**“被安排了”组-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。

## 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。

## 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Ter。

## 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。实验中选用的是最慢的 BASIC算法。

## 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。

## 实验环境

实验中共有 2 个不同的实验环境：ENV1 和 ENV2。

**ENV1：**Ubuntu虚拟机，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；不使用超线程技术。

**ENV2：** Mac

如无特别说明，默认使用 ENV1。

## 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，开设10个线程处理1000个数独，理想情况是每个线程处理100个数独，线程动态获取数独去解决。与 Code1 相比，Code2 的代码量多了 100 行左右。注：Code2 共有 1 种线程， sudoku\_solve 线程（求解数独）。测量时间开销时， sudoku\_solve 线程绑核， 即程序总线程数 = sudoku\_solve 线程数

如无特别说明，默认使用 Code2。

# 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。

## 多线程与单线程性能比较

单线程程序只能利用 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，性能会有所下降。

为了比较多线程与单线程性能差异，实验将提供 1 个大小为 81.0 KB、具有1000 个数独题的文件， 而后分别使用单个 sudoku\_solve 线程和 10 个sudoku\_solve 线程分别对该文件内的所有数独题进行求解，并把解输出到控制台，测量这一部分所需要的时间开销并计算加速比。

如图2-1，原来的单个 sudoku\_solve 线程对数独求解需要的时间开销为874.14秒,而Code2的10 个sudoku\_solve 线程将时间开销减少到了367.36秒。加速比约为2.3。

单位：秒

**图 2-1 单个线程与多个线程需要的时间开销**

## 不同代码实现性能比较

对于实现同一功能的程序，可以有多种不同的代码实现，不同的代码实现在时间开销上不一定会相同。实验中使用的 Code2 比 Code1 多了一些额外的代码段，因此 Code2 的时间开销要略大与 Code1，并且随着问题规模的增大，差距也会愈加明显。

考虑到代码可读性、可扩展性或其他因素，有时会在代码实现上增加一些额外的代码段。当这些额外的代码段被调用的次数足够多时，其所造成的时间开销会逐渐显现出来。

实验将使用 2 份不同的代码进行性能比较：Code1 和 Code2。实验提供 7 个不同大小的文件，每个文件分别有数独题：6 K,14 K,27 K,36 K,46 K,51 K,128 K,256 K,512 K 和 1024 K。分别用 Code1 和 Code2 对这些文件进行求解（此处Code1 和 Code2 都是使用单个数独求解线程进行求解），并测量时间开销。

图 2-2 显示数独题量从 1 K 增长到 1024 K 时，Code1 与 Code2 之间的时间开销差距逐渐拉大。在 1024 K 时，Code2 比 Code1 多花费了约 10 S 的时间。因为在代码实现时，Code2 比 Code1 多增加了一些额外的代码，必然会引入一些额外的开销。在这些额外增加的代码中，有些代码段在程序运行期间调用次数不多， 有些代码段则是进行 1 次数独求解就要调用 1 次，当数独求解量达到 1024 K 之

多时，其所造成的开销就会凸显出来。因而会造成随着数独题量的增多，Code1 和 Code2 在时间花销上差距逐渐明显的现象。

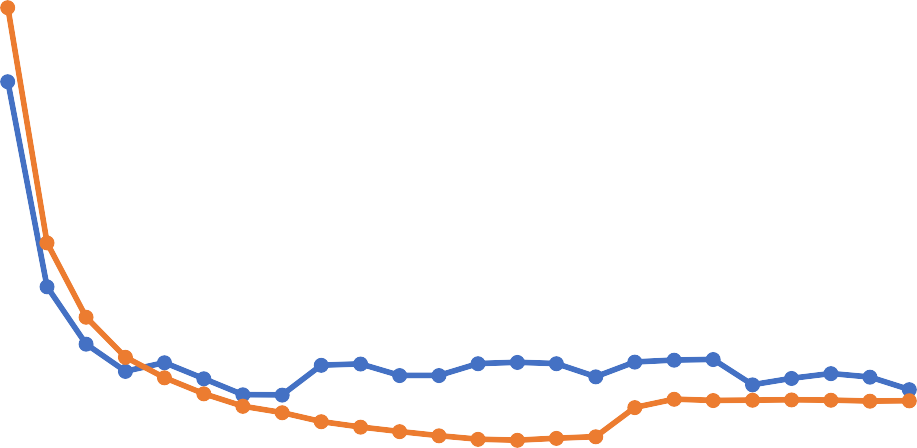
## 不同硬件环境性能比较

硬件环境如 CPU 主频（其它如物理 CPU 个数、物理核心数等）的不同，会导致同一个程序在不同的硬件环境中有不同的表现。对单个 CPU 物理核心，其主频越高，运行速度越快。

实验将使用 Code2 分别在 ENV1 和 ENV2 中对大小为 84.0MB、具有 1024 K 个数独题的文件进行求解，其中 sudoku\_solve 线程数从 1 开始逐步增加，测量时间开销。

图 2-3 为 Code2 在不同的硬件环境 ENV1 和 ENV2 中分别调整不同的sudoku\_solve 线程数的测试结果。从图 2-3 可以看出，当 sudoku\_solve 线程数在 1～4 时，ENV2 的时间花销略小于 ENV1，因为 ENV2 的 CPU 主频为 2.6 GHZ， 略大于 ENV1 的主频 2.0 GHZ，即对单个物理核心而言，ENV2 的运行速度比 ENV1 快。当 sudoku\_solve 线程数量在 5～8 时，ENV2 的时间开销要略大于 ENV1，这是因为 ENV2 总共只有 4 个物理核心，通过超线程技术模拟出 8 个逻辑核心，其单个逻辑核心的性能还是略逊于单个物理核心，但总体上看增加超线程

还是会使性能略有提升。在 ENV2 中，继续增加 sudoku\_solve 线程数量使得线程数超过逻辑核心数（线程数大于 8），其性能会因为线程的调度而出现下降的趋势；同理，在 ENV1 中，继续增加 sudoku\_solve 线程数量使得线程数超过物理核心数（线程数大于 16），其性能也会因线程调度而有所下降。

80

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

70

60

50

Time(seconds)

40

30

20

10

0

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

sudoku\_solve thread number

ENV2 ENV1

**图 2-3 不同硬件环境时间开销对比**