湖南大學

HUNAN UNIVERSITY

云计算技术性能报告

实验名称： 多线程求解数独

组长姓名： 张吴倩-201808010418

组员姓名： 马宇馨-201808010408

组员姓名： 许诺-201808010429

组员姓名： 张辰-201808014013

完成时间： 2021年4月8日

目 录

一、实验概要

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。

Github 网址：https://github.com/HnuXu/CloudComputingLab

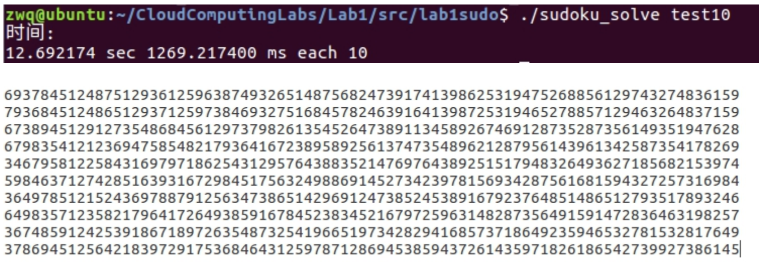
1.1 程序输入

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。 接受一个输入文件名，并且输入文件大小小于 100MB。输入格式如图所示：

```
$ ./sudoku_solve ./test100
```

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 answer.txt 文件中，并且输出到屏幕。



```
zwq@ubuntu:~/CloudComputingLabs/Lab1/src/lab1sudo$ ./sudoku_solve test10
时间:
12.692174 sec 1269.217400 ms each 10
```

```
693784512487512936125963874932651487568247391741398625319475268856129743274836159
793684512486512937125973846932751684578246391641398725319465278857129463264837159
673894512912735486845612973798261354526473891134589267469128735287356149351947628
679835412123694758548217936416723895892561374735489621287956143961342587354178269
346795812258431697971862543129576438835214769764389251517948326493627185682153974
598463712742851639316729845175632498869145273423978156934287561681594327257316984
364978512152436978879125634738651429691247385245389167923764851486512793517893246
649835712358217964172649385916784523834521679725963148287356491591472836463198257
367485912425391867189726354873254196651973428294168573718649235946532781532817649
378694512564218397291753684643125978712869453859437261435971826186542739927386145
```

1.3 Sudoku 算法

实验共提供了 4 种不同的 Sudoku 求解算法：BASIC,DANCE LINK,MINA 和 MINAC。

其中，DANCE LINK 算法速度最快，BASIC 算法速度最慢。实验中选用的是源代码默认的 BASIC 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。

1.5 实验环境

Linu 内核版本为 Linux version 4.15.0-129-generic (buildd@lcy01-amd64-028)； 4GB 内存；CPU 型号为 Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz，共有 2 个 CPU，每个 CPU 均为单核。

1.6 代码实现版本

为适应多线程而在 lab1 给出的源代码上进行了一系列的修改和增添而成。在 main.cc 中，可通过参数的调节而控制线程数量（指定线程数量为 N）。

全局指定 5 个子线程（N=5），主线程负责文件的顺序读入，每读入一个数独记成一个任务（将一个数独存放在全局二维数组里），空闲的线程领取任务（Run 函数中调用 getworkID

函数来给出任务），在完成任务后将解出的数独的解存放在全局的二维数组中，线程任务完成后若有待解决的数独则再去 getworkID 函数中领取完成，在全部数独求解完后将解写入，全局变量 finish 标记任务是否全做完，finish 为 1 后，任务结束，退出 5 个子线程。之后将存放数独的解的二维数组依次输出到文件 answer.txt 中，最后输出运行的时间。

二、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。我们将分析比较多线程程序与单线程程序的性能差异，以及多线程时使用不同线程数和不同 CPU 总核数的性能差异。

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。 实验提供 5 个不同大小的测试文件，每个文件分别有数独题：1、10、100、1000、10000 个。然后分别使用单线程版本和多线程版本（这里定义全局变量为 5 个线程）分别对该文件进行求解测试，比较单线程与多线程的不同耗时。

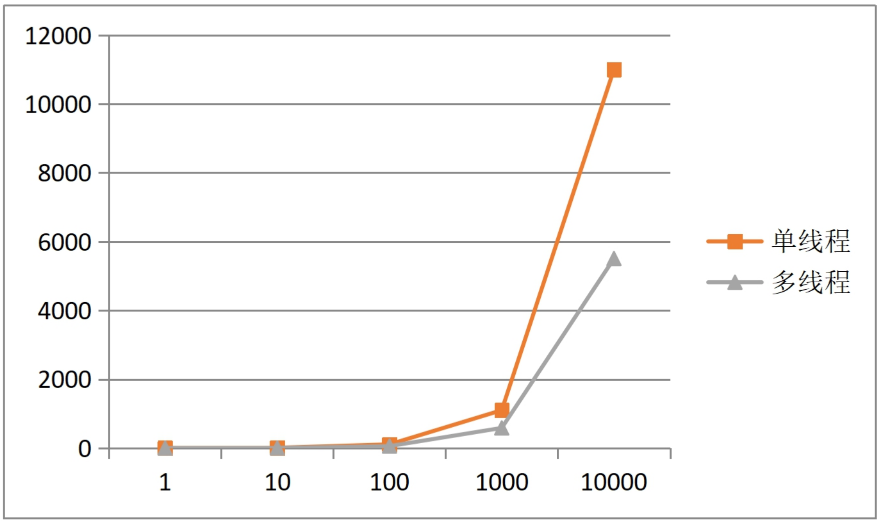
单线程 Basic 算法：

数据规模	1	10	100	1000	10000
第一次测试	2.13	10.14	108.36	1104.13	11000
第二次测试	2.19	10.04	108.08	1100.16	
第三次测试	2.17	9.95	108.04	1097.58	
平均用时（s）	2.16	10.04	108.16	1100.62	11000

多线程算法：

数据规模	1	10	100	1000	10000
第一次测试	1.99	5.32	56.93	583.93	5504.91
第二次测试	2.01	5.82	58.66	581.7	
第三次测试	1.98	5.5	58.25	589.89	
平均用时（s）	1.99	5.54	57.94	585.17	5504.91

（由于规模为 10000 的测试文件运行时间过长，所以只测试了一次，可能存在一定误差）



图表 1 多线程与单线程性能比较

如图，具体化单线程和多线程对性能的影响，画出两条折线：橙色的线为单线程，灰色的线为多线程，分别表示单线程和多线程随着任务量（数独个数）的变化所需要的时间开销。

由数据和图表可知，在刚开始测试的数独数目为 1 的时候，因为任务量太小，单线程和多线程的耗时无太大差别。

随着任务量的增多，单线程和多线程的差距逐渐明显。

一方面，随着任务量的数量级的增加，两种算法所需要的时间同样呈数量级增加。当数独的数量较大时，单线程和多线程的耗时差别明显增大，单线程比多线程增加很多耗时。

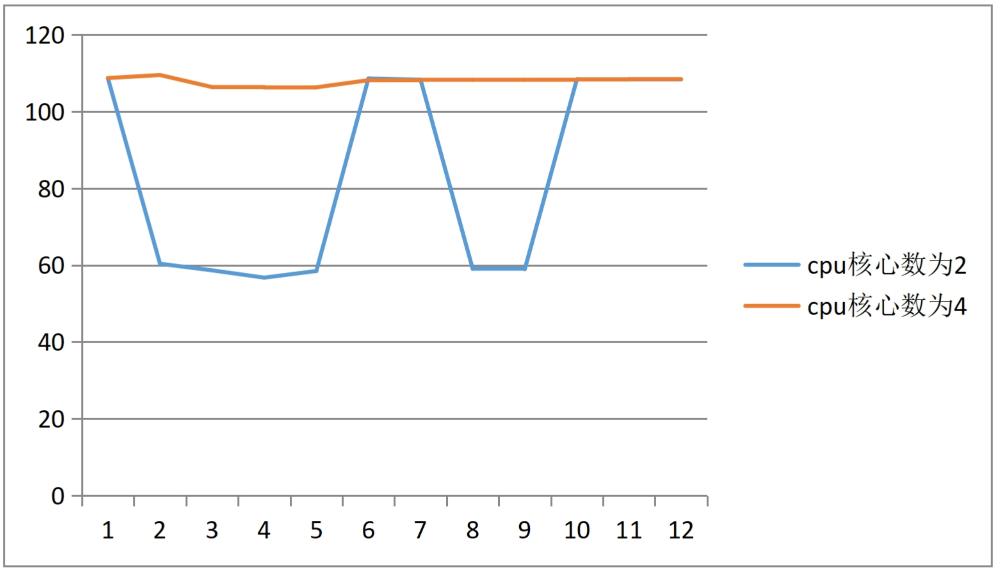
另一方面，在数据规模相同时，多线程的耗时总是约为单线程的一半，表明在 basic 算法上，使用多线程能有效提高算法的性能。以运行时间为指标，当进行运算的子线程数为 5 时，程序的性能比之前提高一倍。

2.2 多线程时使用不同线程数和 CPU 总核数的性能比较

线程数量与 CPU 核心数量的不同，会导致同一个程序在不同的线程数量，不同 CPU 核心数量时有不同的表现。

实验测试不同的线程数量，不同 CPU 核心数量求解规模为 100 的数独对程序性能的影响，其中线程数从 1 开始逐步增加，测量时间开销。CPU 核心数量选取了 2 和 4 两个进行比较。

CPU 总核数	**子线程数**	**1**	**2**	**3**	**4**	**5**	**6**
2	时间	108.66	60.36	58.64	56.74	58.47	108.56
4		108.68	109.47	106.33	106.24	106.27	108.12
CPU 总核数	**子线程数**	**7**	**8**	**9**	**10**	**11**	**12**
2	时间	108.26	59.07	58.94	108.36	108.34	108.41
4		108.23	108.23	108.24	108.31	108.38	108.28



图表 2 多线程时使用不同线程数和 CPU 总核数的性能比较

上图为程序使用不同线程和 cpu 核心数，多次运行数据量大小为 100 的数独文件，取其运行时间的平均值所作出的折线图。由于测试次数不够多，收集的数据存在一定的误差。

当 cpu 核心数量为 4 时，运行的时间与线程数量几乎没有任何关系。一个可能的原因是，由于代码中没有对线程的调度进行控制，在核心数量增加的情况下，操作系统进行随机调度产生的开销使性能提升效果不明显。同时，进行测试时的任务量不大，使性能上的优化无法体现出来。另外，也可能涉及虚拟机软件运行和虚拟机与物理机交互的问题。

当 cpu 核心数量为 2 时，其中当线程数为 1 的时候，等价于单线程程序进行运算，消耗的时间较多。当线程数变为 2 时，性能有较大的提升。线程数为 2，3，4，5 时消耗的时间较少。在此时，用于解题的子线程数与 cpu 核心数较为接近，能充分利用 cpu 的计算能力。当线程数增加时，因为操作系统频繁出现调度，产生的开销影响大于代码优化的影响，使优化效果不明显。同时，因为任务量（100）不大，线程数为 2，3，4，5 时的实际性能差距不大，可以猜想，当任务量加大时，不同数量线程的性能差别将会更加明显。但是由于大规模数据测试时间过长，所以在这一阶段中没有使用大规模数据进行测试，只是使用了规模为 100 的数据来测试。

另外，在实际测试中，当 cpu 核心数为 2、线程数为 7，8，9 时，测试的结果为 108s 与 58s 的概率大致相同。据推测，在数据量不大的情况下，这样的结果可能与每次执行时操作系统具体的调度相关。但总的来看，在这样的条件下，性能的提升不是很稳定。程序使用的线程没有对线程的调度进行控制，随机调度产生的开销造成程序的不稳定性，同时无法对线程进行最大化的优化。

另外：在实验过程中，除了 basic 算法外，我们还对 min_arity 算法进行了多线程的修改。但是，在测试的过程中，min_arity 算法的多线程运行始终不尽如人意，运行的时长比使用单

线程时还要慢很多。经推测原因可能如下：进程只有一个，所以分配的 CPU 资源是一定的，多线程只是轮流抢占 CPU，并不会真正提高处理速度。多线程的作用主要在于提高了并发数量，但是多线程由于轮换使用 CPU，线程之间的轮换以及上下文切换会花费很多时间，造成单个线程的执行速度变慢。