1. (*15 pts)* Amdahl's Law: Mattson et al. (our book) expresses Amdahl's law as:

$$S(P) = \frac{1}{\gamma + \frac{1-\gamma}{P}}$$

in which γ is the serial fraction of the program, P is the number of processors, and $S(P)$ is the possible speedup.

   (a) What is the maximum possible speedup on 50 processors if 5% of a computation is serial and 95% parallel? You may round to an integer.

$$S(P) = \frac{1}{0.05 + \frac{1-0.05}{50}} = \frac{1}{0.069} = 14.49$$

   So, we accepted 14 or 15.

   (b) Describe the implications of Amdahl's law; i.e., what does Amdahl's law state in English?

   The maximum possible speedup of a computation is restricted to improvements to the parallelizable portion of a computation. The serial portion of the computation cannot be improved by adding more resources.

2. (*15 pts*) Synchronized Blocks in Java

   (a) Make the inner block of the following function *synchronized* with respect to the specific object (self object) in which it is called.

```
public void UpdateSharedState ( .... )
{
   // The following block of code needs to be synchronized

   **synchronized ( this )**      // This line is the answer
   {
      ....         // bunch of updates to shared variables
   }
}
```

   Note: There was some confusion that `this` is the self object. Responders put a variety of garbage in its place.

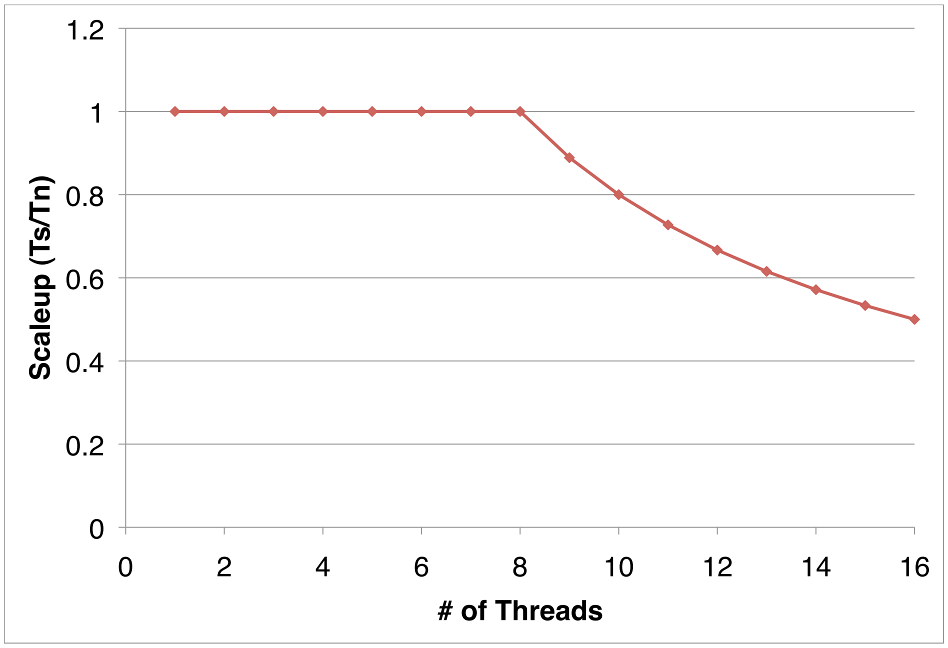
   (b) In part (a), you synchronized on the self object. One may also synchronize on the entire class. **When would one synchronize on the entire class? Why would one prefer to synchronize on a specific object when possible?** (Answer both questions.)

One would synchronize on the entire class when the block accesses data that are shared among all instances of that class, typically static members.

One prefers to synchronize on the individual object when possible to increase parallelism. Multiple threads may access the same synchronized block in different objects (from the same class) concurrently. Synchronizing on all objects of a class means that only one thread at a time can access the block among all objects.

Note: There was some confusion about the term "entire class". I would have been better if the question were phrased as "all objects of that class." Some responders took "entire class" to mean that the synchronized block covered all methods and data in the class.

3. (*20 pts*) The following chart displays the practical scaleup realized when running a parallel algorithm on a shared-memory machine with 8 physical cores. The chart shows the relative performance of running a small problem on a single thread versus running a k times larger program on k threads.



(a) What are the scaling properties of *the parallel algorithm*? (That is, how would you expect the algorithm to perform as you increased its parallel resources.)

The algorithm exhibits perfect or ideal scaleup, solving k times larger instances of the problem on k processing elements in the same amount of time. This scaleup can only be realized to the number of processing elements available.

(b) What practical scaleup was realized on this particular machine? Characterize the data in the chart.

The practical scaleup is limited by the parallel resources available. For up to 8 threads, each thread received its own core. Beyond 8 threads, threads had to share processing elements. The data indicates that this sharing introduced no additional overhead, e.g., because 16 threads decreased scale-up performance by half when compared with 8 threads.

4. (*50 pts*) Map/Reduce for Financial Data The input to the map/reduce program consists of a set of records that represent financial transactions, e.g., the transaction log of a brokerage firm. Records are of the form

$$\text{Input schema: } K = \{\}, V = (\text{symbol}, \text{purchaser}, \text{shares}, \text{price})$$

in which each value record indicates an individual sale of `shares` shares of `symbol` bought by `purchaser` at cost of `price` per share. Analysts at the brokerage firm wish to answer the question

What symbols were bought by the top purchaser?

which will produce outputs of the form

$$\text{Output data: } (\text{purchaser}, \text{symbol})$$

in which a top purchaser does the highest volume (`shares` $\times$ `price`) summed over all of that `purchaser`'s transactions. Write a multi-stage map/reduce program that answers this question. For each stage, you should **(1) specify the input (map) schema and output (reduce) schema** and **(2) describe (in words) what the function does**. **(3) You should also state the cardinality of the inputs and outputs**, e.g. "each map record produces one output record to the reducer" or "the reducer accumulates all records within a key and outputs a single record." The following example gives a sense of how you might answer this question.

Note: You may use the output of a previous map/reduce phase as shared data that is input to all mappers in a subsequent phase. If you do so, that shared data should be small, i.e. one value or one record. Not all solutions require this technique.

Example: To answer the question:

How many shares were bought of each symbol?

One might respond in the following manner.

$$\text{Input Schema: } K = \{\}, V = (\text{symbol}, \text{purchaser}, \text{shares}, \text{price})$$
$$\text{Output Schema: } K = \text{symbol}, V = \text{shares}$$

Map: The mapper performs a key transformation, outputting the `symbol` as the key and the `shares` as the value. The mapper outputs one record for each input record.

Reduce: The reducer sums the `shares` within each `symbol`. It outputs one record for each key that is the `symbol` and the `sum` of the shares.

Answer:

Phase 1:

$$\begin{aligned} \text{Input Schema: } & K = \{\}, V = (\text{symbol}, \text{purchaser}, \text{shares}, \text{price}) \\ \text{Output Schema: } & K = \text{purchaser}, V = (\text{volume} = (\text{shares} \times \text{price})) \end{aligned}$$

Map: The mapper outputs one line of output for each value input emitting the purchaser as the key. The value is the product of the number of shares and the price per share as `volume`.

Reduce: The reducer sums the `volume` over all transactions from the `purchaser` and outputs **one** output record for each key in which the `volume` is the total volume by that purchaser.

Phase 2:

$$\begin{aligned} \text{Input Schema: } & K = \text{purchaser}, V = \text{volume} \\ \text{Output Schema: } & K = \text{volume}, V = \text{purchaser} \end{aligned}$$

Map: The mapper outputs one line of output for each value input emitting the `volume` as the key. Alternatively, each mapper may produce only a single output containing the highest volume.

Reduce: The system should use only a single reducer so that all input records are sorted in the same partition. For only the highest key value, the reducer writes a single output that gives the `volume` and `purchaser` for the top purchaser.

Phase 3:

$$\begin{aligned} \text{Shared Data: } & \text{purchaser} \qquad \text{//output from Phase 2} \\ \text{Input Schema: } & K = \{\}, V = (\text{symbol}, \text{purchaser}, \text{shares}, \text{price}) \\ \text{Output Schema: } & K = (\text{purchaser}, \text{symbol}), V = \{\} \end{aligned}$$

The input is the original data. The shared data is the top purchaser found in Phase 2.

Map: The mapper searches through the input and outputs only records for which the `purchaser` matches the top purchaser in the shared data.

Reduce: The reducer eliminates duplicates only outputting once for each unique key.