Boolean Algebra

At the lowest level modern computers work with signals which are discretized to belong to the set $\{0, 1\}$. As such boolean algebra - the mathematics of values constrained to be either true or false - serves as an effective mathematical model for the behavior of hardware. If you intend to design hardware systems you will need to be proficient at manipulating and simplifying boolean expressions. Here's some practice:

1. Prove that $\overline{x + y} = \overline{x} \cdot \overline{y}$. Use this result to show that $\overline{x \cdot y} = \overline{x} + \overline{y}$.

2. Using algebraic manipulation, show that $a + \overline{a} \cdot b = a + b$. Then, show the same using a truth table.

3. Using algebraic manipulation, show that $x \cdot y + x = x$.

4. Simplify $(\overline{a} + c)(a + b)(a + \overline{b})$. Compare the number of gates needed to implement the simplified version and the number of gates needed to implement the unsimplified version.

5. Write $a \oplus b$ as a sum of products. Do the same for $a \Rightarrow b$ and $a \Leftrightarrow b$. You may find it helpful to draw a truth table before writing the sum of products.

6. Prove using boolean algebra that implication is transitive. That is, show that $a \Rightarrow b \Rightarrow c$ implies $a \Rightarrow c$.

7. $\oplus$ has the interesting property that $a \oplus (a \oplus b) = b$ and $b \oplus (a \oplus b) = a$. Show this, then use the result to write a swap function which uses no temporary variable.

OpenMP

Quick reference: #pragma omp ...

- parallel - run block/statement in parallel based on configured number of threads
- for - make loop index private, run a slice in each thread in parallel
- critical - only let one thread execute code at a time
- private(*var1,var2,var3*) - make variables thread-local
- reduction(*operation: variable*) - make *variable* private, combine values of *variable* in each thread using *operation* after loop
- barrier - threads can't continue until all threads get to barrier
- omp_get_thread_num - function (declared in omp.h) to get number of current thread.

Consider the following C function:

```
/* Assume struct { double ax, ay, vx, vy, x, y; } particles[n]; */
for( int step = 0; step < NSteps; step++ ) {
        //        compute forces
        for( int i = 0; i < n; i++)  {
                particles[i].ax = particles[i].ay = 0;
                for (int j = 0; j < n; j++) {
                        particles[i].ax += xForce(
                                particles[i].x, particles[i].y,
                                particles[j].x, particles[j].y);
                        particles[i].ay += yForce(
                                particles[i].x, particles[i].y,
                                particles[j].x, particles[j].y);
                }
        }

        //        move particles
        for( int i = 0; i < n; i++ ) {
                particles[i].vx += particles[i].ax;
                particles[i].vy += particles[i].ay;
                particles[i].x += particles[i].vx;
                particles[i].y += particles[i].vy;
        }
}
```

How would you parallelize this function using OpenMP?