Lab 4: 生命游戏并行化

221300034 高志轩

关于采用的同步方法：

先后我尝试了许多种同步的方法，主要有以下三种：

1. 每次迭代创建 threads 个线程，每个线程做完后就退出，可以看到，这样写出的程序效果还不错，提交给 oj 还能获得 80 分的成绩。之后一想，这样写天然就避免了线程之间的同步问题，因为每一次迭代的线程都是重新创建的，没有资源竞争的问题。但是，我一开始没有意识到事实上这种方法完全是作弊，实验要求中已经明确说明了不能重复创建、销毁线程。虽然如此，oj 并没有检测出来，大概 oj 的检测存在一些漏洞吧。

2. 于是之后我又重新修改代码，设置了一个全局变量 finished_threads 来记录这轮演化已经算完了多少个部分（每个线程平均分配了(height – 2) / (threads + 1)行进行计算任务），设置了一个锁来保护对 finished_threads 的修改与读取操作，还设置了一个条件变量 cond 来记录是否 finished_threads == threads + 1，即所有线程是否都完成了本轮的计算任务，如果是就将上一轮的地图和计算好的新地图交换。

3. 之后也尝试了使用两把锁分别对 finished_threads 变量和交换地图指针操作进行保护，期望通过降低锁的粒度来提高并行化程度，但是自己本地测试发现影响不大，没有什么必要，所以放弃了这个方法。

关键问题：

1. 首先就是如何保证各个线程之间迭代轮次的同步。

同上述，这一点通过使用锁和条件变量，维护了 finished_threads 变量就能实现。

2. 实验中发现如果每个线程都负责计算连续的几行，计算效率并不高，并不能达到较好的加速比。

事实上，如果每个线程只简单地负责连续的几行，每个线程完成计算的时间相差较大，因为往往是某一部分的细胞在进行演化，大部分区域都不变，于是不少线程会处于忙等待，浪费了计算资源。所以之后修改了每个线程负责的行数，改为每个线程负责相隔 threads + 1 个的位置处的行，例如要计算 5 行，有 3 个线程，则线程 1 负责 1，4 行，线程 2 负责 2，5 行，线程 3 负责第 3 行。

经过测试发现，这样的分配能够有效地提高计算效率，减少线程之间的相互等待。