Lab3 实验报告

人工智能学院 薛正海 181220063

1. 命令行工具

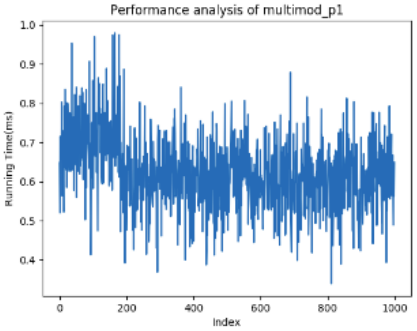
由于需求较简单,我直接手动解析了argc和argv参数.收集到运行时间信息后,我将其写入临时文件中供python使用(由于工具没有性能要求,我没有利用更复杂的通信方式).python运行前需在perf目录内运行pip3 install -r requirements.txt --user.运行时,其利用pandas工具对运行时间作统计,并利用matplotlib对数据可视化.

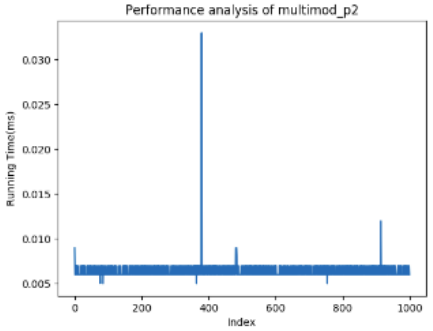
2. 调查 multimod 实现的性能差异

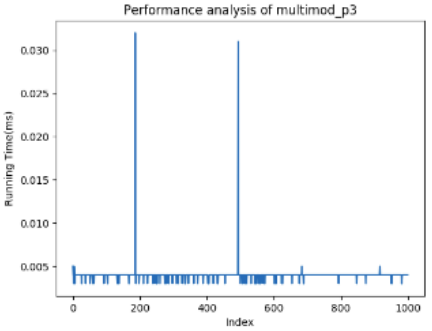
2.1. 性能初探

下面为multimod的三种实现性能的统计数据(单位:ms)

name	mean	std	25%	50%	75%	max
multimod_p1	0.624248	0.093282	0.339000	0.564000	0.624000	0.677250
multimod_p2	0.006395	0.001003	0.005000	0.006000	0.006000	0.007000
multimod_p3	0.003978	0.001265	0.003000	0.004000	0.004000	0.004000





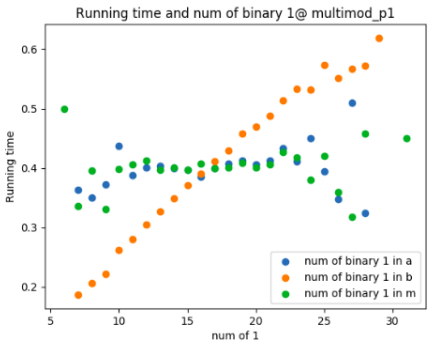


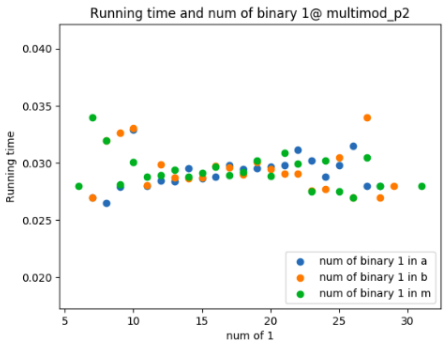
可见multimod_p1的性能较差,且运行时间随运行次数增多而逐渐降低,可能是因为运行次数增多后cache缓存更活跃,命中率更高.multimod_p2由于其$O(n)$的时间复杂度和大量位运算,性能较好,且运行时间较稳定,只略逊于常数时间复杂度的multimod_p3.

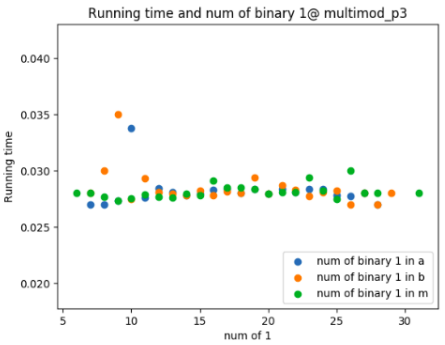
2.2. 影响性能的因素

2.2.1. 二进制位中1的数量

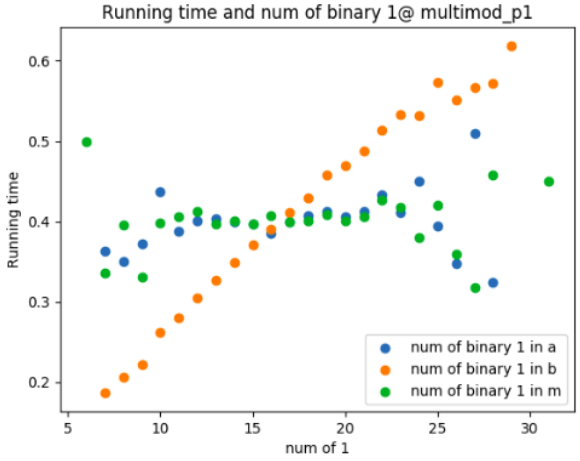
下面三图分别表示了三种multimod实现的运行时间与a,b,m二进制表示中1的数量的关系:

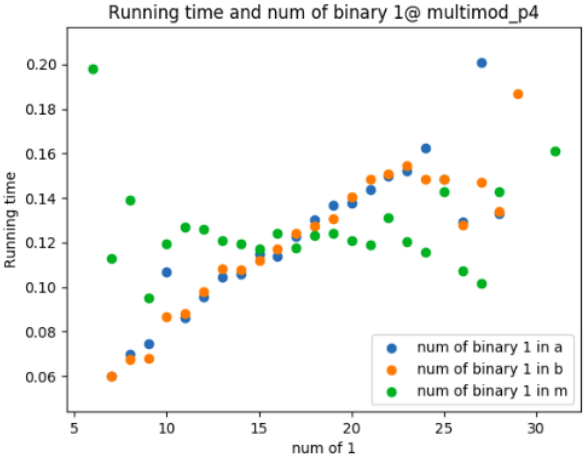






可见在multimod_p1中,运行时间只与乘数b中二进制1的数量呈明显线性关系,其余实现与a,b,m皆无明显关系.这与它们分别的代码实现有关(具体代码可查看文件夹中的p1.c, p2.c, p3.c): multimod_p1是模仿高精度乘法,逐位扫描b.若为0则跳过,若为1则将a相应的位移位后加至结果中.故1的位数越多,计算开销越大.而对乘数a,没有这样的"若为0则跳过"的机制,无论故取值为0或1都要进行同样的运算(其实可以加上这个机制,速度应该会更快,但当时做multimod_p1时没有考虑性能).另外,代码中m只出现在取模符号中,而取模运算可以由硬件直接给出结果,故运行时间与m中二进制1的个数基本无关.在multimod_p2中,若a的某一位为1,代码只会多进行一步加操作,主要耗时的移位,取余等操作与a,b每位的取值无关,故运行时间与a,b中二进制1的个数基本无关,但仍然有微弱的线性关系,同时线性组合系数较小.multimod_p3直接是$O(1)$算法,与a,b,m中二进制1的取值自然基本无关.
为验证我的想法,我对multimod_p1进行改进,加入对a各位是否为0的判断,记该实现为multimod_p4.实验结果如下:

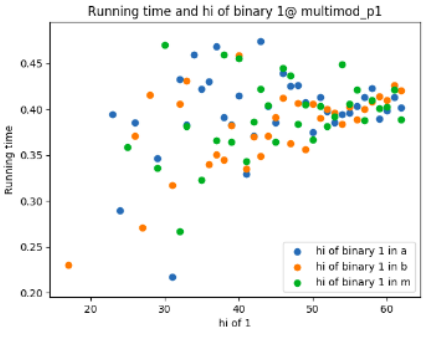


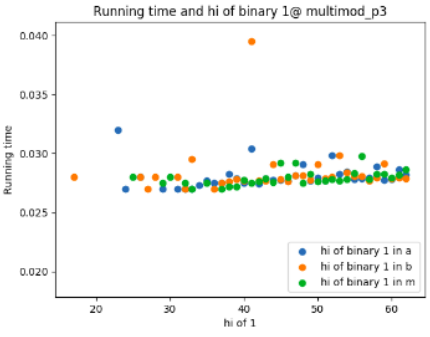


可见修改后的实现的运行时间与a,b中二进制1的数目都直接相关,实验确实验证了我的想法.

2.2.2. a, b, m 的大小

下面三图分别表示了三种multimod实现的运行时间与a,b,m二进制表示中最高位1的位置的关系:





其中a,b,m二进制表示中最高位1的位置是对a,b,m的大小的一个较好估计.从实验结果可以看出,在multimod_p1中,a,b,m的大小与程序运行时间都有大致的线性关系.对a,b而言,这是由于其大小控制了高精度乘法中循环的次数,从而影响程序运行时间.对m而言,上述线性关系更加明显.一种可能是m与取余运算直接相关,而取余运算固化在硬件中,可推测取余运算由若干触发器构成,且若触发器中高位皆为0,其信号传输时间极快.另一种可能是代码中每一步循环都加入了对m取模,m的大小会影响下一次循环中乘数的大小.
而在multimod_p2和multimod_p3中a,b,m的大小与运行时间无明显联系.尤其是multimod_p2,在a,b,m增大时运行时间的方差明显减小.这可能与我的随机数发生程序有关:在统计a,b,m各自不同的最高位1位置与运行时间的联系时,并没有控制另外两个变量的取值基本不变,而是仍让它们随机产生.这导致当待统计统计变量较小时,运行时间受其余变量的变化,影响较大;当待统计变量较大时,其自身可基本bound住总运行时间,受其余变量影响较小.

2.3. 对统计数据的进一步分析

定义变异系数 $c_v = \frac{\sigma}{\mu}$[1], 其中 σ 为变量标准差, μ 为变量均值, 可以计算出

$$
\begin{aligned}
c_v(\mathrm{p1}) &= 6.69 \\
c_v(\mathrm{p2}) &= 6.38 \\
c_v(\mathrm{p3}) &= 3.14
\end{aligned}
$$

可见multimod_p1与multimod_p2的变异系数c_v较为接近,与multimod_p3的变异系数相差较大.联想到multimod_p1与multimod_p2的共同点是代码的主体为循环结构,且这恰恰是multimod_p3不具有的特点,我猜测该现象的原因是每次循环cache的命中率不尽相同,形成运行时间的方差σ.而运行时间的均值越大,循环次数就越多,cache命中率对方差的影响也越大.所以方差σ与均值μ的比值某种程度上反映了一些与cache命中率相关的量.为验证我的猜测,我修改了multimod_p1的实现,加入对a每位是否为0的额外判断.若猜测正确,运行时间的变异系数应与multimod_p1,multimod_p2相仿.实验结果如下:

name	mean	std	25%	50%	75%	max
multimod_p1	0.624248	0.093282	0.339000	0.564000	0.624000	0.677250
multimod_p1_modified	0.213345	0.046218	0.089000	0.183000	0.209500	0.239000

可求得

$$
c_v(\mathrm{p1}') = 4.62
$$

实验结果并没有证实我的猜测.这恰恰体现出计算机系统的复杂性:一个程序的运行真实运行时间与太多因素有关,仅靠朴素的计算机系统知识似乎无法很好地预测.

Reference

1. Coefficient of variation, Wikipedia