Lab1 test report

1. 实验概要

多线程编程是高性能编程的技术之一，实验1将针对数独求解问题比较多线程与单线程的性能差异、同一功能不同代码实现的性能差异以及多线程在不同硬件环境下的性能差异。

1.1 程序输入

本项目完成的是Basic Version，程序将在控制台接收用户输入，该输入应为某一目录下的一个数独谜题文件，该文件包含多个数独谜题，每个数独谜题按固定格式存储在该文件中。

1.2 程序输出

实验中把数独的解按与输入相对应的顺序输出到**标准输出stdout**

1.3 Sudoku算法

选择的是老师提供的舞蹈链算法，即`dance-link`算法。

1.4 性能指标

我们在用户按下回车之后获取当前时刻为`start`，待结果全部输出到标准输出之后获取时刻为`end`，则`(end - start)`即为我们程序运行的总时间。

1.5 实验环境

采用的VMware下的Ubuntu虚拟机，Linux内核版本为4.4.0-21-generic，2GB内存，使用2内核

CPU型号为Intel(R) Core(TM) i5-8400U CPU @ 2.80GHz 2.81GHz。

VMware下的Ubuntu虚拟机，Linux内核版本为3.13.0-32-generic，2GB内存，使用1内核Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz 2.50GHz

VMware下的Ubuntu虚拟机， 4.15.0-91-generic 2GB内存，使用2内核 Intel(R) Core(TM) i5-7200U CPU @ 2.50GH

VMware下的Ubuntu虚拟机， 内核版本为4.18.0-15-generic，每个核的线程数： 1，每个座的核数： 4 Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz

1.6 代码实现版本

实验中共使用两份不同的代码：老师提供的和我们完成的。
完成的代码：为适应多线程而在 Code1 上进行了一系列的修改和增添而成。在Code2 中，可通过参数的调节而控制线程数量。Code2 共有 2 种不同类型的线程，一种相当于生产者进程，负责的是任务的读取和分发，还是就是消费者进程，也就是负责任务的分发的进程。

打开终端进入相应的程序文件夹，终端输入make指令编译相应程序，生成**sudoku_solve**可执行文件
终端输入**./sudoku_solve**运行程序，输入相应的输入文件名即可。

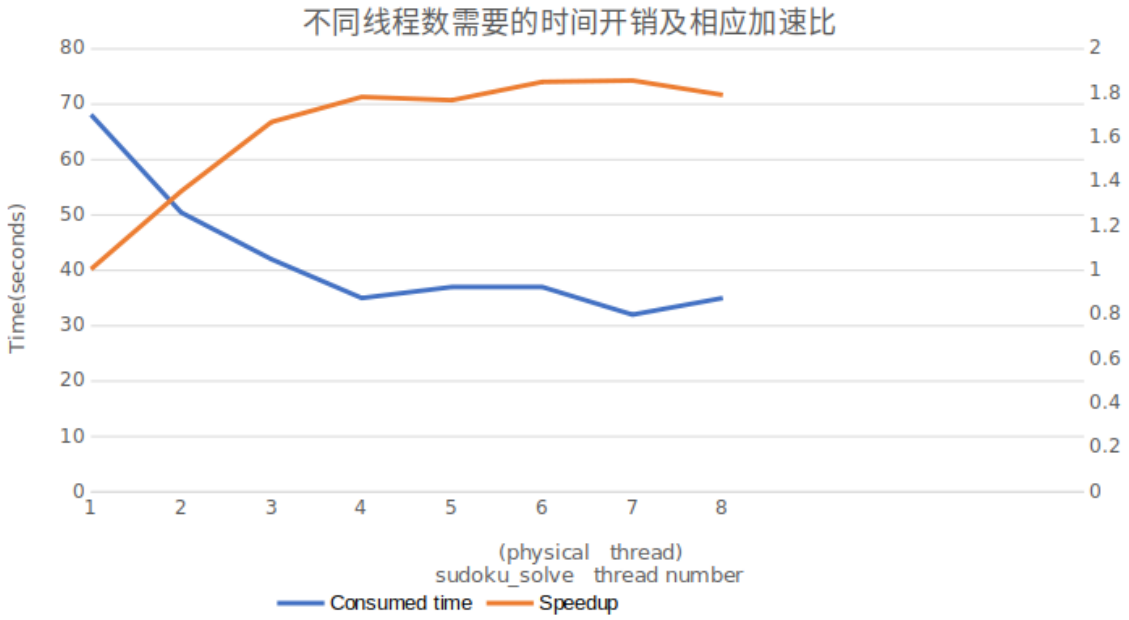
2. 性能测试

程序的性能会受到诸多因素的影响，其中包括软件层面的因素和硬件层面的因素。由于设备配置区别不大，本节将分析比较多线程版本与单线程版本的性能差异，同一功能不同代码实现的性能差异。

2.1 多线程与单线程性能比较

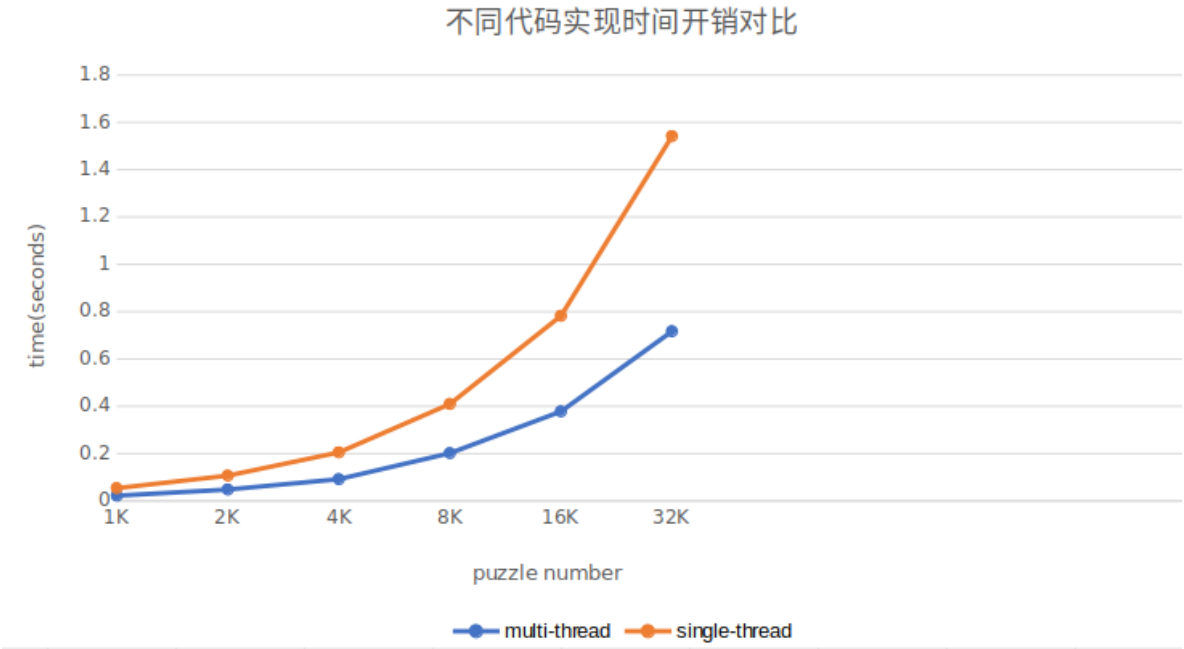
由于单线程版本只能使用1个CPU核心，而多线程程序能使CPU的多个核心并行运作，因此，多线程能够充分发挥多核CPU的优势。在一定范围内，加速比会随着线程数的增加而增长，即时间开销越少、效率越高。当线程数超过CPU核心数时，性能提升将遇到瓶颈，甚至导致下降。为了比较多线程与单线程性能差异，实验将提供若干K个数独题的文件，测量这不同线程输出答案所需要的时间开销并计算加速比。

当总线程数小于CPU总核心数时，随着线程数的增加，所需要的时间开销越小、加速比更高，当线程数为内核数时性能和加速比达到最大。随后，当线程数大于一定数量时，存在两个线程抢用一个CPU内核的情况，在上下文切换时浪费了CPU的运算性能，所以性能提升有限。



2.2 不同代码实现性能比较

接下来，我们比较了多线程版本和单线程版本运行不同大小的测试文件得到的结果，如下图所示。由于多线程版本增加了临界区的控制代码，应该会比单线程版本慢一些，但是图中显示多线程版本的性能是要比单线程好的。


不同代码实现时间开销对比
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
time(seconds)
1K
2K
4K
8K
16K
32K
puzzle number
multi-thread
single-thread