Part 1

- What CPU are you using?
  Intel(R) Xeon(R) CPU W3505 @ 2.53GHz
- What is the operating frequency of the cores?
  2.53 GHz
- How many levels of cache are there and what are the characteristics of each one?
  3-Level
  - L1: 2 x 32 KB 4-way set associative instruction caches
    2 x 32 KB 8-way set associative data caches
  - L2: 2 x 256 KB 8-way set associative caches
  - L3: 4 MB 16-way set associative shared cache
- What is the microarchitecture of the processor core?
  Nehalem
- How many cores are there?
  2

Part 2:

1. The timers we've been given in the code depends on some kind of constants, like RDTSC depends on the CPU frequency. These constants might differ for different systems and hence the accuracy might be different for different systems, or is system-dependent.
   Resolution for the three timers given are :
   gettimeofday : microseconds
   RDTSC : nanoseconds
   times(): milliseconds (I determined it by looking at the trailing zeroes in the times observed by running the code).

2. RDTSC is basically a cycle counter which can be used as a timer by dividing it by the CPU frequency. In modern processors, the CPU operating frequency might change with different processes which will ultimately affect the timer value as well. Hence, it is not as useful as it was earlier.

3. We need to calibrate each clock according to its resolution. So for each clock, the following parameters need to be changed.
   gettimeofday: GET_TOD_TICS
   RDTSC : CLK_RATE
   times() : GET_SECONDS_TICS

   (modified code attached)
   GET_TOD_TICS and GET_SECONDS_TICS need not be touched but CLK_RATE needs to be calibrated according to CPU frequency.

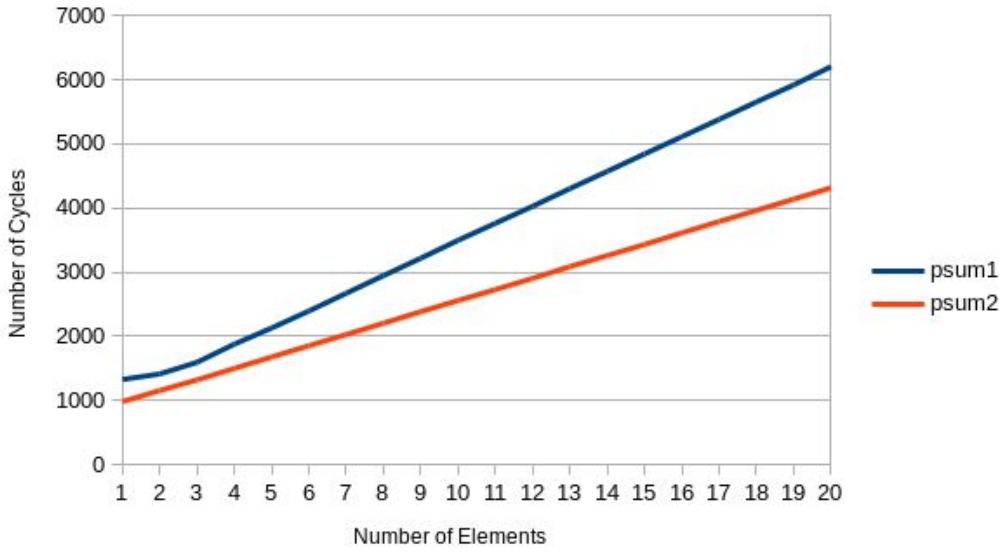
4. a) (modified code attached)
   0.996717130 is my answer.
   0.003282870 is my error.
   2.7345e-6 is my variance.

Part 4:

CPE for psum1 = 13.296
CPE for psum2 = 8.796

The CPE we obtained is different from the one given in the book since the frequency of the CPU we had is different from the one used by the author to make the observations. It also depends on the OS since we don't know how the process schedulers might vary in the systems. Due to all these factors, it's highly unlikely that we'll obtain the same CPE as the one in the book.

Although the graph obtained is similar to the one in the book which establishes the proper working of the code.



Part 5:

1) Starting a loop

   done

   real 0m0.284s
   user 0m0.281s

sys 0m0.001s

2) Starting a loop

done

real 0m0.003s
user 0m0.000s
sys 0m0.002s

4) We observe that O1 optimization reduces the execution time since there is no loop in the assembly code. This is because the variable ‘steps’ that is being used in the loop is never used anywhere in the program and hence the compiler discards it. Whereas in the O0 optimization, the loop statements are still seen in the assembly code due to zero optimization.

5) a) real 1m29.525s
user 0m19.495s
sys 0m5.056s

b) We observe that upon O1 optimization, the loop does show up in the assembly code this time. But there is still some level of optimization observed. Due to optimization, the loop control variable ‘i’ and the variable ‘steps’ are not compared separately, rather the number of iterations are computed in the beginning itself and then compared with the variable ‘i’ in every iteration.
Basically, the optimization changes the way the loop behaves.

Part 6:

1) No.
2) 4-5 hours.
3) No.
4) No.