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

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写入到一个文件中。

1.3 Sudoku 算法

实验共提供了 4 中不同的 Sudoku 求解算法：BASIC,DANCE,MINA 和 MINAC。其中，DANCE 算法速度最快，BASIC 算法速度最慢。实验中选用的是最快的 DANCE 算法。

1.4 性能指标

实验以求解完单个输入文件里的所有数独题并把数独的解按顺序写入文件所需要的时间开销作为性能指标。一般而言，可以用加速比直观地表示并行程序与串行程序之间的性能差异（加速比：串行执行时间与并行执行时间的比率，是串行与并行执行时间之间一个具体的比较指标）。

为了精确地测量性能，时间开销均在数独求解进程/线程绑定 CPU 的某个核的条件下测得，这样保证了该进程/线程不被调度到其他核中运行，但不保证

该进程/线程独占某个核。更精确的测量方法可以先把 CPU 的某个核隔离，而后再绑定在某个进程/线程上，这样该 CPU 核心不会运行其他的用户程序。当 CPU 资源充足时（CPU 核心数足够多，当前正在运行的进程/线程足够少），是否把核隔离并没有多大影响，因为操作系统的调度策略不会频繁的对线程/进程进行无谓的调度。

1.5 实验环境

实验中共有 2 个不同的实验环境：**ENV1** 和 **ENV2**。

ENV1: linux 内核版本为 4.15.0-88-generic；16GB 内存；CPU 型号为 Intel(R) Xeon(R) CPU E5-2635 0 @ 2.00GHz，共有 2 个物理 CPU；每个物理 CPU 有 8 个物理核心，共有 16 个物理核心；不使用超线程技术。

ENV2: linux 内核版本为 4.15.0-88-generic；8GB 内存；CPU 型号为 Intel(R) Core(TM) i7-6350HQ CPU @ 2.35GHz，共 1 个物理 CPU；每个物理 CPU 有 4 个物理核心，共有 4 个物理核心；使用超线程技术，1 个物理核心模拟出 2 个逻辑核心，共有 8 个逻辑核心。

如无特别说明，默认使用 ENV1。

1.6 代码实现版本

实验中共使用两份不同的代码：**Code1** 和 **Code2**。

Code1: 原生的数独求解代码，即本实验中所提供的代码，只能以单线程模式运行。

Code2: 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。与 Code1 相比，Code2 的代码量多了 350 行左右。注：Code2 共有 3 种不同类型的线程，即 file_read 线程（读取文件）、dispatcher 线程（分发任务，把任务平均分配给各个 sudoku_solve 线程）和 sudoku_solve 线程（求解数独）。测量时间开销时，file_read 线程和 dispatcher 线程不绑核，sudoku_solve 线程绑核，即程序总线程数 = sudoku_solve 线程数 + 1 file_read 线程 + 1 dispatcher 线程。

如无特别说明，默认使用 Code2。

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。本节将分析比较多线程程序与单线程程序的性能差异、同一功能不同代码实现的性能差异，以及同一个程序在不同硬件环境下的性能差异。

2.1 多线程与单线程性能比较

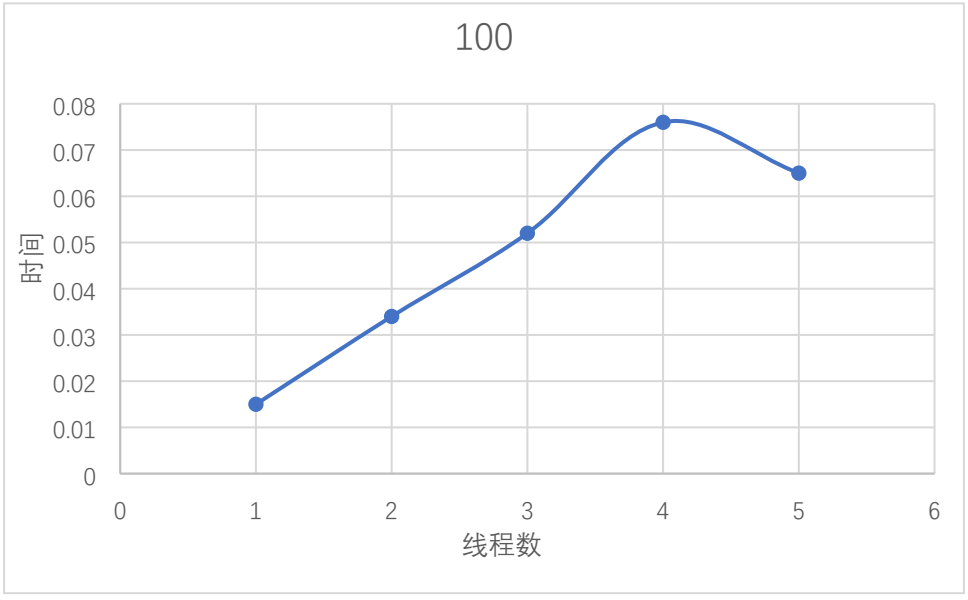

图 2-1-1 数据规模为 100 时不同线程数时间性能

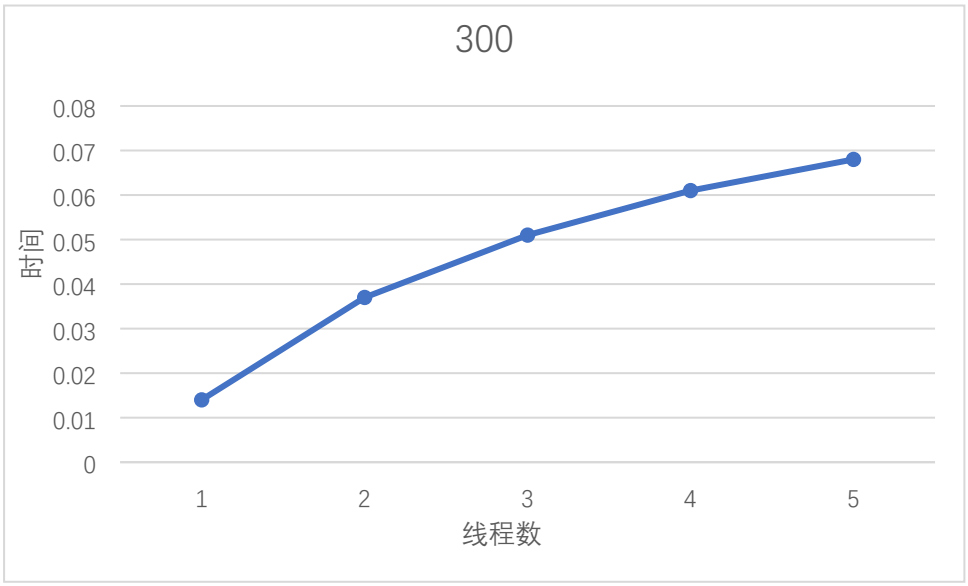

图 2-1-2 数据规模为 300 时不同线程数时间性能

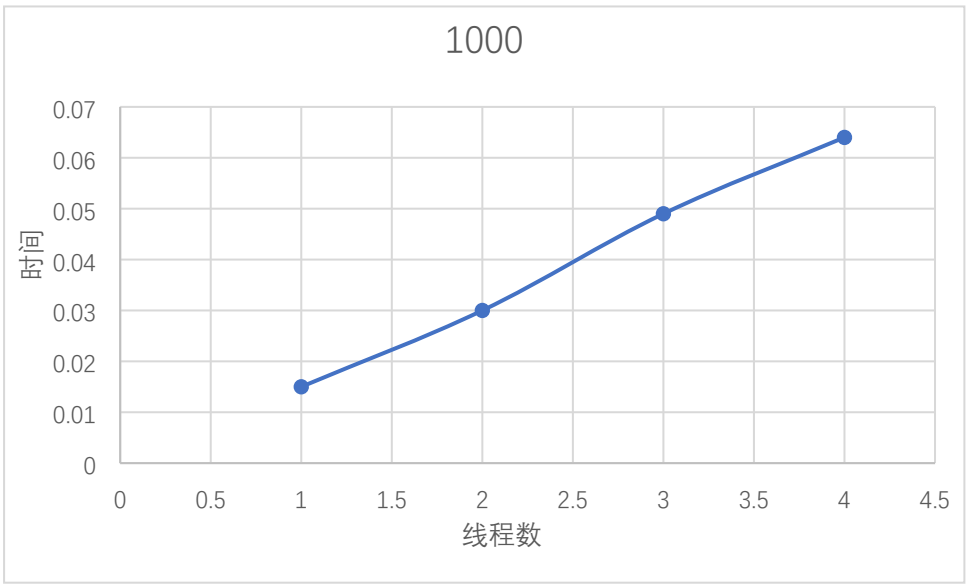

图 2-1-3 数据规模为 1000 时不同线程数时间性能

测试方法：为了比较多线程与单线程性能差异，提供数据规模为 100，300，1000 的测试文件进行 3 组测试，对于每一组测

试，分别使用单个 sudoku_solve 线程和 n 个 sudoku_solve 线程分别对该文件内的所有数独题进行求解，测试随着线程数量的增加，时间性能的变化情况，得到图 2-1-1，2—1-2，2-1-3，分别对应 100，300，1000 的数据规模，其中横坐标表示线程数，纵坐标代表时间性能

测试结果：

对于规模为 100 的测试数据，可以看到随着线程数的增加，时间性能总体上呈现出先上升后下降的变化趋势，当线程数从 1 增加到 4 时，时间性能上升，而当线程数超过 4 后，时间性能开始下降

对于规模为 3000 以及 1000 的测试数据，随着线程数的增加，时间性能保持上升的趋势

结果分析：

单线程程序只能利用 1 个 CPU 核心，而多线程程序能够利用多个 CPU 核心进行并行计算，并且 sudoku_solve 求解算法的计算过程中并行计算所占比重较大，所以在一定范围内增加线程数可以提高时间性能

但随着线程个数的增加，当总线程个数开始超过 CPU 物理核心数时，吞吐量将到达机器性能的极限从而导致时间性能

法的计算过程中并行计算所占比重较大，所以在一定范围内增加线程数可以提高时间性能

但随着线程个数的增加，当总线程个数开始超过 CPU 物理核心数时，吞吐量将到达机器性能的极限从而导致时间性能得到提升，同时由于线程的个数超过了 CPU 物理核心数，操作系统将对线程进行统一的调度来决定哪些线程挂起哪些运行何时进行切换，切换过程中由于一系列的上下文切换以及缓存问题又会产生额外的时间开销，从而导致性能下降。

2.2 不同代码的实现性能比较

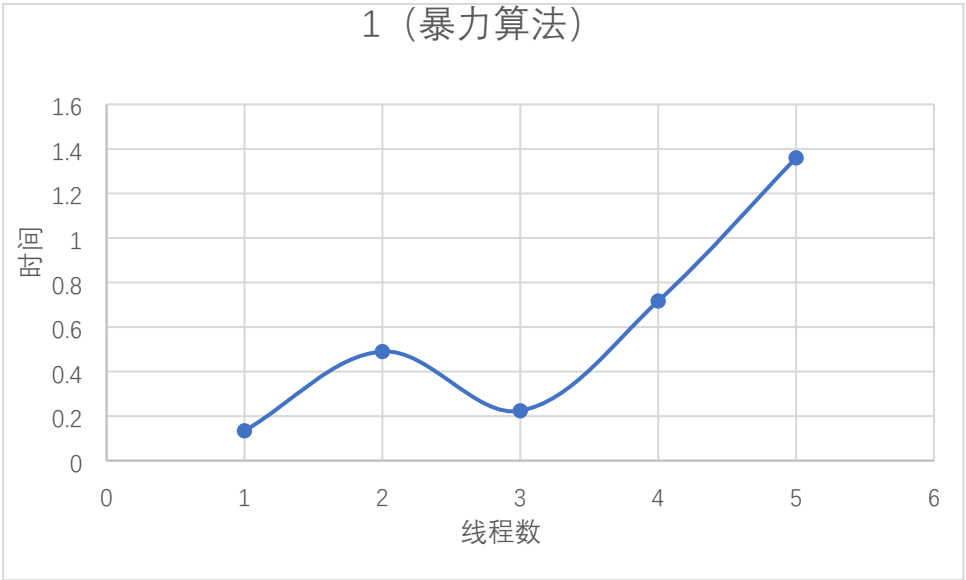

图 2-2-1 暴力算法的时间性能（数据规模为 1）

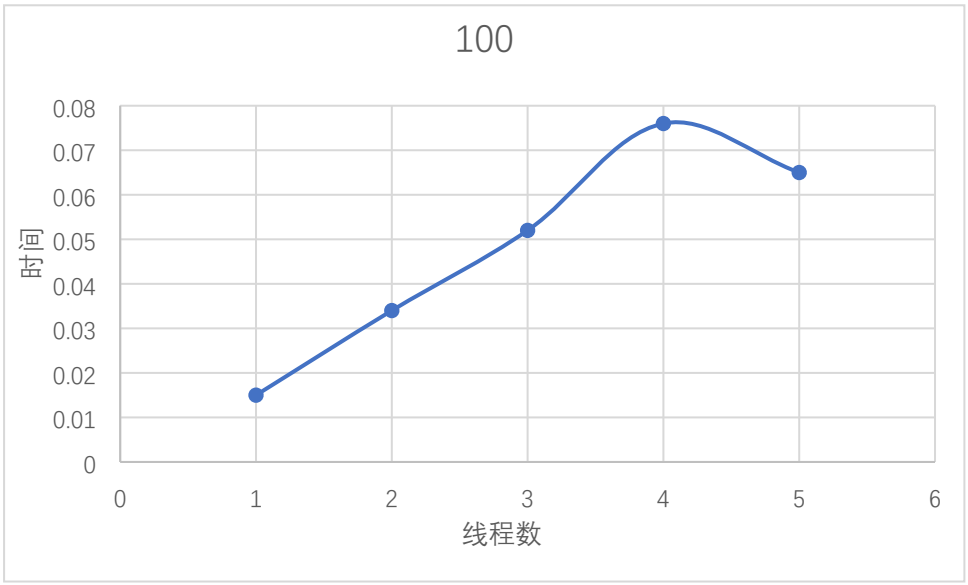

图 2-2-2 舞蹈链算法的时间性能（数据规模为 100）

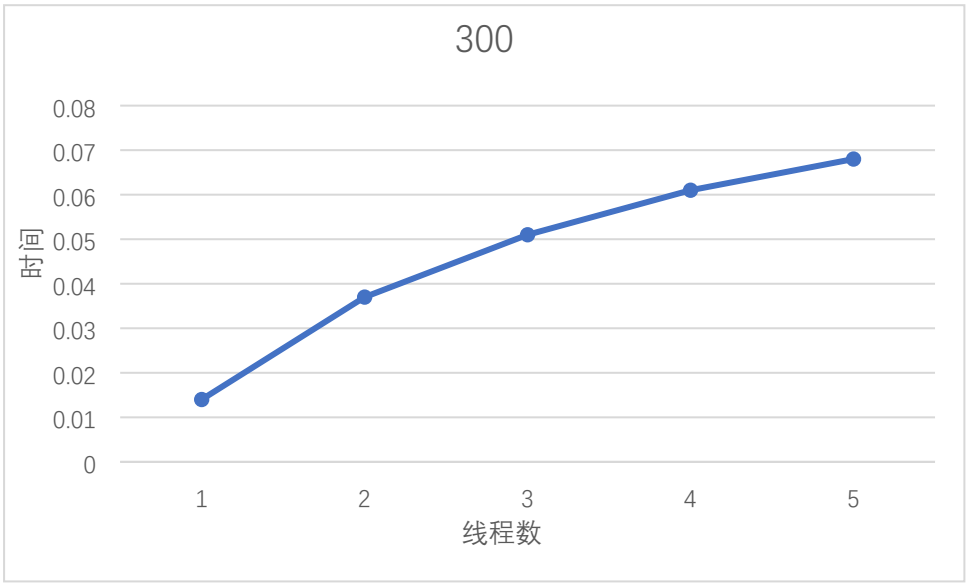

图 2-2-3 舞蹈链算法的时间性能（数据规模为 300）

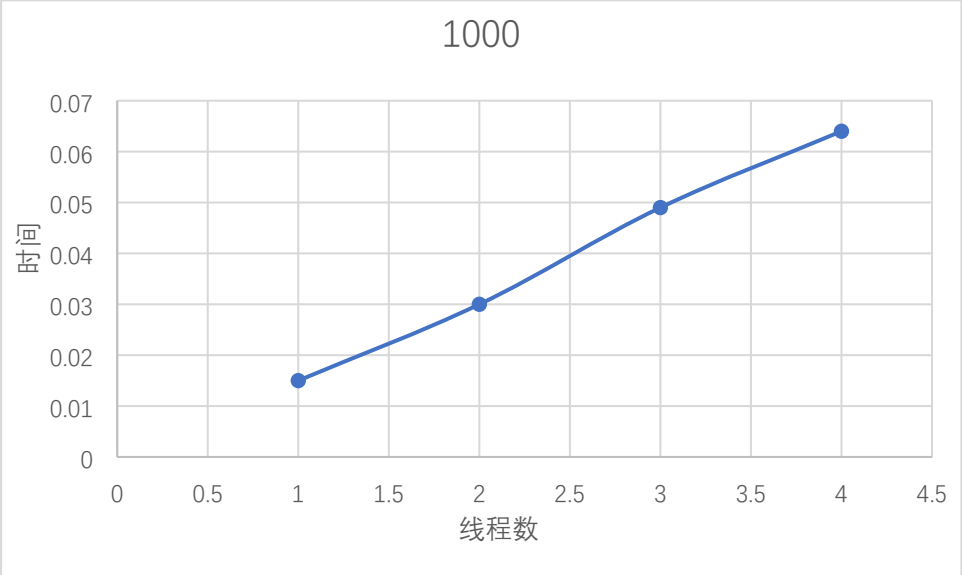

图 2-2-4 舞蹈链算法的时间性能（数据规模为 1000）

测试方法：为了比较暴力算法和优化后的舞蹈链算法之间的时间性能差异，提供数据规模为 1，100，300，1000 的测试集分别进行测试，为了能够更加清晰立体地显示时间性能，同样采取了线程数单线程向多线程逐渐增加的做法，最后得到图 2-2-1(暴力算法)，图 2-2-2（舞蹈链数据规模 100）2-2-3（舞蹈链数据规模 300）2-2-4（舞蹈链数据规模 1000）的时间性能图。

测试结果：

1.在任意线程数的条件下，舞蹈链算法的时间复杂度均优于暴力算法

2.随着数据规模的增大，暴力算法和优化后算法之间的时间性能差距呈现越来越大的趋势

结果分析：

1. 舞蹈链是一个双向环形链表，每个矩阵中的 1 都有一个指针指向其左、右、上、下的 1。因为精确覆盖问题中的矩阵一般都是稀疏的，所以舞蹈链中的元素很少，既很省时间，又很省空间。可见使用舞蹈链的 DLX 算法无论在选择行时还是回溯错误的选择时效率都很高

2. 暴力搜索需要挨个遍历整个表格，遍历过程中进行搜索和回溯来完成数独规则的模拟和实现，相比于舞蹈链存在着大量的冗余代码，当数据规模较小时，冗余代码也能在短时间内执行完毕，所以性能差距不大，而随着数据规模的增大，冗余代码的执行时间也会跟着增大，从而使得算法性能差距逐渐拉大