CSC 410 Prog 1

Documentation

By Zachary Owen

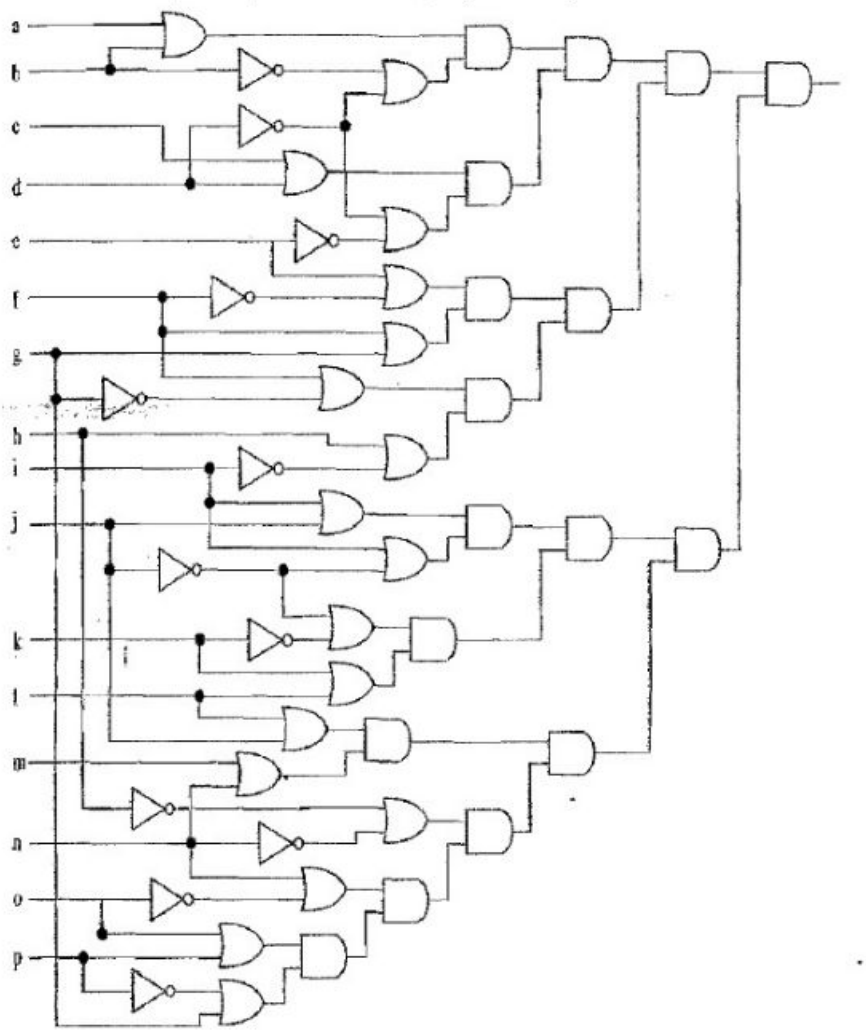
Program Overview:

There are two programs contained within this submission. They are “sat” and “sieve”. Both of which are exercises in parallel computing.

Program descriptions:

Sat.c:

The first program is a satisfiability solver which takes the following diagram and lists the inputs to the program which will give a single output of TRUE.



Due to 3-Sat being an NP-Problem the only way to do this is by checking all 65,536 possibilities in PARALLEL!!!

Program Algorithm:

Very Simple actually, Using the given code for the assignment, the program simply uses the High level control structure for and the omp prama of the same name to try every combination.

Sieve.c

This program takes in a large number as its upper limit for primes and then in increments of 10 prints them to the screen. Sieve uses the Sieve of Eratosthenes

Program Algorithm:

After calling, the program takes the first argument, interprets it as a number and allocates an array of shorts of that size. After memseting the array to 1, the parallel loop begins. At first small sections of the array are given to the sieve function which sets the values at multiples of that index to -1. Particularly before any parallelism takes place the 2's are crossed out. Then those numbers are skipped going forward. Then whenever the program reaches a value in the array that is not set to -1 it is assumed to be a prime and that numbers multiples are removed from the array.
The cool part about this algorithm since shorts are interpretable as words and there cannot be any writing on a read index so, no critical sectioning or reduction is necessary.

Program Testing:

The program was tested first by finding at what size *n* at which the parallel version of the program runs faster than the serial version (around 100,000,000 for good speedup) and then the program was to be tested under 3 environments, Static, Dynamic, and Guided.
This was the plan but after Dynamic and Guided took twice as long as Serial, no more testing was done and the parallelization was done at compile time. This makes sense given the repetitive and predictable nature of the function.