Question 1

In the PDF report, please report the wall-clock runtime of your parallel program on the four tests using 1, 2, 4, and 8 threads and make a speedup table and an efficiency table. Please comment on the scalability of your parallel code (strongly scalable vs weakly scalable). If you fail to write a correct and scalable algorithm before the deadline, please explain the likely causes in the report and you will receive points for the error analysis.

ANSWER: We can see that this problem is strongly scalable due to the efficiency maintaining itself or slightly improving when the problem size stays the same and the processor count increases. The efficiency is also still maintained when across the main diagonal of the chart when processor count has a correlated increase with the problem size. Looking at the serial chart, we can see that there is a drastic increase in the runtime when the problem size increases and our performance issue is solved when we implement parallelism, which just goes on to show the true power of parallel code.

Runtime	Test 1	Test 2	Test 3	Test 4
P = 1	4.760074	10.67946	22.993956	98.793573
P = 2	2.43292	5.832469	12.102908	47.948518
P = 4	1.199255	2.562233	5.808676	24.895725
P = 8	0.595387	1.341086	2.731035	12.42197

Speedup	Test 1	Test 2	Test 3	Test 4
P = 1	1	1	1	1
P = 2	1.9565	1.8310	1.8999	2.0604
P = 4	3.9692	4.1680	3.9586	3.9683
P = 8	7.9949	7.9633	8.4195	7.9531

Efficiency	Test 1	Test 2	Test 3	Test 4
P = 1	1	1	1	1
P = 2	0.9782	0.9155	0.9499	1.0302
P = 4	0.9923	1.0420	0.9896	0.9921

P = 8	0.9994	0.9954	1.0524	0.9941

Question 3

In the report, please describe the algorithm used to solve the problem, including how the data was distributed to the processors. The program should be run using 1, 2, 4, and 8 threads on data of 4 different sizes. For each run, you should time the encryption computation. The report should include 4X4 tables for the computation time, speedup, and efficiency for all these runs. Is your program strongly scalable or weakly scalable and why?

ANSWER: The algorithm used to solve the problem is an implementation of the caesar cipher which is a "shifting" cipher that inputs certain letters depending on the key. Below is a snippet of the algorithm used.

```
#pragma omp parallel for num_threads(threads)
   for (int i = 0; i<lSize; i++) {
       encrypted_buffer[i] = buffer[i] + key % 256;
   }
   if (DEBUG) printf("Values encypted! \n");
```

This algorithm will distribute the plaintext almost evenly among all the threads using the "#pragma" statement. These threads will then each perform the operation of encrypting the plaintext using the caesar cipher. Each thread will insert into different parts of the buffer, thus there will not be any collision or need for a critical section. The implementation is "embarssingly parallel". We only needed to add in the single line "#pragma omp parallel for num_threads(threads)" in order to achieve effective parallelism.

Further, looking at both the Speedup and Efficiency table, we can see that the algorithm is strongly scalable. There is only a very minor dropoff in the efficiency when the processor count is increased and the problem size stays the same (the efficiency is essentially maintained when the processor count increases and the problem size stays the same). We can also see that the efficiencies are almost identical between the different problem sizes and processor counts. Implementing parallelism does drastically increase the speedup (through around a 7 fold increase with 8 processors when compared to 1).

Runtime	Test 1	Test 2	Test 3	Test 4
P = 1	0.065293	0.130533	0.299523	0.676419
P = 2	0.033018	0.066146	0.152014	0.342093
P = 4	0.016612	0.033094	0.076464	0.171317

P = 8	0.008506	0.016739	0.038224	0.086198

Speedup	Test 1	Test 2	Test 3	Test 4
P = 1	1	1	1	1
P = 2	1.9775	1.9734	1.9704	1.9773
P = 4	3.930471948	3.9443	3.9172	3.9483
P = 8	7.67611098	7.7981	7.8360	7.8473

Efficiency	Test 1	Test 2	Test 3	Test 4
P = 1	1	1	1	1
P = 2	0.9887	0.9867	0.9852	0.9886
P = 4	0.9826	0.9860	0.9793	0.9871
P = 8	0.95951	0.9747	0.9795	0.9809