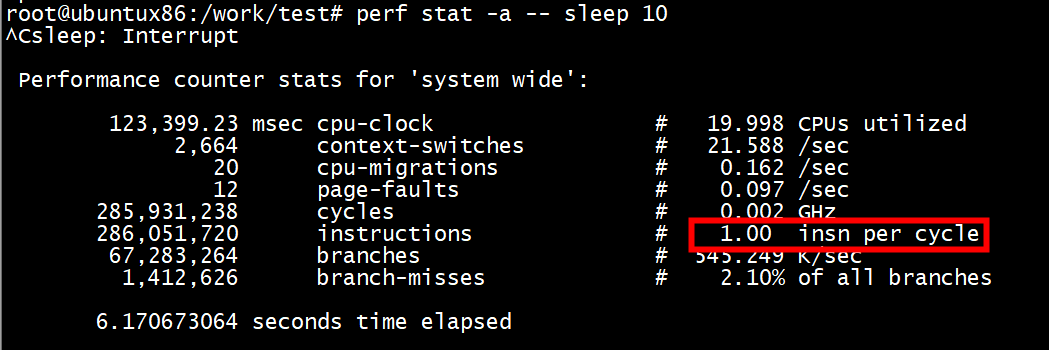
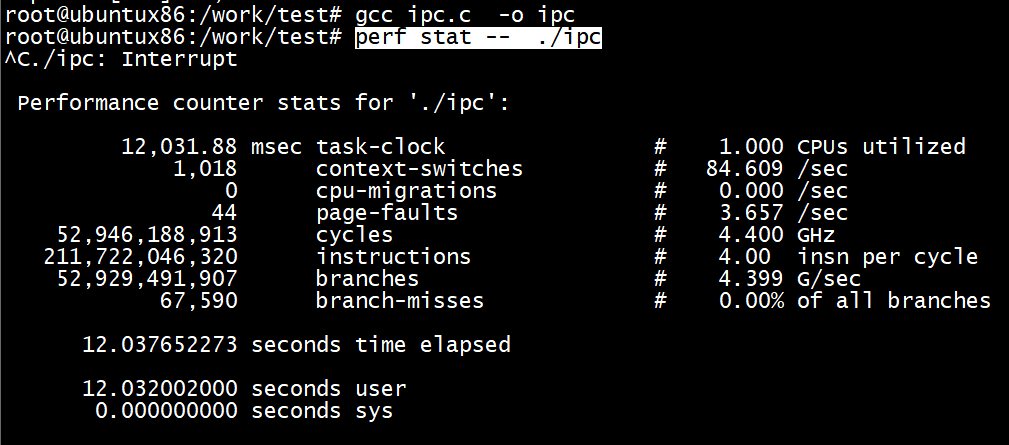
# insns per cycle

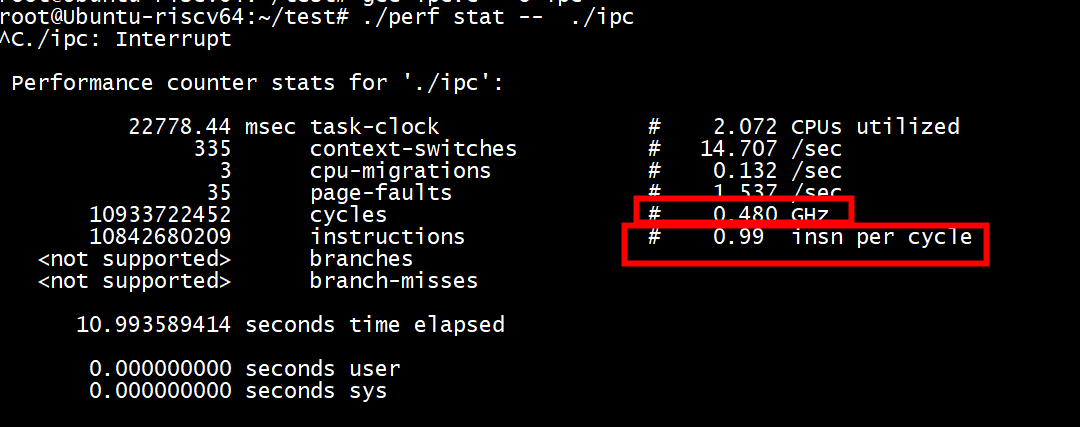
|  |
| --- |
| perf stat -a -- sleep 10 |
| perf nop |
| ./perf stat -- ./ipc |

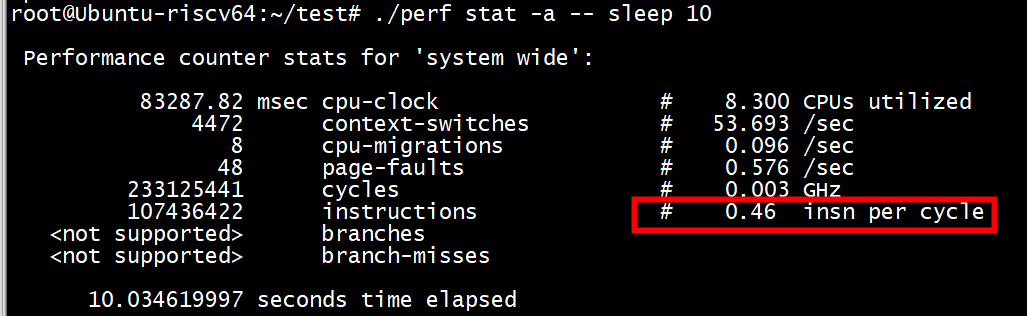
## X86

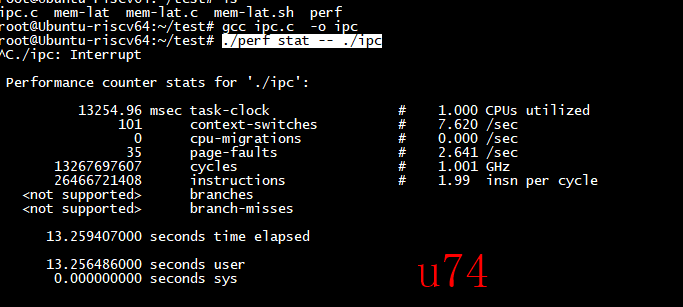




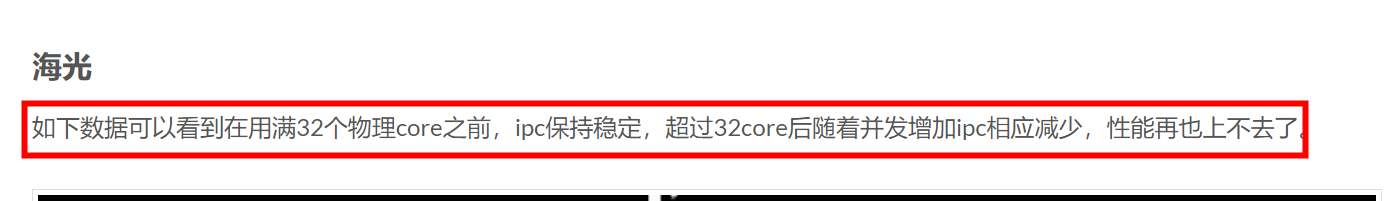
## Riscv







## ipc和并发



## 1.3 参考

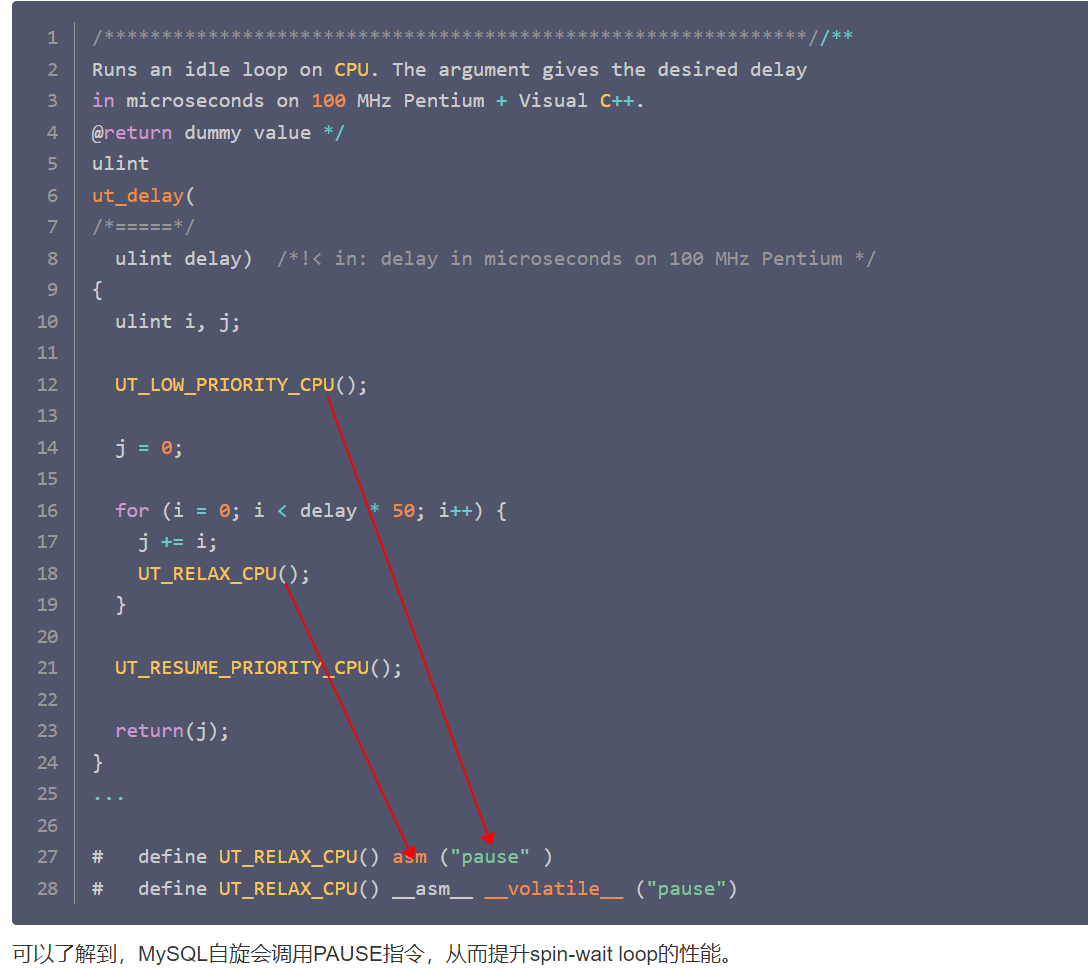
Perf IPC以及CPU性能

https://zhuanlan.zhihu.com/p/385519404

# Pause

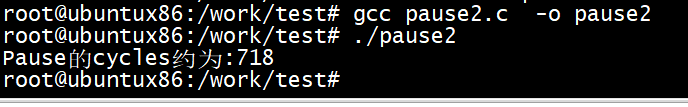
|  |
| --- |
|  |
| ./pause2 |

## Pause使用



## Pause分析

在Skylake的架构中，将pause由10个时钟周期增加到了140个时钟周期。主要用在spin lock当中因为spin loop 多线程竞争差生的内存乱序而引起的性能下降。pause的时钟周期高过了绝大多数的指令cpu cycles，那么当我们利用perf top统计cpu 性能的时候，pause会有什么影响呢？



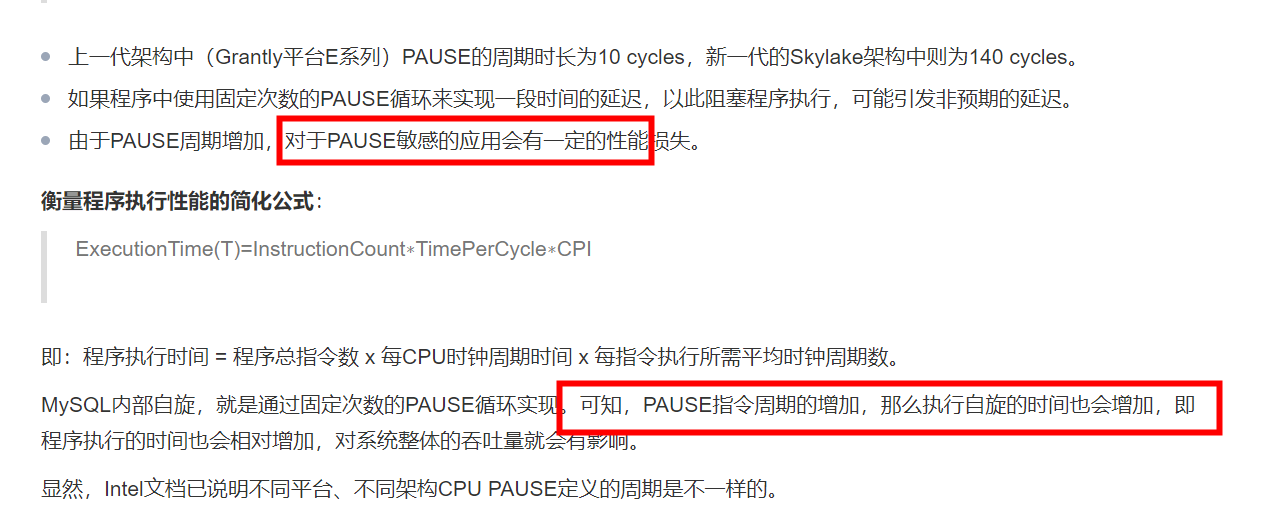
**elapsed / 100的值越小越好**

问题总结：  
我们知道perf top是通过读取PMU的PC寄存器来获取当前执行的指令进而根据汇编的symbol信息获得是执行的哪条指令。所以看起来CPU在执行pause指令的时候，从PMU中看到的PC值指向到了下一条指令，进而导致我们看到的这个现象。通过查阅《Intel® 64 and IA-32 Architectures Optimization Reference Manual》目前还无法得知这是CPU的一个设计缺陷还是PMU的一个bug(需要对pause指令做特殊处理)。不管怎样，这个实验证明了我们统计spin lock的CPU占比还是准确的，不会因为pause指令导致PMU采样出错导致统计信息的整体失真。只是对于指令级的CPU统计，我们能确定的就是它把pause的执行cycles 数统计到了下一条指令。

补充说明： 经过测试，非skylake CPU也同样存在perf top会把pause(执行数cycles是10)的执行cycles数统计到下一条指令的问题，看来这是X86架构都存在的问题。







## 参考

Intel PAUSE指令变化影响到MySQL的性能，该如何解决？

<https://cloud.tencent.com/developer/news/614886>

# CPU-bound(计算密集型) 和I/O bound(I/O密集型)

I/O密集型 (IO-bound)  
I/O bound 指的是系统的CPU效能相对硬盘/[内存](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/sinat_36053757/article/details/_blank)的效能要好很多，此时，系统运作，大部分的状况是 CPU 在等 I/O (硬盘/内存) 的读/写，此时 CPU Loading 不高。  
CPU bound 指的是系统的 硬盘/内存 效能 相对 CPU 的效能 要好很多，此时，系统运作，大部分的状况是 CPU Loading 100%，CPU 要读/写 I/O (硬盘/内存)，I/O在很短的时间就可以完成，而 CPU 还有许多运算要处理，CPU Loading 很高。

计算密集型 (CPU-bound)  
在多重程序系统中，大部份时间用来做计算、逻辑判断等CPU动作的程序称之CPU bound。例如一个计算圆周率至小数点一千位以下的程序，在执行的过程当中

绝大部份时间用在三角函数和开根号的计算，便是属于CPU bound的程序。  
It is because the performance characteristic of most protocol codec implementations is CPU-bound, which is the same with I/O processor threads.

根据以上分析，可以认为通常情况下，大部分程序针对某个特定的性能metric而言  
都可分为CPU bound 和 I/O bound两类。  
CPU bound的程序一般而言CPU占用率相当高。这可能是因为任务本身不太需要访问I/O设备，也可能是因为程序是多线程实现因此屏蔽掉了等待I/O的时间。  
而I/O bound的程序一般在达到性能极限时，CPU占用率仍然较低。这可能是因为任务本身需要大量I/O操作，而pipeline做得不是很好，没有充分利用处理器能力

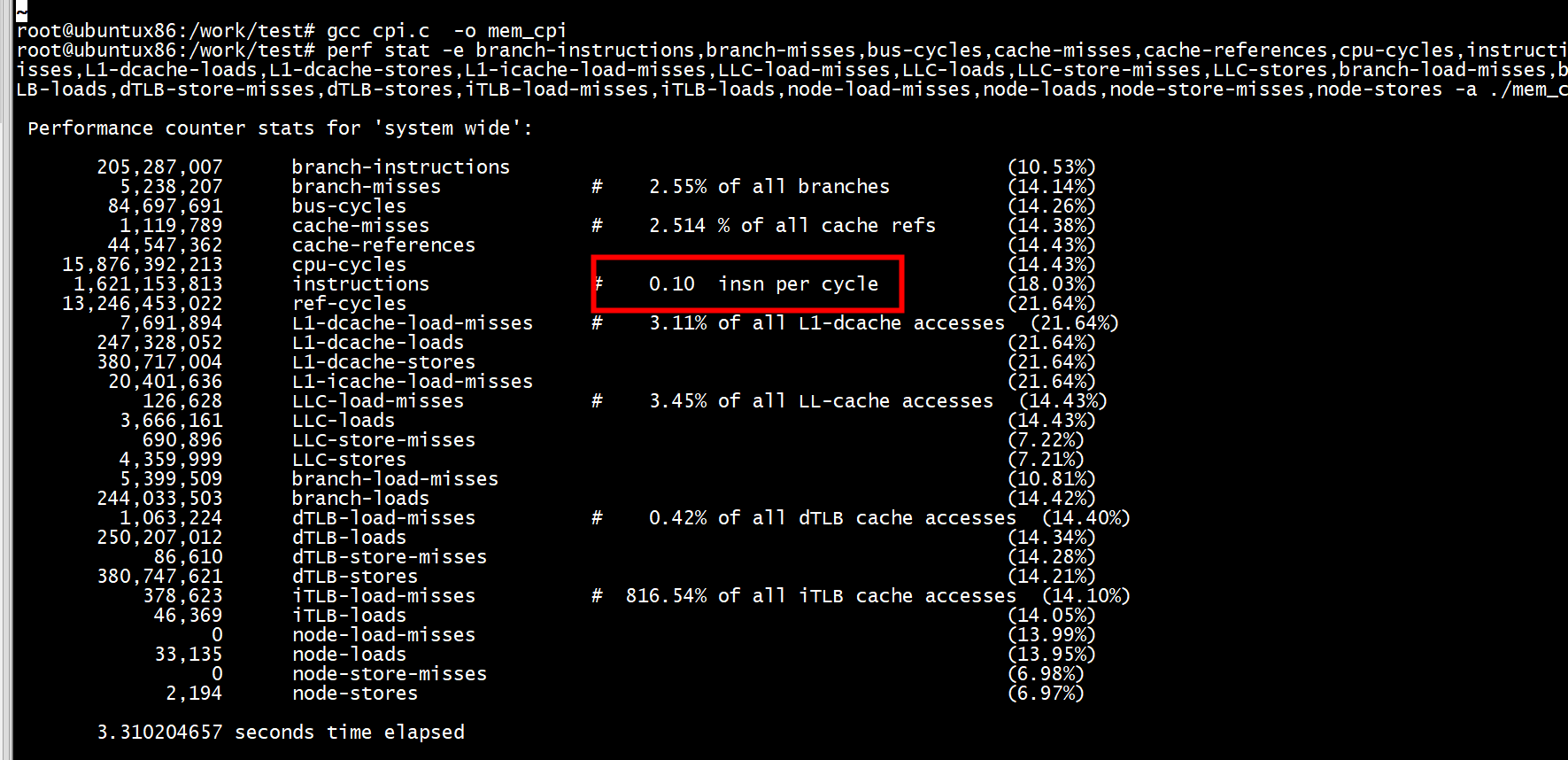
## 测试benchmark

让我们写一个人造的小程序，展示在 Linux 下 CPI 火焰图的使用。

这是一个最简的小程序，其中包含如下两个函数：

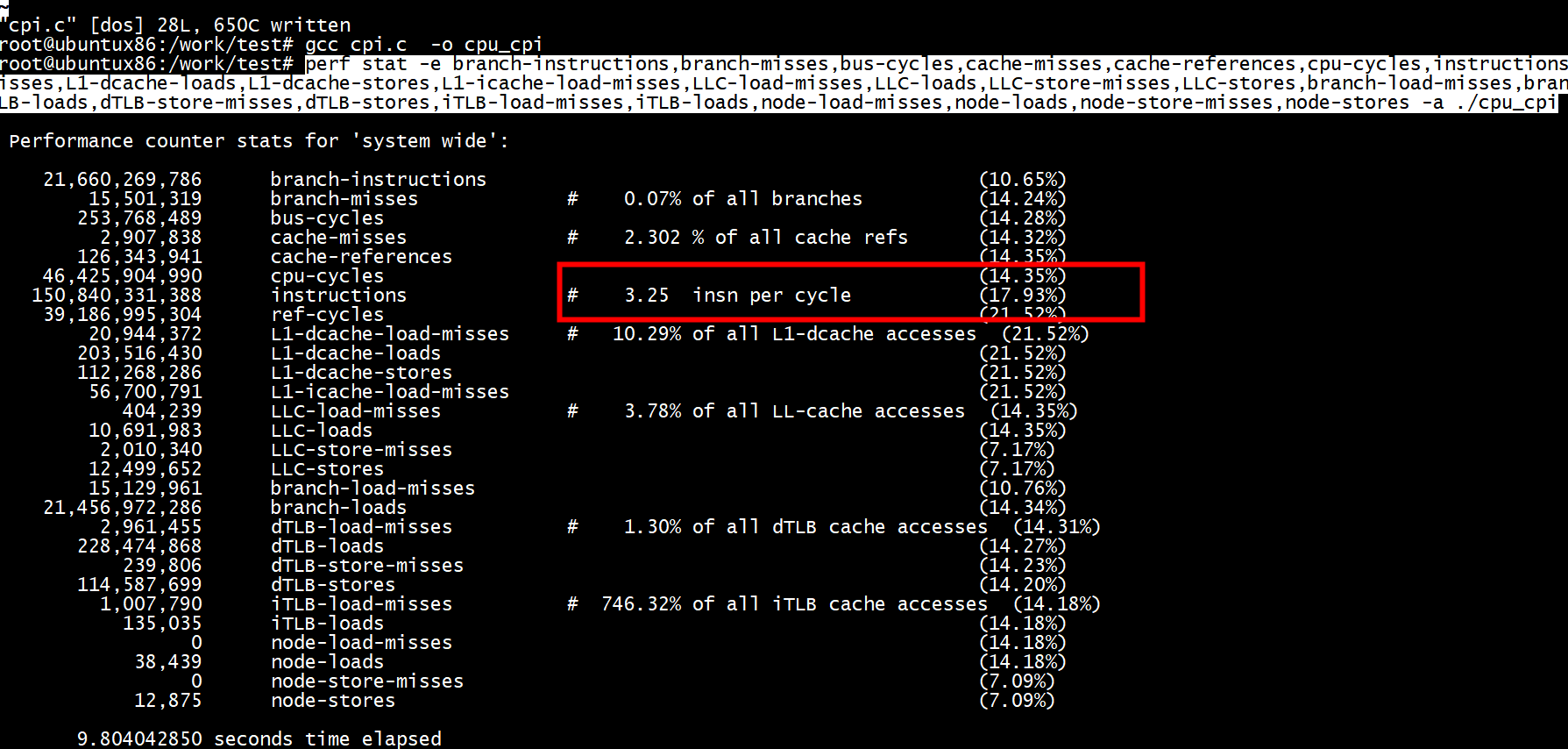
1. cpu\_bound 函数主体是 nop 指令的循环；由于 nop 指令是不访问内存的最简指令之一， 因此该函数 CPI 一定小于 1，属于典型的 CPU 密集型的代码。
2. memory\_bound 函数使用 \_mm\_clflush 驱逐缓存，人为触发程序的 L1 D-Cache Load Miss。 因此该函数 CPI 必然大于 1，属于典型的 Memory 密集型的代码。

perf stat -e branch-instructions,branch-misses,bus-cycles,cache-misses,cache-references,cpu-cycles,instructions,ref-cycles,L1-dcache-load-misses,L1-dcache-loads,L1-dcache-stores,L1-icache-load-misses,LLC-load-misses,LLC-loads,LLC-store-misses,LLC-stores,branch-load-misses,branch-loads,dTLB-load-misses,dTLB-loads,dTLB-store-misses,dTLB-stores,iTLB-load-misses,iTLB-loads,node-load-misses,node-loads,node-store-misses,node-stores -a ./mem\_cpi

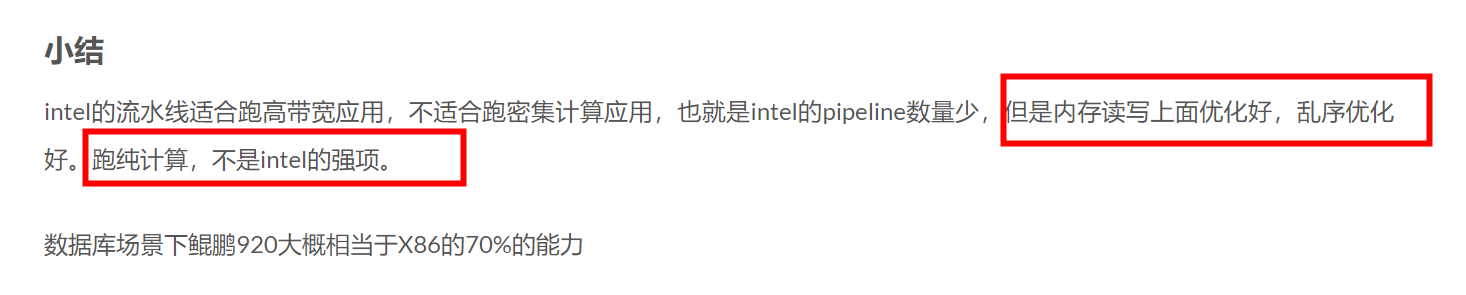


此方法可以用来观察不同架构的IPC

perf stat -e branch-instructions,branch-misses,bus-cycles,cache-misses,cache-references,cpu-cycles,instructions,ref-cycles,L1-dcache-load-misses,L1-dcache-loads,L1-dcache-stores,L1-icache-load-misses,LLC-load-misses,LLC-loads,LLC-store-misses,LLC-stores,branch-load-misses,branch-loads,dTLB-load-misses,dTLB-loads,dTLB-store-misses,dTLB-stores,iTLB-load-misses,iTLB-loads,node-load-misses,node-loads,node-store-misses,node-stores -a ./cpu\_cpi



## 小结



## riscv测试

./perf stat -e branch-instructions,branch-misses,bus-cycles,cache-misses,cache-references,cpu-cycles,instructions,ref-cycles,L1-dcache-load-misses,L1-dcache-loads,L1-dcache-stores,L1-icache-load-misses,LLC-load-misses,LLC-loads,LLC-store-misses,LLC-stores,branch-load-misses,branch-loads,dTLB-load-misses,dTLB-loads,dTLB-store-misses,dTLB-stores,iTLB-load-misses,iTLB-loads,node-load-misses,node-loads,node-store-misses,node-stores -a ping 10.11.11.82 -l 1500

## 参考

 Linux 下 CPI 火焰图

<http://oliveryang.net/2018/03/linux-CPI-flamegraph/>

# Cache

|  |
| --- |
| root@Ubuntu-riscv64:~/test# gcc bi\_search.c -o bi\_search  root@Ubuntu-riscv64:~/test# gcc bi\_search.c -DDO\_PREFETCH -o bi\_search\_pre  root@Ubuntu-riscv64:~/test# time ./bi\_search |
|  |
|  |

## 参考

一文带你剖析cache对规范代码至关重要！

<https://zhuanlan.zhihu.com/p/488056622>