1. Recall that in the scatter operation, a process has n pieces of data, each of size W, and wants to ensure the i'th process receives the i'th piece of data, for $0 \le i < n$. Suppose the communication topology of the processes is a $\sqrt{n} \times \sqrt{n}$ grid. Write an MPI program to perform scatter using only point-to-point communication operations. Analyze the time complexity of your algorithm. Assume that sending a message of size m has cost $t_s + m \cdot t_w$, for some constants t_s, t_w.

(25 points)

2. Suppose you are given an array A containing n integer values, and a number $1 \le k \le n$. Let $s_i = \sum_{j=i}^{i+k-1} A[j]$, for $0 \le i \le n-k$. Write an efficient OpenMP program which finds the largest s_i value, and returns both the index i and value s_i. If there is a tie, *i.e.* there are several indices with the same maximum s value, return the smallest such index. Compute the asymptotic time complexity of your algorithm when using p threads.

(25 points)

3. Recall that Peterson's algorithm is a deadlock-free mutual exclusion algorithm for two threads. The algorithm can be generalized to achieve wait-free mutual exclusion for $n > 2$ threads using the code below. Give a (reasonably) complete and convincing argument why this algorithm is correct, *i.e.* why it satisfies both the mutual exclusion and wait-free properties.

```
// shared variables, all initialized to -1
int level[n]
int waiting[n-1]

// thread i tries to enter the critical section
void trying(int i) {
  for k = 0 to n-2 {
       level[i] = k;
       waiting[k] = i;
       while (waiting[k] == i && (exists j != i: level[j] >= k));
  }
  // critical section code
}

// thread i exits the critical section
void exit(int i) {
  level[i] = -1;
}
```

Hint 1: Try running the code for $n = 2$ or 3 to see how the algorithm works.

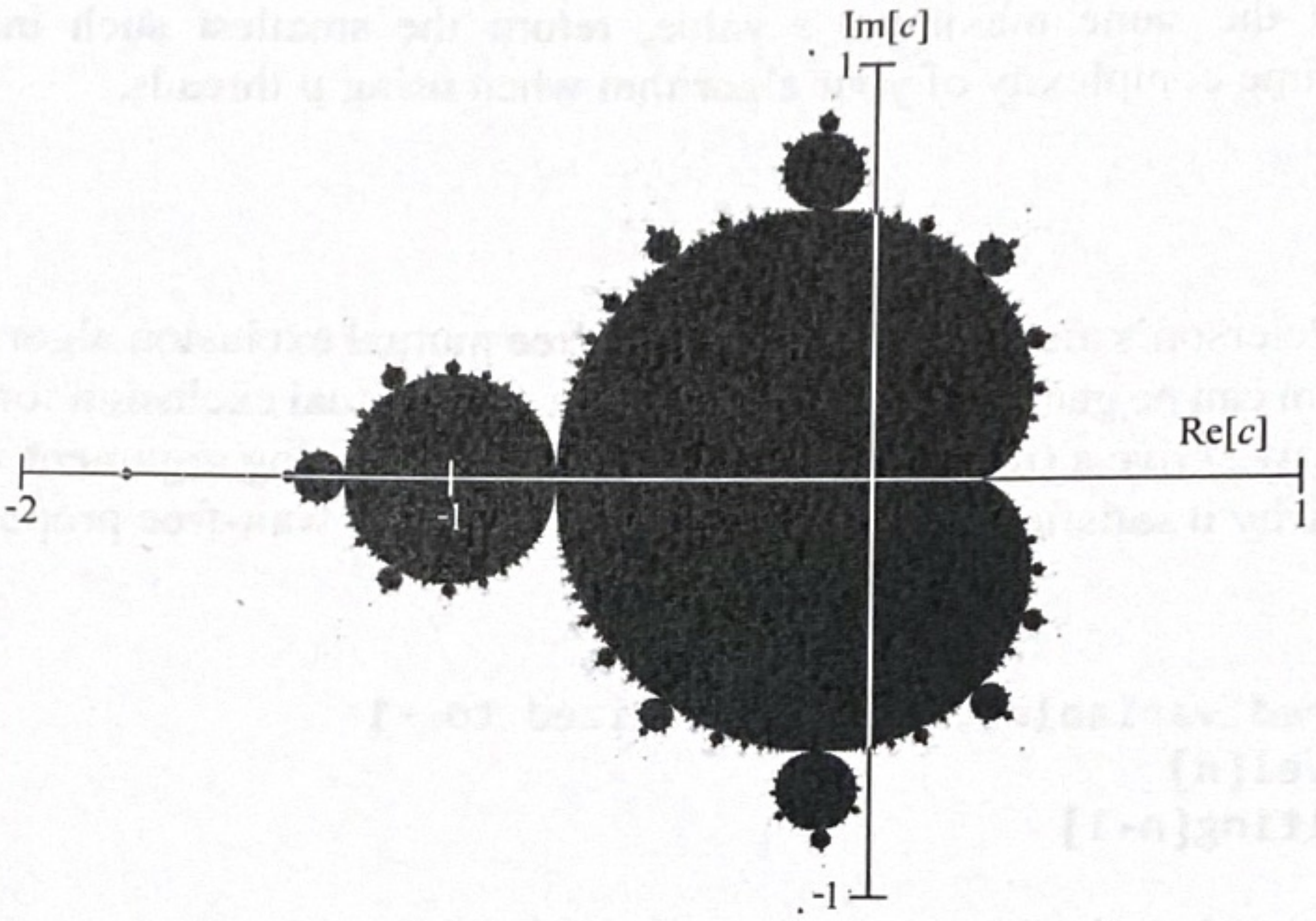
Hint 2: Note that if `level[i]==n-1`, then thread i is in the critical section.

Hint 3: Show that for any level k, $0 \le k \le n-1$, there are at most $n-k$ threads i with `level[i]==k`.

(25 points)

4. Recall that the Mandelbrot set is a fractal subset of the complex plane consisting of complex numbers c which do not diverge under iterations of the function $z \leftarrow z^2 + c$. In practice, for each value c we typically perform a bounded number N of iterations. If $|z| \geq 2$ in any iteration, we conclude z will diverge and stop iterating. Otherwise, if $|z| < 2$ for all N iterations, we declare that z will never diverge.

It is known that the real part of the Mandelbrot set lies in the range $[-2, 0.5]$, and the complex part lies in $[-1.2, 1.2]$. Write a CUDA program to compute the Mandelbrot set by subdividing this region into a grid of size 1024×1024. Use $N = 256$. Your program should output a matrix containing the grid points, where a matrix entry is 1 if the corrresponding complex value belongs to the Mandelbrot set and 0 otherwise. Be sure to write both the GPU kernel function as well as the CPU host code.



(25 points)

END OF EXAM