

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

To show the first part we simply construct truth tables. For the second we consider the relationship of the first part, and substitute in \bar{x}, \bar{y} for x, y :

$$\overline{\bar{x} + \bar{y}} = x \cdot y \Rightarrow \overline{x \cdot y} = \bar{x} + \bar{y}$$

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

$$a + \bar{a} \cdot b = a(b + \bar{b}) + \bar{a}b = ab + a\bar{b} + \bar{a}b = ab + a\bar{b} + \bar{a}b + ab = a(b + \bar{b}) + b(a + \bar{a}) = a + b$$

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

$$xy + x = xy + x(y + \bar{y}) = xy + x\bar{y} = x(y + \bar{y}) = x$$

4. Simplify $(\bar{a} + c)(a + b)(a + \bar{b})$. Compare the number of gates needed to implement the simplified version and the number of gates needed to implement the unsimplified version. One and, versus 2 not's, 2 and's, and 3 or's.

$$(\bar{a} + c)(a + b)(a + \bar{b}) = (\bar{a} + c)(a) = ac$$

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.

$$a \oplus b = (a\bar{b}) + (\bar{a}b), a \Rightarrow b = \bar{a} + b, a \Leftrightarrow b = ab + \bar{a}\bar{b}$$

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

$$\begin{aligned} \overline{(\bar{a} + b)(\bar{b} + c)} + (\bar{a} + c) &= \overline{(\bar{a} + b) + (\bar{b} + c)} + \bar{a} + c = a\bar{b} + b\bar{c} + \bar{a} + c \\ &= a\bar{b} + b\bar{c} + \bar{a}b + \bar{a}\bar{b} + c = \bar{b}(a + \bar{a}) + b\bar{c} + \bar{a}b + bc + \bar{b}c \\ &= \bar{b}(a + \bar{a} + c) + b(c + \bar{c} + \bar{a}) = \bar{b} + b = 1 \end{aligned}$$

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.

(part 1) $a \oplus a \oplus b = 0 \oplus b = b$ (To be properly rigorous we should demonstrate associativity first). The second assertion follows trivially from the first. (part 2) let our variables be X,Y. Then the algorithm is as follows: X:= X \oplus Y; Y:= X \oplus Y; X:= X \oplus Y;

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?

Add `#pragma omp parallel` for before the two second-level loops. We can't parallelize the outer one because each iteration depends on the one before it. Parallelizing the innermost one is difficult as written because there is no synchronization to the updates to `particles[i]`. Finally, the two second-level loops can't be merged because the second depends on the completion of the first.