Homework 7

```
import numpy as np
```

Question 1

Given an algorithm where the fraction that the program is serial (f) is .15, what is the speedup on 2, 4, 8, 16 and an infinite number of processors. You may ignore communication costs and the problem size is fixed.

Using Amdahl's law:

$$\Psi(P) \leq \frac{1}{f + \frac{(1-f)}{P}}$$

```
f = 0.15
P = np.array([2,4, 8, 16, np.inf])
speedup = np.array([ 1/(f + (1-f)/p) for p in P])
print("speedup:")
print(*P)
print(*speedup)
```

```
speedup:
2.0 4.0 8.0 16.0 inf
1.7391304347826089 2.7586206896551726 3.9024390243902443 4.923076923076923 6.666666666666
```

Question 2

What is the efficiency at 2, 4, 8 and 16 processes?

Efficiency is given by:

$$\epsilon(P) = \frac{\Psi(P)}{P}$$

```
e = [s/p for s,p in zip(speedup, P)]
print(*e)
```

```
0.8695652173913044 0.6896551724137931 0.48780487804878053 0.3076923076923077 0.0
```

p	2	4	8	16	∞
speedup	1.7391304347826089	2.7586206896551726	3.9024390243902443	4.923076923076923	6.666666666666667

p	2	4	8	16	∞

Question 3

Solve problem 1 for a non-infinite number of processors where the serial portion s of a parallel execution is .15 What is the efficiency of each?

Using G-B Law

$$\Psi(n, P) = s + (1 - s)P$$
$$\epsilon(n, P) = \Psi(n, P)/P$$

```
s = 0.15
P = np.array([2, 4, 8, 16])
speedup_gbl = np.array([ s+ (1-s)*p for p in P])

print("speedup (G-B Law):")
print(*P)
print(*speedup_gbl)
```

```
speedup (G-B Law):
2 4 8 16
1.8499999999999999 3.55 6.95 13.75
```

Question 4

What is the efficiency of each?

```
efficiency_gbl= speedup_gbl/P
print(*efficiency_gbl)
```

```
0.9249999999999999 0.8875 0.86875 0.859375
```

p	2	4	8	16
speedup	1.8499999999999999	3.55	6.95	13.75
efficiency	0.9249999999999999	0.8875	0.86875	0.859375

Question 5

Given a 1000 machine cluster, what must s and f be to obtain an efficiency of 80%?

Using G-B Law and Amdahl's equations:

$$\epsilon(n,P) = \Psi(n,P)/P \Rightarrow \Psi(n,P) = P\epsilon(n,P)$$
$$\Psi(n,P) = P + (1-P)s \Rightarrow s = (\Psi(n,P) - P)/(1-P)$$
$$\frac{\phi}{\sigma} = (1/f - 1) = P(1/s - 1) \Rightarrow f = \frac{1}{1 + P(1/s - 1)}$$

```
P = 1000
e = 0.8
speedup_q5 = e*P
s = (speedup_q5 - P)/(1-P)
f = 1./(1.+ P*(1./s -1))
print("speedup: %f"%(speedup_q5))
print("s: %.6f"%(s))
print("f: %.6f"%(f))
```

```
speedup: 800.000000
s: 0.200200
f: 0.000250
```

Question 6

Given the following two tables, what can you say about the scalability of the two programs that yield these results? If the scaling is poor, is it a result of Amdahl's Law or an increasing degree of sequential execution/overhead in the program? If the scaling is good, why do you think it is?

Table 1

p	2	4	8	16	32	64
Ψ	1.8	3.6	7.2	14.4	28.8	57.6
e	0.111	0.037	0.0158	0.007	0.004	0.002

Comment on Scalabilty

Scaling is good in this case because as the number of processors increases the speedup increases and the amount of work done by each processor is also decreasing (decreasing efficiency). This means that our code can scale up to any number of processors without reaching a resource bottleneck.

Table 2

p	2	4	8	16	32	64
Ψ	1.9	3	4	5	5.5	5.7
e	0.053	0.111	0.143	0.147	0.155	0.162

Comment on Scalabilty

The scalabilty is bad for this case because the efficiency is increasing with the speedup. This means that, more work is done by each processor due to increased communication overhead and therefore for some number of processors this scaling strategy would completely use up all resources and would not be able to scale up further.
