1. There are three different types of parallelism depending on their granularity: Instruction level parallelism, thread level parallelism, and processor level parallelism. For instruction level parallelism, the hardware/computer is enough to exploit it. For thread level and processor level parallelism, programmer involvement is needed to exploit parallelism.
2. (a) (1) The traditional multiprocessor systems have several chips, but each chip has a single core. This means on each chip the computation is still sequential. However, with current multicore processors, where each chip has multiple cores, parallelism is possible on each chip.
   (2) In the traditional multiprocessor systems, communications are between chips. However, with current multicore processors, communications are between cores on the same chip.
   (3) Multiprocessor systems have several CPUs, while a multicore processor has only one CPU with multiple cores or execution units.
   (b) The parallel algorithms designed for multiprocessor systems can be used for multicore processors, too.
   (c) The communication and synchronization methods between chips used in multiprocessor systems cannot be used for communication and synchronization between cores on a single chip on multicore processors.
3. (1) The majority of the application is the sequential part, so there is not much to parallelize.
   (2) The communication and synchronizing overheads are very expensive, thus prohibits parallelization.
   (3) Load may not be balanced among cores / processors after parallelization.
   (4) It is hard to divide the problem into a set of parallelizable tasks.
   (5) Sometimes programmers simply don't know how to write parallelized programs.
4. Because the communication or synchronizing overheads are more expensive than the gain from parallelizing the parallelizable parts of the program.
5. (1) Speed, i.e., the time each parallel program takes to solve the problem.
   (2) Resource usage, e.g., power consumption, number of cores/processors needed.
6. No. Assume "best" means fastest in this case. If the best sequential algorithm is not parallelizable, then parallelizing another parallelizable sub-optimal algorithm may give us faster execution time.
7. (a) Yes, we can parallelize the program. For example, for each pair (i, i+N/2), where i ranges from 0 to N/2 – 1, we can compute a[i] += a[i+N/2] in parallel. The reason we are able to parallelize is because each computation above is independent of each other.
   (b) N/2. Since there are N/2 computations, each is non-divisible and independent of each other. The best we can do is to assign each computation to a separate core, and finish the computation after one unit of time.