**龙龙狂吃队Lab1性能测试报告**

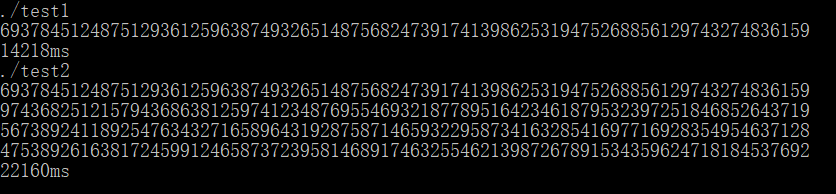
### **实验概要**

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**

（在更改线程数目的时候，修改代码main函数的第一行 ，num0fworkerThread的大小，需要开几个线程，就定义为多大，并且注释掉接下来的两行参数读取即可。）

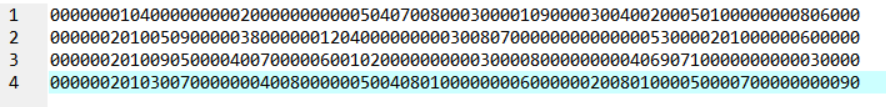
程序将在控制台接收用户输入，该输入应为某一目录下的一个数独谜题文件，该文件包含多个数独谜题，每个数独谜题按固定格式存储在该文件中。输入如下图第一行与第四行。



test1文件内容如下：

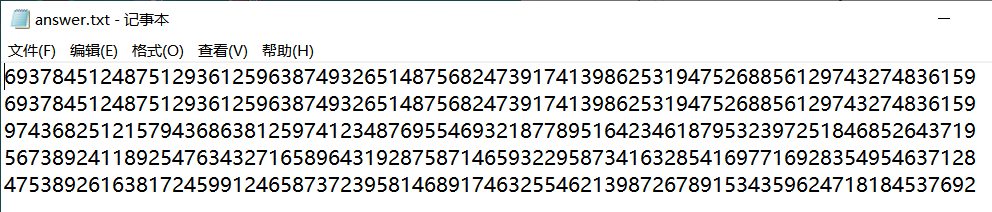


test2文件内容如下：

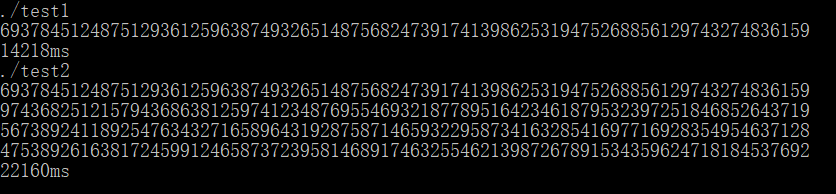


### **1.2 程序输出**

   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。上述输入对应文件输出如下：



在控制台上也会有相应的输出，每输入一个测试文件名，就会显示相应的解和该解是在哪个线程中完成的，最后会显示完成该文件中的所有测试样例需要多长时间。控制台输出实例如下（第一行与第四行为输入）：



### **1.3 DBS深度优先遍历算法**

  本次试验没有选用实验文件中给出的四种算法，而是使用了深度优先遍历，递归调用DBS函数来进行求解的方法，相对较慢，可以更好的显示出多线程的优势。

### **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：****dingmeng436@LAPTOP-8G746GLA****和****jkl@jkl-virtual-machine****。

****dingmeng436@LAPTOP-8G746GLA:****

**Linux version 4.4.0-18362-Microsoft (Microsoft@Microsoft.com) (gcc version 5.4.0 (GCC) ) #476-Microsoft Fri Nov 01 16:53:00 PST 2019；8G内存；CPU型号为Intel(R) Core(TM)i5-7300HQ CPU @ 2.50GHz，共有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。**

****jkl@jkl-virtual-machine:****

**Linux version 4.15.0-91-generic (buildd@lcy01-amd64-017) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.12)) #92~16.04.1-Ubuntu SMP Fri Feb 28 14:57:22 UTC 2020；8G内存；CPU型号为Intel(R) Core(TM)i5-7200U CPU @ 2.50GHz, ~2.7GHz，共有1个物理CPU；每个物理CPU有2个物理核心，共有2个物理核心；不使用超线程技术。**

如无特别说明，默认使用****dingmeng436@LAPTOP-8G746GLA****。

### **1.6 代码实现版本**

  实验中共使用两份不同的代码：**simple.cpp**和**sudoku\_solve.cpp**。

**simple.cpp**:**** 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。

**sudoku\_solve.cpp**:**** 为适应多线程而在**simple.cpp**上进行了一系列的修改和增添而成。在**sudoku\_solve.cpp**中，可通过参数的调节而控制线程数量。与**simple.cpp**相比，**sudoku\_solve.cpp**的代码量多了五十行左右。注：**sudoku\_solve.cpp**的多线程主要运用在DFS求解函数上，可以开多个线程分开求解样例，在求解进度上缩短时间。

  如无特别说明，默认使用**sudoku\_solve.cpp**。

### **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**

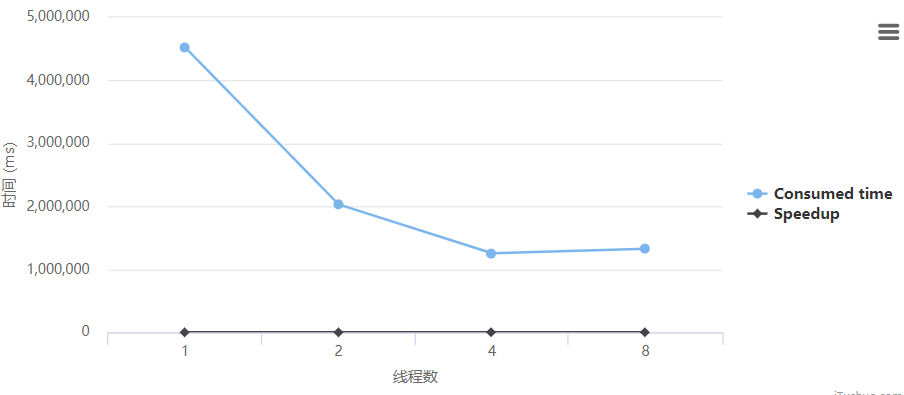
  单线程程序只能利用1个CPU核心，而多线程程序能使CPU的多个核心并行运作，因此，多线程能够充分发挥多核CPU的优势。在一定范围内，加速比会随着线程数的增加而增长，即时间开销越少、效率越高。当线程数超过CPU核心数时，性能会有所下降。

  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 KB、具有1000 K个数独题的文件test1000，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。

（纵坐标数值太大，加速比曲线不明显，实际第一个点到第三个点上升，第三个点到第四个点下降）

（使用环境：****dingmeng436@LAPTOP-8G746GLA****）



****图2-1 不同线程数需要的时间开销及相应加速比****

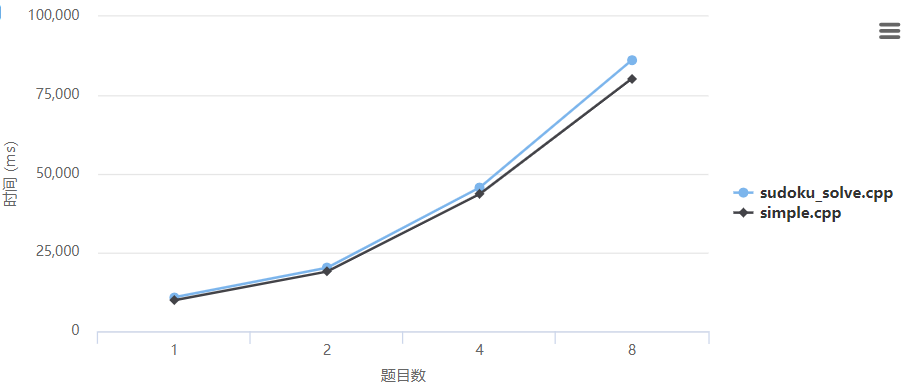
### **2.2 不同代码实现性能比较**

  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sudoku\_solve.cpp比simple.cpp多了一些额外的代码段，因此sudoku\_solve.cpp的时间开销要略大与simple.cpp，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：sudoku\_solve.cpp和simple.cpp。实验提供8个不同大小的文件，每个文件分别有数独题：1 ,2 ,4, 8。分别用sudoku\_solve.cpp比simple.cpp对这些文件进行求解（此处sudoku\_solve.cpp和simple.cpp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显示数独题量增长时，sudoku\_solve.cpp和simple.cpp之间的时间开销差距逐渐拉大。在8时，sudoku\_solve.cpp比simple.cpp多花费了约6 S的时间。因为在代码实现时，sudoku\_solve.cpp比simple.cpp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，当数独求解量越来越多时，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，sudoku\_solve.cpp和simple.cpp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****

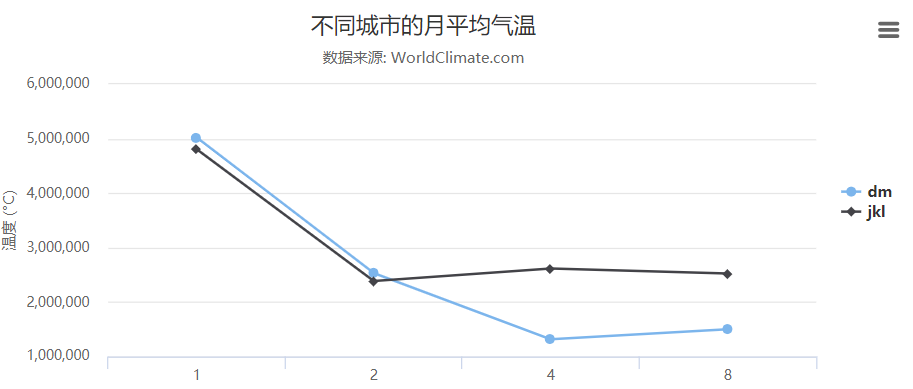
### **2.3 不同硬件环境性能比较**

  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sudoku\_solve.cpp分别在****dingmeng436@LAPTOP-8G746GLA****和****jkl@jkl-virtual-machine****中对大小为80.0KB、具有1000 K个数独题的文件进行求解，其中sudoku\_solve线程数,分别为1、2，测量时间开销。

  图2-3为sudoku\_solve.cpp在不同的硬件环境****dingmeng436@LAPTOP-8G746GLA****和****jkl@jkl-virtual-machine****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1～2时，****jkl@jkl-virtual-machine****的时间花销略小于****dingmeng436@LAPTOP-8G746GLA****，因为****jkl@jkl-virtual-machine****的CPU主频为2.6 GHZ（2.5 GHZ~2.7 GHZ），略大于****dingmeng436@LAPTOP-8G746GLA****的主频2.5 GHZ，即对单个物理核心而言，****jkl@jkl-virtual-machine****的运行速度比****dingmeng436@LAPTOP-8G746GLA****快。当sudoku\_solve线程数量在2～4时，****jkl@jkl-virtual-machine****的时间开销大于****dingmeng436@LAPTOP-8G746GLA**并且趋势不在成倍下降**，这是因为****jkl@jkl-virtual-machine****总共只有2个物理核心，再多开线程也没有物理核心可以使用。在****jkl@jkl-virtual-machine****中，继续增加sudoku\_solve线程数量使得线程数，其性能会因为线程的调度而出现下降的趋势；同理，在****dingmeng436@LAPTOP-8G746GLA****中，因为有4个物理核心，所以当线程数大于4时，时间开销才不再成倍下降，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4），其性能也会因线程调度而有所下降。

（图中dm代表实验环境****dingmeng436@LAPTOP-8G746GLA**，jkl代表实验环境**jkl@jkl-virtual-machine****）



****图2-3 不同硬件环境时间开销对比****