Quiz 1 Solutions

1. A good assembly programmer can potentially obtain a factor of 1.5 to 2 performance improvement over optimized C code. Why don't we program in assembler if performance is important?

   Performance is important. But a view of the entire system must take into consideration all components. Certainly, a program written in assembler could outperform programs written in higher level languages, but the cost of producing this code in terms of programmer's time, and the overall cost of maintaining this code throughout the lifecycle of the solution may thwart any benefit from the improvement in performance. The choice of metric is important to highlight this issue. When one consider performance/$ instead of absolution performance, then the proposition looks unprofitable indeed.

2. A server is showing excessive performance degradation. Outline a methodology for solving the problem.

   It is important to look for bottlenecks. These can be in the code or in the system configuration. Profiling the code is a first step to understand where the code is spending its time. Profiling should be conducted on as many system components as possible, including the CPU, memory, network and back storage. Tight memory can leading to excessive paging activities and hurt performance. Misconfigured back storage can lead to bottleneck in I/O operations. CPU utilization showing near 100% indicate that the code is either flawed or the chosen CPU is not adequate for the job.

3. A big problem in the supercomputing field is the difficulty of writing code that can exploit the parallel architecture. Not only writing multithreaded and message-passing codes is difficult, but also the performance debugging phase can often be very frustrating. Therefore, it has been argued that a better metric is to measure the "time-to-solution" to evaluate a system. The time-to-solution includes the time to write a program to solve the problem, and the running time of the machine to produce an answer. Critique this metric and consider its practical use.

   Time to solution is a useful metric in some situations where the programmer's time is a large component in the time to arrive to a solution. This happens a lot in research environments or in data searches when programmer expertise is need to modify the code and the program runs are relatively small in number. However, production code that runs on a daily basis with no change over relatively long periods is more sensitive to the running time (the machine, not the programmer), and the time to solution converges to simply the runtime of the program, given that the initial programmer’s time will be a small component of the overall time to solution.