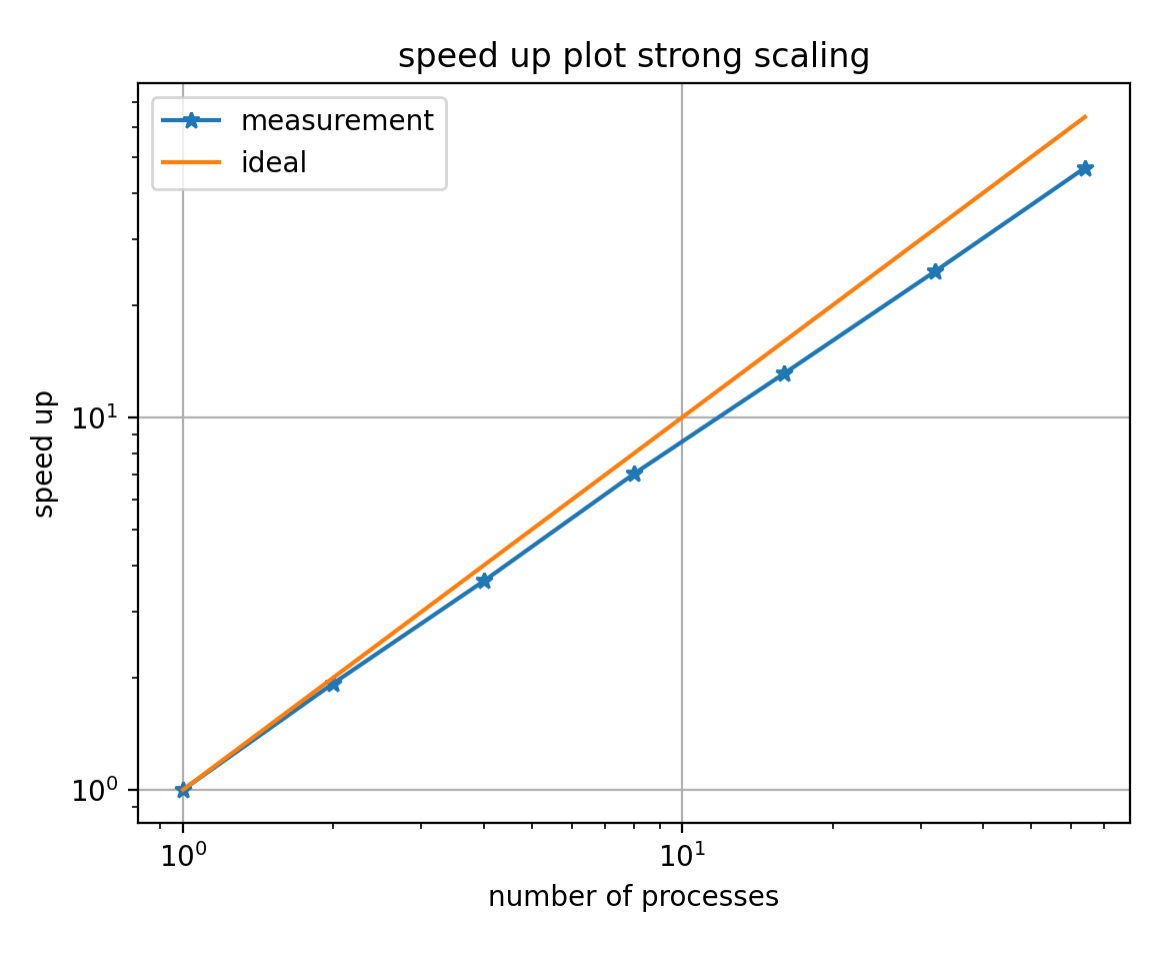
The serial code gives the same results for standard deviation and mean as in task 1.

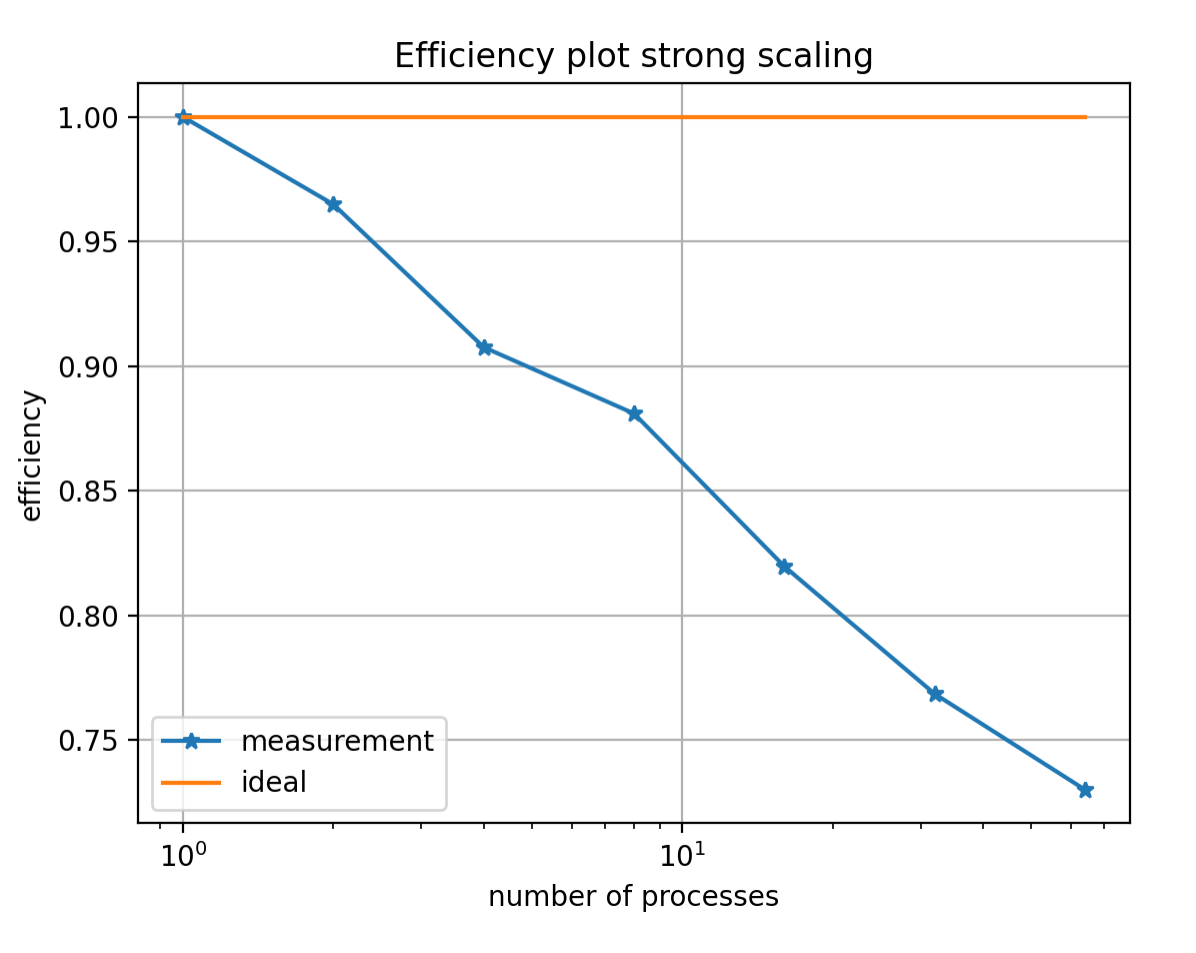
Task 3

3. Create strong scaling experiment using nproc=1,2,4,8,16,32,64 and problem size $N = 2^{28}$. In your write-up, show the speedup and efficiency plots and comment on how they compare to strong scaling in Task 1.

In [6]:
```
#strong scaling
# number of processors
np = array([1,2,4,8,16,32,64])

#serial time
T1 = 10.847608

#parallel time
Tp = array([10.847608,5.621049,2.988099,1.539187,0.827202,0.441136,0.232195])

#speed up
s = T1/Tp

#efficiency plot
eff = s/np
```

In [7]:
```
figure(5)
loglog(np,s,"-*",label='measurement')
loglog(np,np,label='ideal')
legend()
grid()
ylabel("speed up")
xlabel("number of processes")
title("speed up plot strong scaling")
figure(6)
semilogx(np,eff,"-*",label='measurement')
semilogx(np,e,label='ideal')
title("Efficiency plot strong scaling")
ylabel("efficiency")
legend()
grid()
xlabel("number of processes")
show()
```





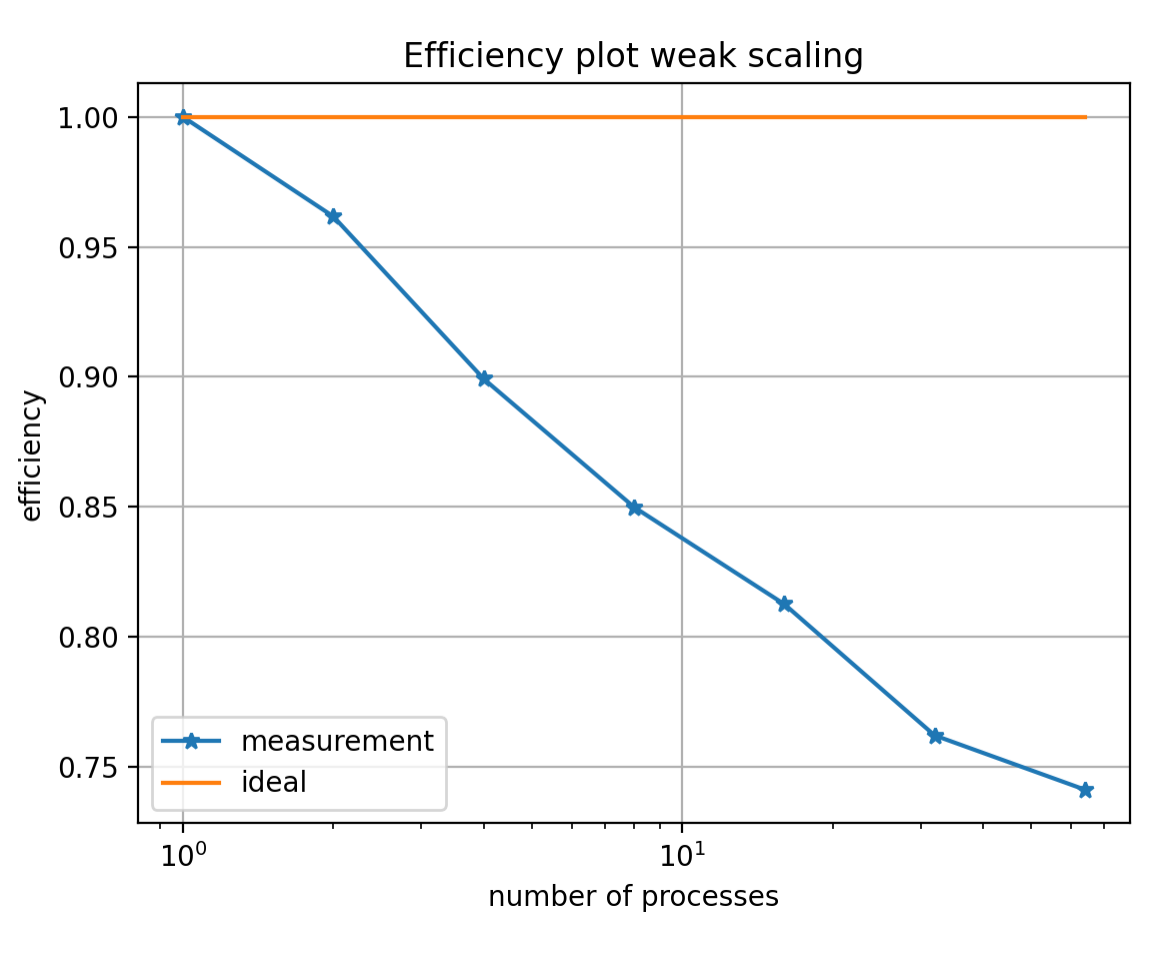
In this run we obtain a linear speedup which implies that the parallel code has been perfectly strongly scaled compared to the one in Task 1, which was almost linear for the first four processors: 1,2,4,8, then lost on.

In this run, the parallel efficiency, drops almost propotional to the number of processors, as the number of processes increase, by a drop of 5% of the previous one, while in task one strong scaling, as the number of processors increased, the efficiency droped unpredictably, by a large factor to below 50% for the last 64 processors.

4. Run a weak scaling experiment using the same number of processors, starting with nproc=1 and $N = 2^{22}$ and doubling the problem size each time you double the number of processes. In your write-up show the weak scaling efficiency plot and comment on how it compares to weak scaling in Task 1.

In [8]:
```
#weak scaling
# number of processors
np = array([1,2,4,8,16,32,64])

#serial time
T1 = 10.753679

#parallel time
Tp = array([10.753679,5.591031,2.989730,1.581634,0.827177,0.440994,0.226710])

#speed up
s = T1/Tp

#efficiency plot
eff = s/np
```

In [9]:
```
figure(7)
loglog(np,s,"-*",label='measurement')
loglog(np,np,label='ideal')
legend()
ylabel("speed up")
xlabel("number of processes")
grid()
title("speed up plot weak scaling")
figure(8)
semilogx(np,eff,"-*",label='measurement')
semilogx(np,e,label='ideal')
title("Efficiency plot weak scaling")
ylabel("efficiency")
xlabel("number of processes")
legend()
grid()
show()
```



In figure 7, speed up varies linearly with increasing number of processors as problem size doubles, this explains that the parallel system was perfectly scaled. compared to Task1 in which linearity droped at large number of processors.

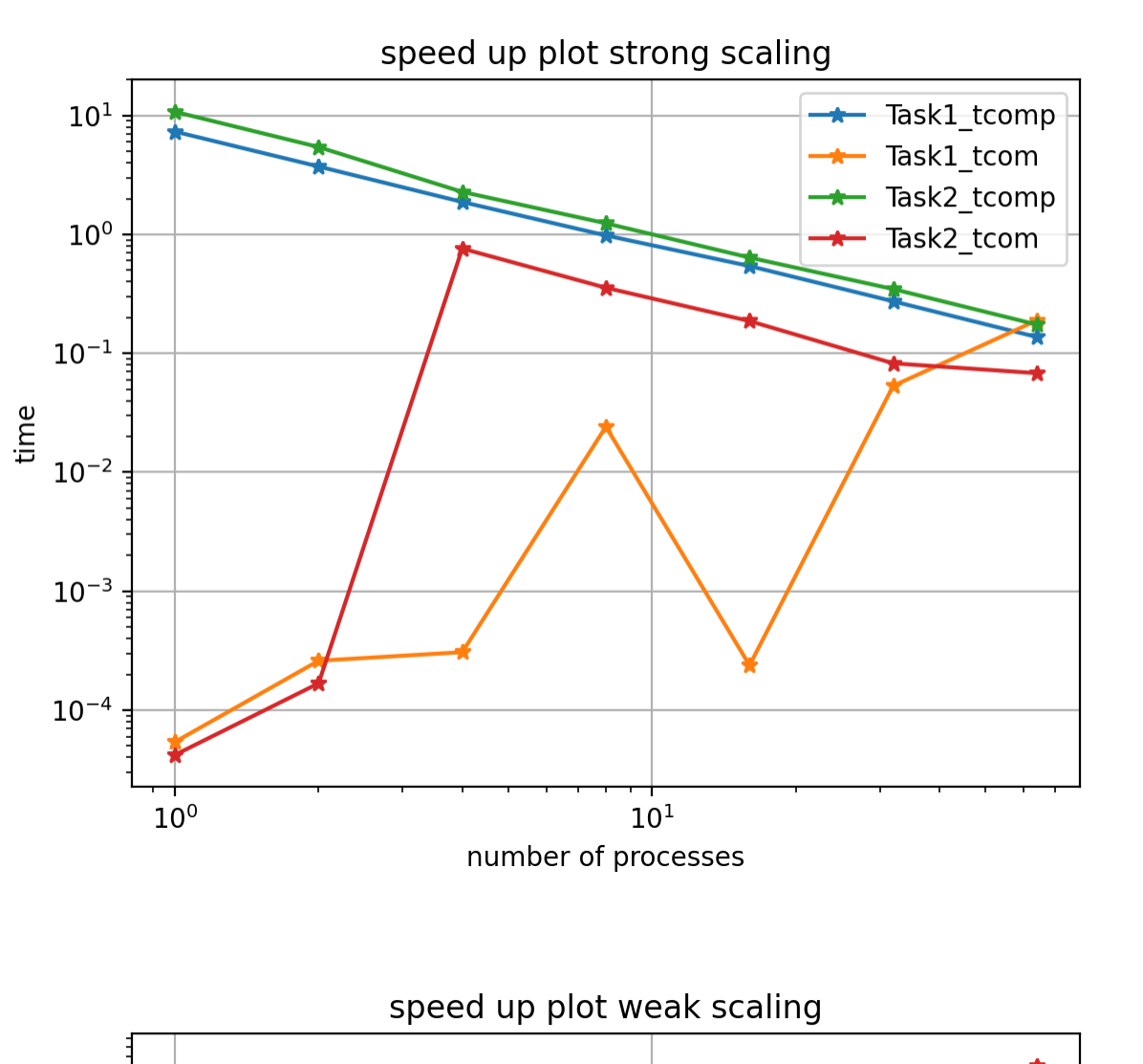
In figure 8, parallel efficiency drops linaerly with increasing number of processors, as the problem size doubles. This implies that this program has been made cost optimal, hence the system sustained efficiency. However, in Task 1, figure 4, the system fails to keep efficiency as problem size and number of processors increases.

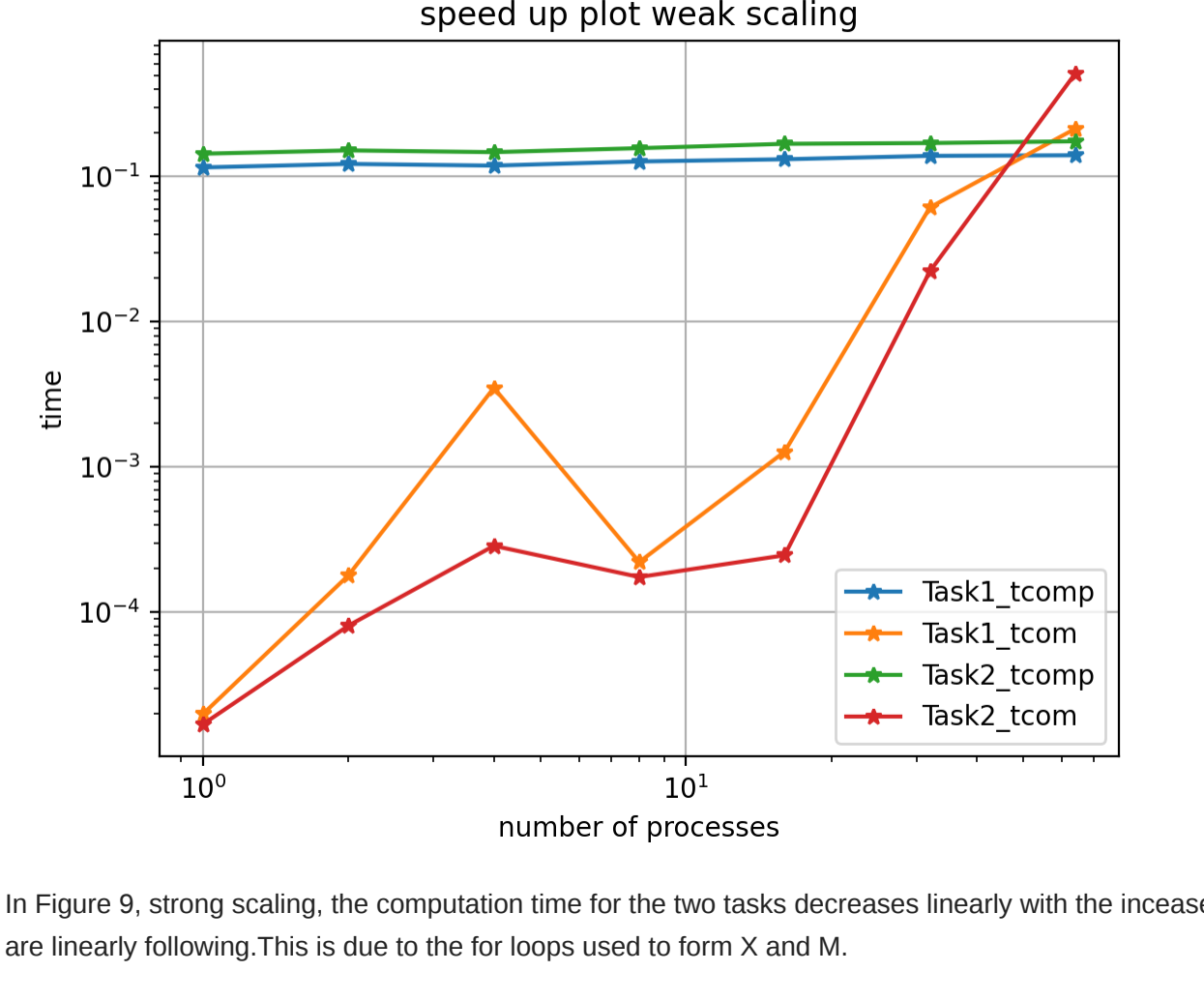
Mastery

Time the computation and communication time for both algorithms separately. Compare them in one plot. What conclusions can you draw from the plot?

In [10]:
```
# number of processors
np = array([1,2,4,8,16,32,64])

#task1_strong
time_comput_task1_s = array([7.319304,3.722973,1.874252,0.980094,0.541893,0.273862,0.137116])
time_comm_task1_s = array([0.000054,0.000260,0.000307,0.024173,0.000238,0.053267,0.189134])

#task1_weak
time_comput_task1_w = array([0.115713,0.122275,0.119035,0.127126,0.131627,0.138919,0.140253])
time_comm_task1_w = array([0.000020,0.000180,0.003513,0.000222,0.001275,0.061398,0.213632])

#task3_strong
time_comput_task3_s = array([10.742264,5.427192,2.277480,1.239713,0.638648,0.347072,0.174478])
time_comm_task3_s = array([0.000042,0.000168,0.760382,0.356010,0.186826,0.082397,0.068114])

#task3_weak
time_comput_task3_w = array([0.143788,0.151476,0.147318,0.156957,0.168179,0.170289,0.175017])
time_comm_task3_w = array([0.000017,0.000081,0.000287,0.000175,0.000247,0.022298,0.513266])
```

In [11]:
```
figure(9)
loglog(np,time_comput_task1_s,"-*",label='Task1_tcomp')
loglog(np,time_comm_task1_s,"-*",label='Task1_tcom')
loglog(np,time_comput_task3_s,"-*",label='Task2_tcomp')
loglog(np,time_comm_task3_s,"-*",label='Task2_tcom')
legend()
ylabel("time")
xlabel("number of processes")
grid()
title("speed up plot strong scaling")
figure(10)
loglog(np,time_comput_task1_w,"-*",label='Task1_tcomp')
loglog(np,time_comm_task1_w,"-*",label='Task1_tcom')
loglog(np,time_comput_task3_w,"-*",label='Task2_tcomp')
loglog(np,time_comm_task3_w,"-*",label='Task2_tcom')
legend()
ylabel("time")
xlabel("number of processes")
grid()
title("speed up plot weak scaling")
show()
```





In Figure 9, strong scaling, the computation time for the two tasks decreases linearly with the inceases in number of processors, however, much time is spent on computation in task2 more than task1, but they are linearly following.This is due to the for loops used to form X and M.

The Communication time for task2 increases as the number of processors increases, as more chunks of data is sent to different processors at the time, up to when it scales almost linearly with the both computation time for the two tasks. Overall less time is spent on communication for task1 compared to task2, as seen fro the graph above, and this is due to fact that MPI_Gather and MPI_Reduce take more communication time than broadcasting, allreduce and reduce used in Task1, as the number of processors increase on a fixed problem size.

In Figure 10, weak scaling, the computation time for the two tasks is independent of the number of processors (is constant), even though, again Task2 uses more computational time than task1. This is due to the for loops used to form X and M.

As the problem size doubles with increasing number of processors, the communication time for Task2 keeps on increasing as the same as for Task1, this is because more time is spent on communicating to different processors, as the problem size doubles at the same time increasing the number of processors.

If the scaling plots in Task 3 do not look much different from Task 1, try a significantly smaller (or significantly larger) problem size to see whether the problem size affects the strong and weak parallel efficiency.

Plots in Task 3 look much different from plots in Task 1.