1. Differences between sequential and parallel programs (from textbook):
- sequential - operations performed one at a time, so its straightforward to reasoning about correctness and performance
- parallel - many operations take place at once, so complicates reasoning about correctness and performance

a) Why is claiming that sequential programs performs operations one at a time slightly misleading?

b) Why are there so many debugs in sequential programs if it is straightforward to reasoning about their correctness?

2. What are your options if your sequential program is too slow?

i.

ii.

iii.

3. Why don't we just use a parallelizing compiler to translate our sequential program to a parallel program which utilitizes all the cores on my multi-core computer?

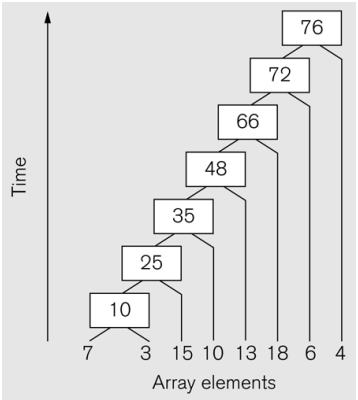
4. There is a "paradigm shift" making parallel programming conceptually different from sequential programming. Simple textbook example, summing an array x containing n elements.

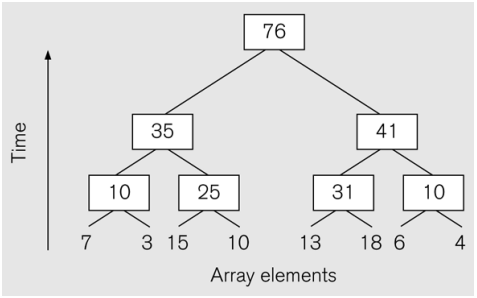
Sequential Algorithm:

```
sum = 0;
for (i = 0; i < n; i++) {
    sum = sum + x[i];
} // end for
```

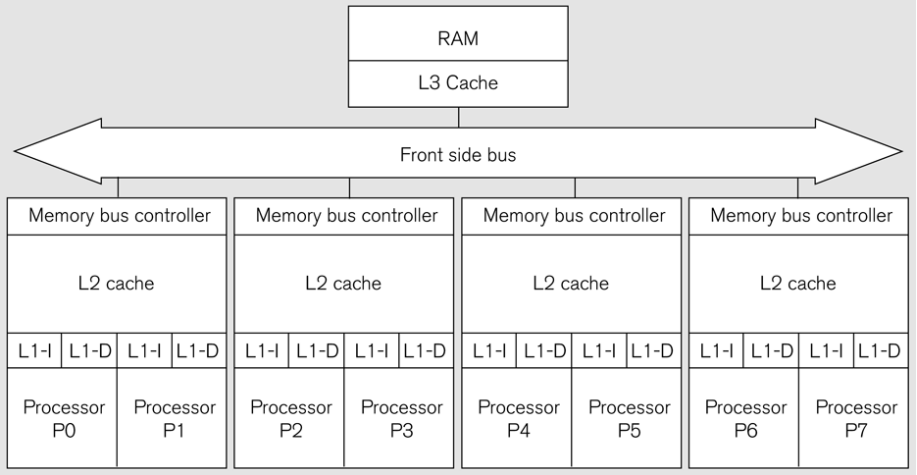
a) How long would each algorithm take?

Parallel Pair-Wise Summation Algorithm

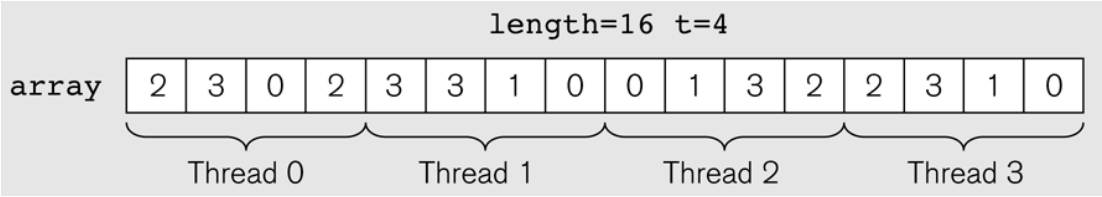




5. The textbook uses a trival problem to illustrate the complexity of parallel programming. Details:
   - Problem: count the number of 3s in an array
   - Parallel computer used: 8-cores
   - Multitheaded program
     - all threads share global memory space: program, array, and count
     - each thread has its own PC (pgm. counter) and run-time stack



- General algorithm: each of t threads will process a block of the array in parallel:



First Try at a Solution:

```
 1  int t;                /* number of threads */
 2  int *array;
 3  int length;
 4  int count;
 5
 6  void count3s()
 7  {
 8    int i;
 9    count = 0;
10    /* Create t threads */
11    for(i=0; i<t; i++)
12    {
13      thread_create(count3s_thread, i);
14    }
15
16    return count;
17  }
18
19  void count3s_thread(int id)
20  {
21    /* Compute portion of the array that this thread
         should work on */
22       int length_per_thread=length/t;
23       int start=id*length_per_thread;
24
25    for(i=start; i<start+length_per_thread; i++)
26    {
27      if(array[i]==3)
28      {
29        count++;
30      }
31    }
32  }
```

a) Why does this program give the wrong count of the number of 3s?

6. Shared variable(s) needs to updated *mutual exclusively* -- at any given time at most one thread executing the *critical section* of code which updates the shared variable.

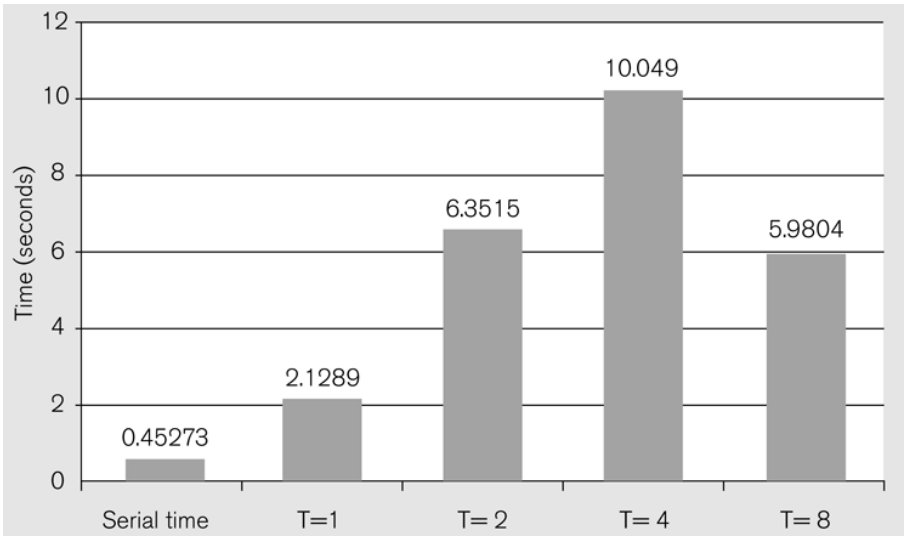
A *mutex* object can be used to provide mutual exclusion. A mutex can have two states: locked or unlocked which can be changed by the operations:

- mutex_lock - thread tries to lock the mutex for mutually exclusive access. If the mutex is unlocked, then it is locked and the thread is allowed to continue. If the mutex is locked, the thread is forced to wait.
- mutex_unlock - the mutex is unlocked and the unlocking thread continues. Unlocking the mutex causes a waiting thread to wakeup.

Second Try Code:

```
 1  mutex m;
 2
 3  void count3s_thread(int id)
 4  {
 5    /* Compute portion of the array that this thread
        should work on */
 6    int length_per_thread=length/t;
 7    int start=id*length_per_thread;
 8
 9    for(i=start; i<start+length_per_thread; i++)
10    {
11      if(array[i]==3)
12      {
13        mutex_lock(m);
14        count++;
15        mutex_unlock(m);
16      }
17    }
18  }
```

Second Try Code provides the correct result, but the performance is slow:



a) Why is the performance serial code time faster than the parallel time with one thread, T=1?

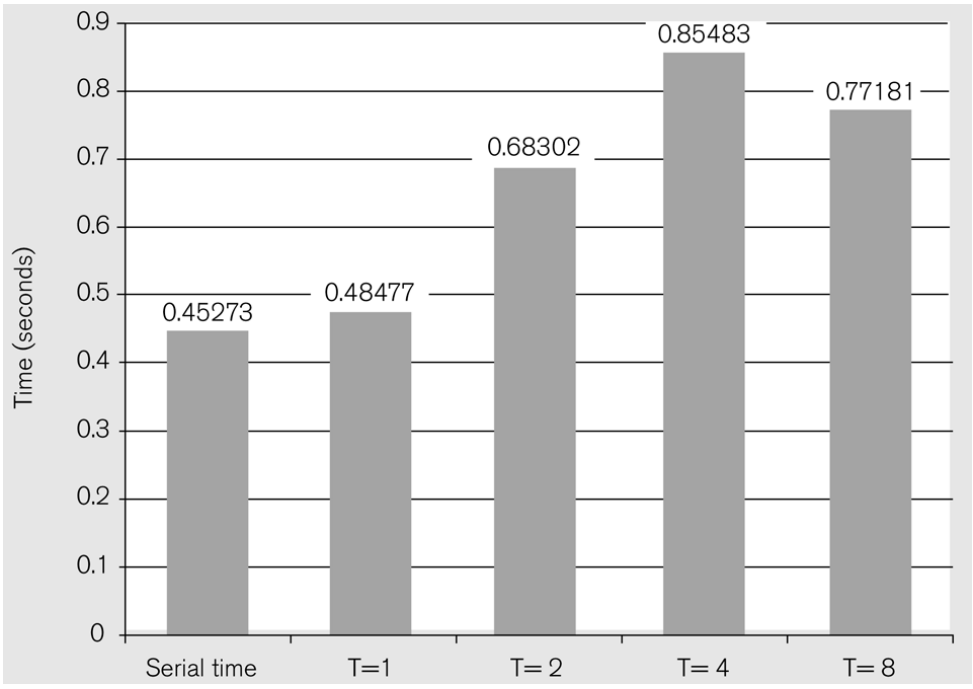
b) Why is the parallel performance time with 2 threads faster than with 4 threads?

c) What can we try to help solve the problem?

Third Try Code:

```
 1  private_count[MaxThreads];
 2  mutex m;
 3
 4  void count3s_thread(int id)
 5  {
 6    /* Compute portion of array for this thread to
        work on */
 7    int length_per_thread=length/t;
 8    int start=id*length_per_thread;
 9
10    for(i=start; i<start+length_per_thread; i++)
11    {
12      if(array[i] == 3)
13      {
14        private_count[id]++;
15      }
16    }
17    mutex_lock(m);
18    count+=private_count[id];
19    mutex_unlock(m);
20  }
```

Performance of the Third Try code:



a) Why is the performance serial code time only slightly faster than the parallel time with one thread, T=1?

b) Why is the parallel performance time with 2 threads faster than with 4 threads?

c) What can we try to help solve the problem?