Vorlesung 3 – Exam Assignments

```
1  #include <stdio.h>
2  #include <omp.h>
3
4  long fib (int n) { return (n < 2 ? 1 : fib (n - 1) + fib (n - 2)); }
5
6  int main () {
7    int n = 45;
8    #pragma omp parallel
9      {
10       int t = omp_get_thread_num ();
11       printf ("%d: %ld\n", t, fib (n + t));
12     }
13   return 0;
14 }
```

Listing 3.2 Computing some Fibonacci numbers.

Die Berechnung der 45. Fibonaccizahl ist bereits sehr rechenintensiv und erfordert daher eine hohe Laufzeit. In Zeile11 ruft das Programm für jeden Thread die Funktion fib auf um die 45 + thread_num Fibonaccizahl zu berechnen. Da es also deutlich länger dauert die 47. Fibonaccizahl zu berechnen als die 46. und deutlich länger dauert die 48. als die 47. zu berechnen usw. wird Thread 1 auf jeden Fall schneller fertig mit der Berechnung als Thread 2 und Thread 2 schneller als Thread 3 usw. Somit kommt es zur Ausgabe in der richtigen Reihenfolge aufgrund der jeweils benötigten Zeit für die Berechnung von fib(n + t).

```
1    while (gens-- > 0) {
2      #pragma omp parallel for collapse(2)
3        for (int i = 0; i < size; i++)
4          for (int j = 0; j < size; j++) {
5            int neighs = neighbors (plane, size, i, j);
6            switch (plane[i][j]) {
7              case 0: aux_plane[i][j] = (neighs == 3);
8                      break;
9              case 1: aux_plane[i][j] = (neighs == 2) || (neighs == 3);
10                     break;
11           }
12         }
13     char **tmp_plane = aux_plane; aux_plane = plane; plane = tmp_plane;
14   }
```

Listing 3.10 Computing generations of Convay's Game of Life.

Parallelisierung beschleunigt die Laufzeit nur, wenn die Aufgaben, die parallelisiert werden, aufwendig genug sind. Sind die Aufgaben zu primitiv ist es schneller wenn sie sequentiell ausgeführt werden. In Convay's Game of Life ist die entscheidende Variable die Spielfeldgröße. Ist das Spielfeld riesig kann es Sinn machen, wie oben dargestellt, zu parallelisieren. Bei einem kleinen Spielfeld ist es wahrscheinlich schneller das Programm sequentiell zu belassen, da die Aufgaben auf aux_plane[i][j] zuzugreifen und sie zu updaten nicht sehr aufwendig ist.

Example 3.5

Runtime:

- Runtime verbessert sich von ~2,4 Sekunden auf ~0,13 Sekunden

Anpassungen:

- Statt normalen Random-Generator zu nutzen, wurde Funktion rnd verwendet, diese ist reentrant und thread-safe
- Allerdings kann rnd-Funktion nur begrenzt unterschiedliche Seeds erzeugen, da seed vom Typ Integer ist und der Datentyp in seiner Größe beschränkt ist
- Für die Addition der Koordinaten wurde reduction clause verwendet, daher sind auch keine lokalen counter Variablen mehr nötig, die am Ende anddiert werden müssen