Lab3: 性能分析

匡亚明学院　181840326　张祎扬

Lab3: 性能分析
- 实现命令行工具
- 获取精确时间
- 调查multimod实现的性能差异
- 思考题
- 心得体会

实现命令行工具

实现命令行工具需要对参数进行解析。经过一些printf的尝试，main函数的第一个参数argc是参数的个数，argv的第一个参数（下标为0）是执行可执行文件时输入的命令，比如`./perf-64`,所以真正对函数起作用的参数从`argv[1]`开始。实现参数解析的过程不是很复杂，判断一下参数的个数然后分情况处理即可（在这里没有过多考虑输入不符合要求的情况），最后基本实现了可以自主选择函数和执行次数的功能。

用用户友好的形式展示时间

显然我还没有很强的画图能力……所以我选择将一些信息输出到终端。函数运行完之后，我获得的是一个数组，里面存放的是每一次运行的时间。在这个基础上，首先我可以计算多次运行的总时间，并且通过计算输出每一次运行的平均时间。但我觉得这还不够，还可以找到最大最小值，并且将最大最小值范围之间的数据分段计数，以数据的方式展现一个“直方图”的形式。

获取精确时间

通过stfw，头文件<time.h>中声明的clock()函数可以获取从“开启这个程序进程”到“程序中调用`clock()`函数"之间的CPU时钟计时单元。所以把函数`gettime()`的返回值设为`clock()`就可以返回精确的时间。

调查multimod实现的性能差异

首先我在头文件中添加了对三个函数的支持，并且将这三个函数复制过来，再修改它的形式，使返回值是void，参数也是void，然后在函数内部通过读取文件来获取参数。

- a,b,m的大小。

  首先我假设a,b,m的范围相同。在函数p1,p2,p3中，a,b,m的范围都是0～2^63-1,所以我每次将范围递增2的10次方，在这里只展示一个运行的平均时间（即每执行一次的时间）。结果如下

a,b,m的范围	P1	P2	P3
0~2^10	2.533400	2.682800	2.502500
2^10~2^20	2.581700	2.543000	2.526200
2^20~2^30	2.542300	2.611700	2.616900
2^30~2^40	2.637300	2.699700	2.531000
2^40~2^50	1.827900	1.841200	1.879400
2^50~2^60	2.747600	2.770300	2.738800
2^60~2^63-1	2.696300	2.663300	2.662200

从以上数据可以看出，从大体上来看，随着a,b,m的增大，运行的时间从总体上是不断增大的。所以，当a,b,m的值较小时函数的性能总体较好。但是三个函数都在2^40~2^50处出现了反常，运行时间是最低的。因为我在测试的时候是在每一个范围中随机取了a，b，m三个数，所以不排除出现巧合的可能。

接着，我将a，b控制在2^30~2^40之间，不断改变m的大小，来比较a,b,m之间的相对大小和性能的关系。

m的范围	P1	P2	P3
0~2^10	2.526400	2.546800	2.592500
2^10~2^20	2.632200	2.485100	2.582300
2^20~2^30	2.607200	2.528900	2.546900
2^30~2^40	2.586200	2.762100	2.571000
2^40~2^50	2.626000	2.543500	2.606900
2^50~2^60	2.639800	2.588200	2.585200
2^60~2^63-1	2.465900	2.674000	2.650400

这里情况就比较复杂了，对P1来说，当m很小和很大时，性能都表现很好，当m的范围和ab的数量级相差不大的时候，性能次之。在其他情况下，无论是m大于a，b还是m小于a，b，性能均表现不好。而对于p2来说，当m与a，b数量级相差不大的时候，性能是最差的。对于p3来说，m很大时性能最差，m小于a，b时性能是比较好的。

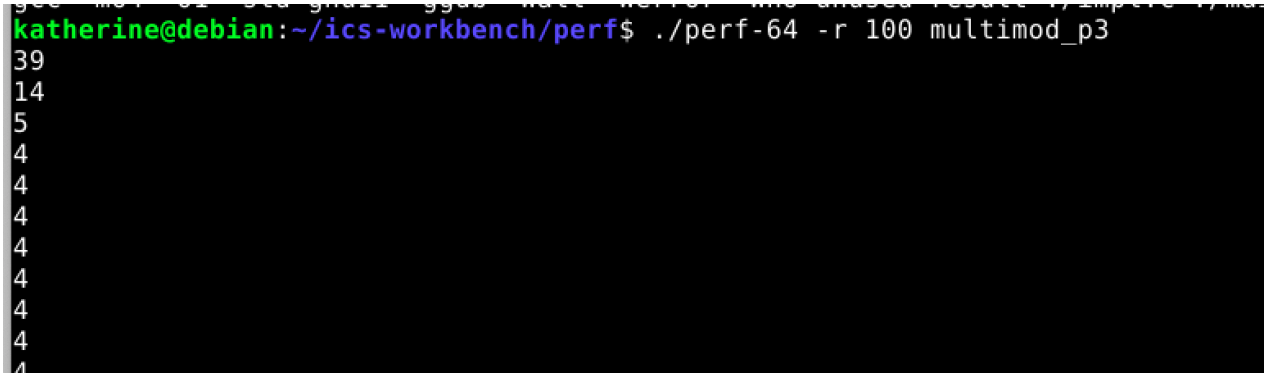
- a,b中二进制的数量

  因为a,b在这里是等价的，所以我保持b和m不变，改变a中二进制1的数量。在这里我将b设为0x1010101010101010, m设为106271614（随机）。

a中二进制1数量	P1	P2	P3
1	2.667700	2.593400	2.613700
10	2.695300	2.664600	2.658300
20	2.685100	2.727900	2.665500
30	2.681500	2.701000	2.841400
40	2.582800	2.803600	2.717800
50	2.751600	2.890600	2.653500
60	2.703800	3.036600	2.689700

从中可以看出，随着a中二进制1的增加，p1和p3的性能变化并没有非常明显的规律，而p2表现出了很明显的规律，那就是随着a中二进制1的增加，函数的性能下降，执行时间变长。这和函数的实现是相关的，p1和p3都直接进行乘法运算，而p2是通过位运算实现的。当a中二进制1增多，进入循环的次数就增多，执行的指令也增多（如果某一位是0的话相当于什么也不做），所以函数的性能会逐渐下降。

在这其中，有很多我难以解释的现象，比如时间突然变长或者突然变短。我觉得这也和两次执行perf的间隔有关。间隔时间较短的时候，缓存还处于一个比较活跃的状态，所以性能会好一些，假如间隔久了时间就会慢一些。这也可以从单次运行看出。我在每一次函数执行完都打印出时间的值，可以很明显的看出前两次的时间和后面的相比非常长，这应该是缓存不活跃的结果。为了印证，我又执行了simple_loop,它的单次执行时间比mulimod长的多，可以明显的观察到单次执行时间随着执行次数的增加有所减少。



思考题

1. 如果使用void *ptr，没有办法得知指针指向的函数类型，可能会在函数调用的时候出问题。
2. 当复制的字节数变大时，各种方法对优化的敏感性都下降。

   CPU主频的提高对性能提高有正面作用。

   asm指令表现最优时复制的字节未必最小，但是其他基本遵循吞吐率随复制字节增加而下降的规律。体现出了asm指令在数据对齐时复制的优势。

   总的来说，优化级别越高性能越好（参考simple）（显然不绝对）。

心得体会

影响函数性能的因素是多方面的，可能是函数自身的实现，也可能是参数的设置，还可能是编译器的优化级别甚至是硬件的差距。还有一些不确定的因素，比如缓存是否在活跃状态等。所以，追求性能的优化是一个很值得探索的领域，并且它可以有很多不同的优化方向。一个优秀的程序员应该通过写出优秀的代码来在软件上达到性能的优化。

但同时我也觉得，性能优化是一个循序渐进的过程，是不断修修补补的过程。如果一上来程序还没有基本框架或者一个基本能跑起来的正确做法都没有，就开始事无巨细地考虑优化细节的话，效果可能会适得其反。