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，DANCE算法最快，然而我们在这里使用了另一种方式——利用生产者消费者模式构造多个线程，从而加快Sudoku算法的执行效率。

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3.13.0-32-generic；1GB内存；CPU型号为Intel(R) Core(TM) i5-3317u CPU @ 1.7GHz，共有1个物理CPU；每个物理CPU有1个物理核心，共有4个物理核心；不使用超线程技术。

ENV2: linux内核版本为3.13.0-32-generic；2GB内存；CPU型号为Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz，共1个物理CPU；每个物理CPU有1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。其间用到了生产者消费者模型对线程进行调度安排（每次读入一组数据，多组数据一并处理）。在Code2中，可通过参数的调节而控制线程数量。code2的代码量在190行左右，比code1的代码略多。从总体上来看，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

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80.0 MB、具有1024 K个数独题的文件，而后分别使用单个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，并把解写入到文件中，测量这一部分所需要的时间开销并计算加速比。

图2-1展示了不同线程数对性能造成的影响，其2条折线：**Consumed time**和**Speedup**分别表示随sodoku_solve线程数量的变化所需的时间开销和相应的加速比。从图2-1可以看出，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。从sudoku_solve线程数为4开始，总线程数（详见1.6）开始超过CPU物理核心数，线程开始被操作系统调度，调度会有一定的开销，所以性能会有所下降。

在实验中，dispatcher线程按任务量平均分配给各个sudoku_solve线程而没考虑按线程的空闲程度分配，造成最慢的sudoku_solve线程会成为瓶颈。实验中file_read线程和dispatcher线程所需要的时间开销是很小的，CPU资源主要用在了sudoku_solve线程上。当sudoku_slove线程数为15和16时，sudoku_solve线程会被调度，但调度次数不是特别多（因file_read线程和dispatcher占用CPU资源少，并且完成后线程会退出），所以最慢的sudoku_solve线程成为瓶颈的问题不是特别明显。

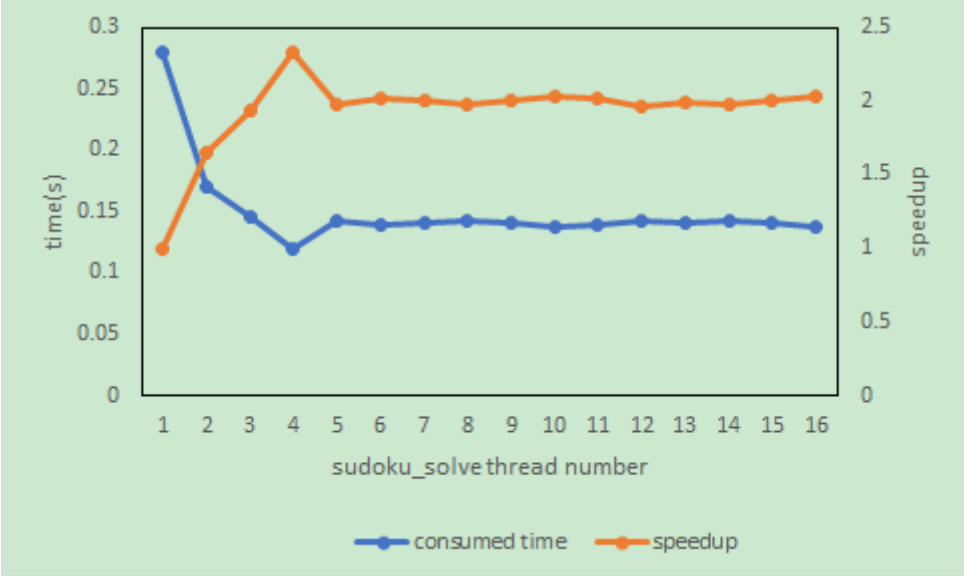

图2-1 不同线程数需要的时间开销及相应加速比

2.2 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8个不同大小的文件，每个文件分别有数独题：1 K,2 K,4 K,8 K,16 K,32 K,64 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8S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。

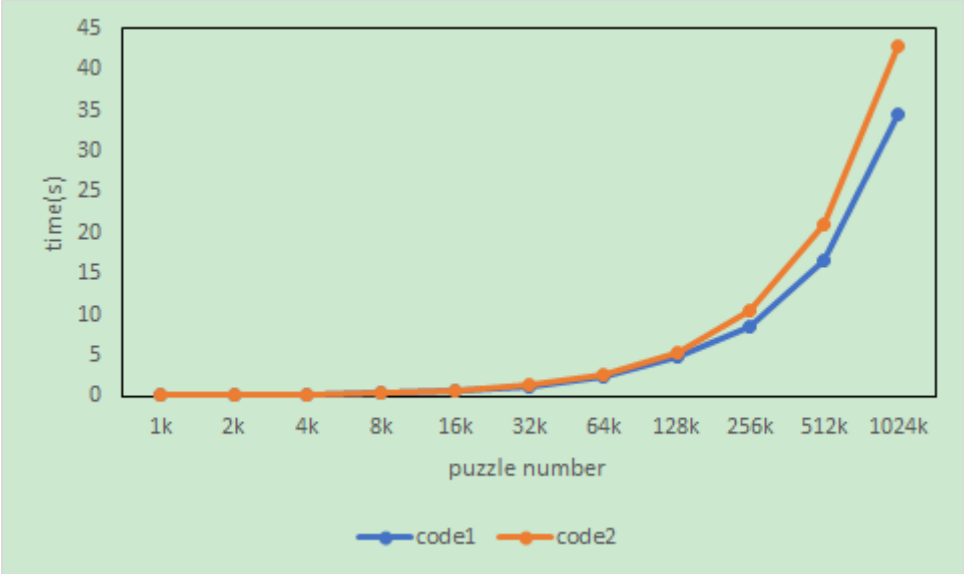

图2-2 不同代码实现时间开销对比

2.3 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80.0MB、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的时候，ENV2的时间花销略小于ENV1，因为ENV2的CPU主频为2.5GHZ，略大于ENV1的主频1.7 GHZ，即对单个物理核心而言，ENV2的运行速度比ENV1快。但在1线程之后，ENV2的时间花销几乎没有变化了，和ENV1几乎相同，后面还有时大于

ENV1，在4线程之后，因为线程数超过了虚拟机的核数，ENV2的时间开销小幅度增大，随后，ENV2变动不大，而ENV1会在较小范围内波动，ENV1和ENV2得到的结果相近。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2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。

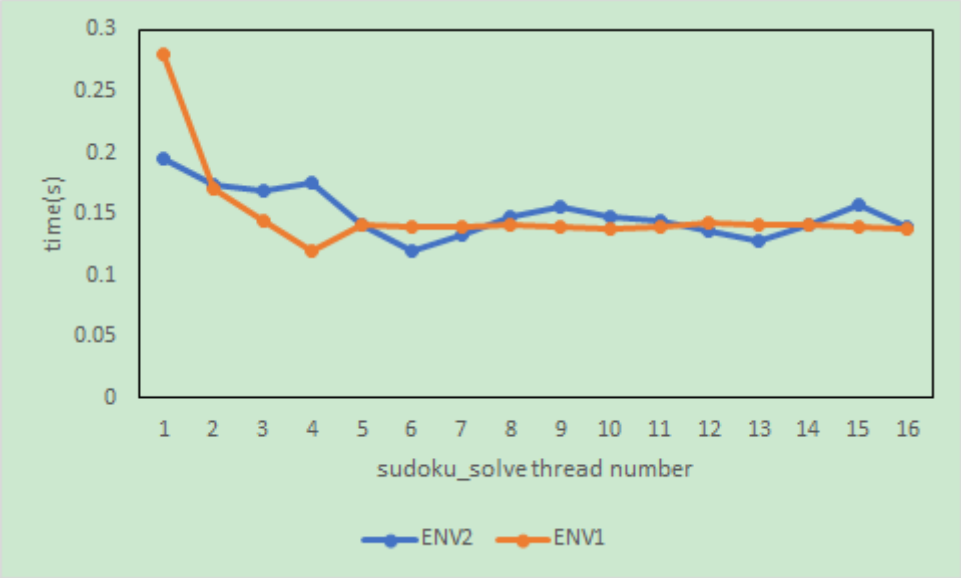

图2-3 不同硬件环境时间开销对比