AM225: Assignment 1 (due 5 PM, February 11)

*This homework is designed to get you up to speed with multithreaded programming in OpenMP, and highlight interesting challenges that can occur. Complete at least **four out of six** problems. If you submit answers for more, your grade will be calculated using the four best scores. The questions are roughly ordered from easiest to hardest.*

1. **A randomized Casino game.** Consider a casino game that costs \$250 to play. The casino draws successive numbers from a uniform distribution $U(0,1)$ until the sum exceeds 1. The player then receives \$100 for each number that was drawn.

   (a) Using OpenMP, write a multithreaded code that simulate 10^9 instances of this game, and calculate the expected amount of winnings, reporting your answer to at least five significant figures. Measure the wall clock time of your code using 1, 2, and 4 threads. Based on the expected winnings is this a game worth playing?

   [Note: A difficulty with this problem is that the C++ `rand` function for generating random numbers cannot be accessed by multiple threads simultaneously, since it contains an internal state that becomes corrupted due to race conditions. In the program files, you will find a program `custom_rng.cc` that provides a custom self-contained random number generator. You will need to make multiple instances of this random number generator, each of which is used by a separate thread.]

   (b) **Optional.** Analytically derive the expected winnings.

2. **Cellular automaton mazes.** Consider a cellular automaton on a regular periodic $m \times n$ grid, with cells indexed as (i, j) where $i \in \{0, 1, \ldots, m-1\}$ and $j \in \{0, 1, \ldots n-1\}$. Each cell can either be alive or empty. Define $N_{i,j}$ to be the number alive neighbors of (i, j) in the adjacent orthogonal and diagonal cells. Hence $0 \leq N_{i,j} \leq 8$.

   In one generation, all the cells are simultaneously updated according to the following rules:

   - an alive cell at (i, j) survives if $N_{i,j} \in S_{survive}$,
   - an empty cell at (i, j) becomes alive if $N_{i,j} \in S_{birth}$.

   (a) Implement this cellular automaton, using $S_{survive} = \{1, 2, 3, 4, 5\}$ and $S_{birth} = \{3\}$. Use a grid with $(m, n) = (80, 40)$ that is initially empty except for the square $\frac{m}{2} - 6 \leq i < \frac{m}{2} + 6$ and $\frac{n}{2} - 6 \leq j < \frac{n}{2} + 6$ where each cell is set alive with probability $\frac{3}{4}$. Print a snapshot of the initial condition.[1] Perform 150 generations, and print a snapshot of the system after every 25 generations.

   (b) Extend your program to use OpenMP to process the grid in parallel. Consider simulating on a square $n \times n$ grid, using the same initial condition as in part (a). Consider grid sizes of $n = 16, 32, 64, 128, 256, 512, 1024$ using $T = 1, 2, 3, 4$ threads. For each combination of n and T measure the wall clock time $w(n, T)$ to compute a generation.[2]

   (c) For $T = 2, 3, 4$, make a plot of the parallel efficiency $p(n, T) = w(n, 1)/(Tw(n, T))$ as function of $\log_2 n$.

[1] A simple way to do this is to use ASCII art and the `printf`, `puts`, *etc.* functions presented in the lectures.

[2] To get an accurate measure, you may need to average over the wall clock time to compute a number of generations. For smaller grids, more generations may be required.

(d) **Optional.** On a very large grid, what proportion of cells are alive once the system reaches a steady state?

(e) **Optional.** Investigate the alternative growth model $S_{survive} = \{5,6,7,8\}$ and $S_{birth} = \{3,5,6,7,8\}$ and make a movie showing the typical patterns that emerge.

(f) **Optional.** Is it possible to write the code so that the memory M required scales like $M \sim mn$ byte? Is possible to write the code so that $M \sim mn$ bit?

3. **A Mersenne unprime.** The Mersenne primes are a special sequence of prime numbers of the form $2^n - 1$ for an integer n. In December 2018, a new Mersenne prime of $M = 2^{82589933} - 1$ was discovered, which is currently the largest known prime number. In decimal notation it has 24,862,048 digits.

   The number M is far too large to be represented using a standard computer integer. However, it can be represented in a long expansion as

$$M = \sum_{k=0}^{K} d_k B^k \tag{1}$$

   where B is the base and $d_k \in \{0, 1, \ldots, B-1\}$ are the digits.[3] For this question, it is convenient to set B as a large power of two that fits within a standard computer integer.

   (a) Write a program to find[4] all prime numbers less than 2×10^5.

   (b) Write a program that performs division with remainder on a number represented in the format of Eq. 1. You can assume that the divisor is smaller than B.

   (c) Use your program from part (b) to test how many primes less than 2×10^5 are factors of M. Extend your program to use OpenMP so that several primes can be simultaneously tested using multiple threads.

   (d) The number $N = 2^{82589932} - 1$ is presumably not prime. Using your program from (c), determine the number of primes less than 10^6 that are factors of N.

   (e) **Optional.** Compute the decimal expansion of M. Count the frequencies of the numbers $0, 1, \ldots, 9$ and show that they pass a χ^2 test for randomness.

4. **A multithreaded Sudoku solver.** Sudoku is a popular puzzle that involves filling numbers from 1 to 9 in a 9×9 grid according to simple rules. In magazines and newspapers, you will often find partially completed grids of numbers from which you can fill in the remaining numbers to uniquely complete the puzzle.

   Suppose that the grid squares are numbered from 0 to 80 in some order. Then a simple recursive algorithm to solve a Sudoku is shown in Algorithm 1.

   (a) Implement Algorithm 1 and use it to find the unique solution to the partial grid shown in Fig. 1(a). Find the wall clock time required to solve the puzzle. To gain accurate statistics, you may need to time N repeats, and then take the average.

[3] Hence, if $B = 10$ this is equivalent to the standard decimal notation for a number.

[4] This could be done using the Sieve of Eratosthenes, for example. The Sieve is well-suited to being multithreaded.

(b) Use your program to count the total number of solutions to the partial grid shown in Fig. 1(b). Show that there are exactly 283576 solutions.

(c) Extend your program from the previous section to use OpenMP. Calculate wall clock times using 1, 2, and 4 threads.

(d) Find the total number of solutions to the partial grid shown in Fig. 1(c).

(e) **Optional.** It turns out that many Sudoku puzzles found online and in magazines have redundancies, meaning that some numbers can be removed while still achieving a unique solution. Find an example of real puzzle with a redundancy and use your code to demonstrate this.

(f) **Optional.** Use your program to collect some interesting statistics, such as how the number of solutions scales with the number of initially filled-in squares.

(g) **Optional.** Implement some algorithmic improvements to the solver.

5. **Testing thread performance on the diffusion equation.** Consider the diffusion equation

$$\frac{\partial u}{\partial t} = \nabla^2 u \tag{2}$$

in the two-dimensional plane with coordinate system $\mathbf{x} = (x, y)$. Suppose that instead of solving Eq. 2 directly, the equation is solved in a transformed coordinate system $\mathbf{X} = (X, Y)$ where $\mathbf{x} = T\mathbf{X}$ for some constant 2×2 matrix T. Show that Eq. 2 can be rewritten as

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial X^2} + \beta \frac{\partial^2 u}{\partial X \partial Y} + \gamma \frac{\partial^2 u}{\partial Y^2} \tag{3}$$

and determine α, β, and γ in terms of the components of T. Consider discretizing Eq. 3 using an explicit finite difference scheme with equal spacing h in the X and Y directions, and timestep Δt in time, so $u^n_{j,k} = u(jh, kh, n\Delta t)$. Use an explicit Euler step for $\partial u/\partial t$, the standard centered finite difference formula for $\partial^2 u/\partial X^2$ and $\partial^2 u/\partial Y^2$, and the stencil

$$\left[\frac{\partial^2 u}{\partial X \partial Y}\right]^n_{j,k} = \frac{u^n_{j+1,k+1} - u^n_{j-1,k+1} - u^n_{j+1,k-1} + u^n_{j-1,k-1}}{4h^2} \tag{4}$$

for the cross-derivative term.

(a) Suppose that $\alpha = \gamma$. Perform a Fourier stability analysis[5] by substituting in the ansatz $u^n_{j,k} = [\lambda(l,m)]^n e^{i(jl+km)h}$ into the discretized formula, where $\lambda(l,m)$ is the amplification factor. You may find it useful to express λ in terms of

$$q = \cos\frac{(l+m)h}{2}, \qquad r = \cos\frac{(l-m)h}{2}. \tag{5}$$

Show that there exists a β_* such that if $|\beta| > \beta_*$ the method is unconditionally unstable, but that if $|\beta| \leq \beta_*$ then the method is conditionally stable. In the latter case, determine the maximum allowable timestep.

[5] Notes from AM205 Unit 3 may be useful.

(b) Write a multithreaded code using OpenMP that integrates this scheme on a $N \times N$ grid by dividing the gridpoint updates among P threads. Use the periodic domain $[0,1]^2$, the matrix

$$T = \begin{pmatrix} 1 & 1/2 \\ 0 & 1 \end{pmatrix} \tag{6}$$

and the initial condition

$$u(X,Y,0) = e^{-\cos 2\pi X - \cos 2\pi Y}. \tag{7}$$

Choose a timestep that satisfies the restriction in part (a). Find the wall clock time to run 1000 timesteps for $N = 64, 128, 256, 512, 1024, 2048$ and $P = 1, 2, 3, 4$.

(c) Define

$$\langle u^k \rangle(t) = \int_0^1 dX \int_0^1 dY \, [u(X,Y,t)]^k \tag{8}$$

and

$$S(t) = \sqrt{\langle u^2 \rangle - [\langle u \rangle]^2}. \tag{9}$$

Using $N = 400$ and your choice of threads, plot $S(t)$ over the range $0 \le t \le 1/10$ for the transform T and initial conditions used in part (b). On the same graph, plot $S(t)$ when the transform T is just the identity. Which of the two $S(t)$ curves decreases more rapidly, and why?

(d) **Optional.** Extend your analysis from part (a) to the case when $\alpha \neq \gamma$.

6. **A wooden puzzle.** Chris has a large collection of wooden puzzles, which he mainly received as gifts from his father. Recently, Chris received the *Letter Block Puzzle* in a series from Professor Puzzle, a UK-based company. It consists of fourteen wooden pieces that spell out the letters of "ALBERT EINSTEIN" (Fig. 2). The puzzle contains several different challenges, and the hardest one is to arrange the pieces into $4 \times 4 \times 5$ cuboid. This can be done with a recursive algorithm.

(a) Consider the nine distinct pieces in Fig. 2. For each piece, write down the number of different orientations in which it could be placed into the cuboid. You should find that the pieces can be divided into four sets with different orientation groups.

(b) Let the individual blocks in the $4 \times 4 \times 5$ cuboid be labeled from $i = 0, 1, \ldots, 79$ in some order.[6] Let the nine distinct pieces be labeled $j = 0, 1, \ldots, 8$ and let N_j be the number of available pieces of type j. Using Algorithm 2 write a program to find a valid arrangement of the pieces in the $4 \times 4 \times 5$ cuboid.

(c) Extend your program to use OpenMP, and find the total number of ways that the pieces can be assembled to make the cuboid.

(d) **Optional.** Extend your program to find the following solutions:

i. Using two complete sets of pieces, assemble them into a $2 \times 8 \times 10$ cuboid.

ii. Using three complete sets of pieces, assemble them into a $5 \times 6 \times 8$ cuboid.

iii. Using four complete sets of pieces, assemble them into the shape in Fig. 3.

[6] For example, if the blocks are indexed as (r,s,t) with $r \in \{0,3\}$, $s \in \{0,3\}$, and $t \in \{0,4\}$, then the blocks could be labeled using $i = r + 4s + 16t$.

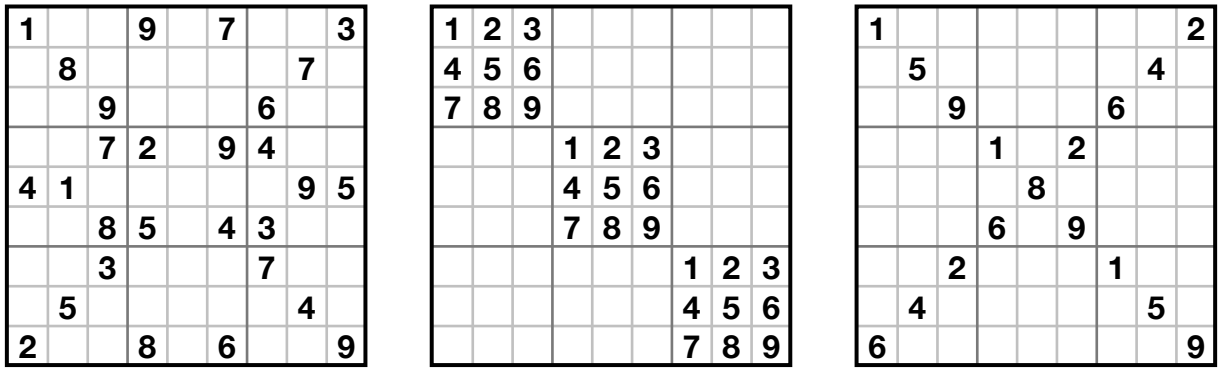

Figure 1: Three partial Sudoku grids.

For these cases, the search tree of possible solutions becomes considerably larger, and you may need to implement code optimizations in order to find.

```
void SudokuSolve (i)
    if i = 81 then
        The grid is complete;
    else
        if square i is empty then
            for j ∈ {1, 2, ..., 9} do
                if j allowed at square i then
                    Add j to square i;
                    Call SudokuSolve(i + 1);
                    Remove j from square i;
                end
            end
        else
            Call SudokuSolve(i + 1)
        end
    end
```

Algorithm 1: A recursive Sudoku solver. Here the squares in the 9×9 Sudoku grid are labeled from $i = 0$ to $i = 80$.

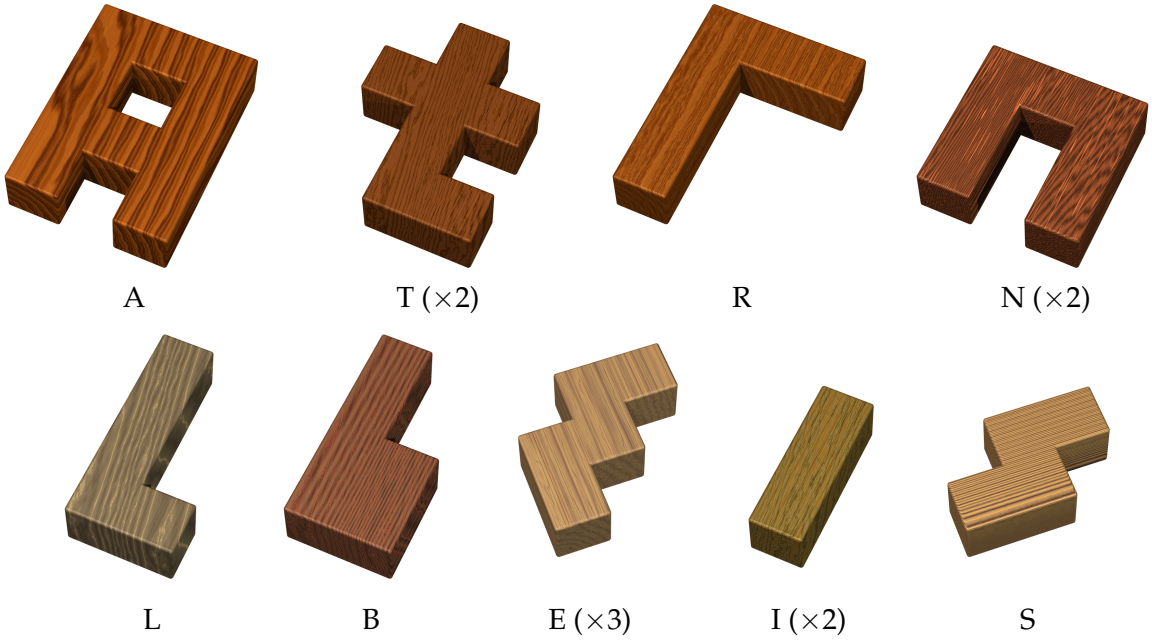

Figure 2: The nine pieces that make up the wooden puzzle in question 6. The brackets show how many times that piece is repeated.

```
void PuzzleSolve (i)
    if i = 80 then
        The grid is complete;
    else
        if block i is empty then
            for j ∈ {0, 1, ..., 8} do
                if N_j > 0 then
                    for all valid positions P of piece j at i do
                        Insert piece j into grid with position P;
                        Decrement N_j;
                        Call PuzzleSolve(i + 1);
                        Increment N_j;
                        Remove piece j from grid with position P;
                    end
                end
            end
        else
            Call PuzzleSolve(i + 1)
        end
    end
```

Algorithm 2: A recursive algorithm to solve the wooden puzzle. Here, the blocks in the $4 \times 4 \times 5$ cuboid are labeled from $i = 0$ to $i = 79$. N_j is the number of available pieces of type j.

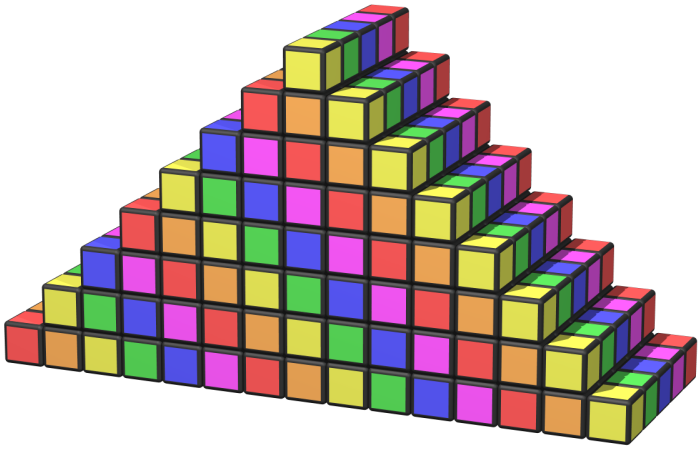

Figure 3: A target shape to make using four complete sets of the *Letter Block Puzzle* with pieces shown in Fig. 2.