[ATC 16] Unlocking Energy

Babak Falsafi, Rachid Guerraoui, Javier Picorel, and Vasileios Trigonakis, École Polytechnique Fédérale de Lausanne (EPFL)

概要：

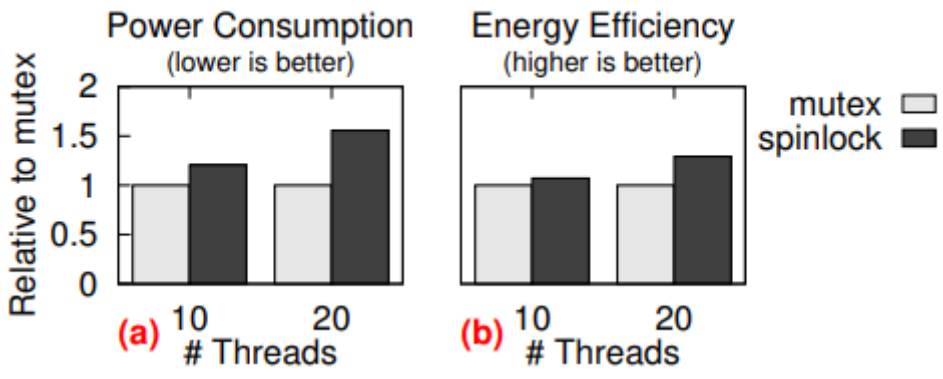
从系统功耗出发，找到了既不损耗吞吐率又提高能源效率的算法mutexee

Backgorund & Motivation

- 几十年来吞吐率一直是衡量计算系统效率的主要指标，但是现在降低系统功耗也被认为十分主要
  - 各种研究估计，在2010年，数据中心贡献了超过2%的美国总用电量
- 现有的一些降低功耗的硬件或软件技术需要改变硬件，安装新的调度程序或运行时系统，甚至重建整个系统。
  - 如硬件技术的时钟门控（clock gating), 电源门控（power gating)，软件技术的 power-aware schedulers
- 优化锁是一种非常有效的节约能源的方法
  - 设计减少能量消耗的锁定方案会影响许多软件系统
  - 锁方案的选择对能耗有显著影响，同步的主要后果是让一些线程相互等待——这是一个节省能源的机会
  - 锁抽象定义很好，可以直接替换实现它的算法，而不需要修改系统的其他部分

Survey

- mutex和spinlock是锁的两种最常见的方式，它们的功耗差别是什么样的
  - mutex在等待一段时间后会直接睡眠。
  - spinlock则是一直循环等待

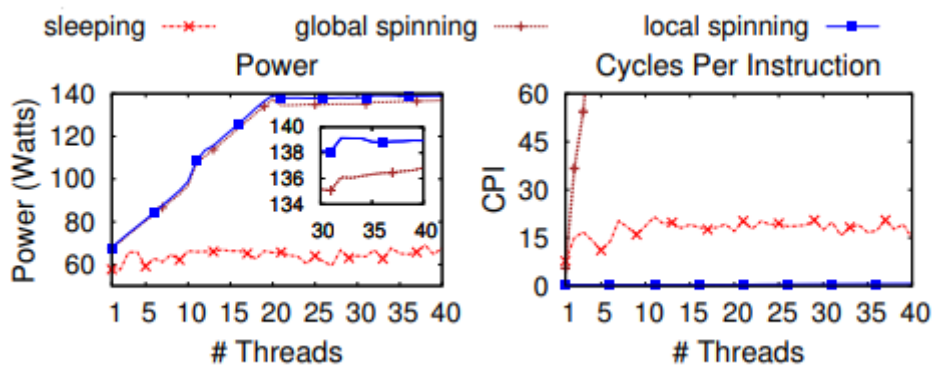


- 如上图所示，自旋等待的spinlock 消耗了更多的能源，在20个线程的时候比mutex多消耗了50%，然而在能源效率（能源效率是指完成单位任务时消耗的能源量）方面却是mutex的1.5倍，这是由于spinlock的吞吐量达到了mutex的2倍
- 初步结论：
  - 忙等提高了能源消耗
    - 在忙等待的情况下，底层硬件上下文保持活动状态。
    - 目前没有有效的方法减少繁忙等待的功耗

        - 降低这些循环成本的传统方法，即x86种的 pause 指令，实际上增加了功耗（原因后面讲）
        - monitor/mwait指令需要内核级特权，因此在用户空间中使用它们会产生很高的开销。
        - 传统的DVFS技术用于降低核心的电压和频率(从而降低其功耗)，粒度太粗
    - sleep确实能节省能源
        - 如果线程在等待一个锁后处于休眠状态，自然可以节省能源
    - 进入到睡眠状态会降低能源效率
        - Pthread mutex lock中的sleep主要通过futex调用实现
        - 在很多现实的场景中，futex调用开销抵消了在繁忙的等待中sleep所带来的能源效益，从而导致更糟糕的能源效率。
            - mutex在调用futex之前仅自旋等待几百个cycle，然而调用futex 唤醒一个沉睡的线程需要7000个cycle
            - mutex的spin-then-park策略没有调整到考虑这些开销，线程调用futex来休眠，但却立即被唤醒，损害了能源效率
    - 因此，一些线程应该长时间睡眠
        - 在高争用的情况下，让部分线程长时间睡眠，让活动的线程数减少。
        - 这样做会带来
            - 锁的争用减少，吞吐量提高
            - 部分线程长时间睡眠，更高的尾时延

关键设计

- 使用mfence替代pause
    - 测试场景：所有的线程都在等待一个不释放的锁，下图展示了不同策略的CPI（一条指令执行所需的平均CPU周期数） 和功耗



    - 局部自旋比全局自旋多消耗3%的能量。根据CPI图:全局旋转对锁的共享内存地址执行原子操作，导致非常高的CPI（高达530），而在本地自旋中每个线程每个周期都能执行一次L1 load
    - 自旋等待中，传统的方法是通过pause指令降低功耗，然而本篇论文发现pause指令在锁中不仅不节省能源，还会损害能源效率(提高4%的能源效率)。
        - 根据英特尔的软件开发人员手册 “Inserting a pause instruction in a spin-wait loop greatly reduces the processor’ s power consumption”
        - pause指令 (local-pause) 会将 CPI 提高至 4.6。但却提高了能源效率（具体原因论文没给解释）

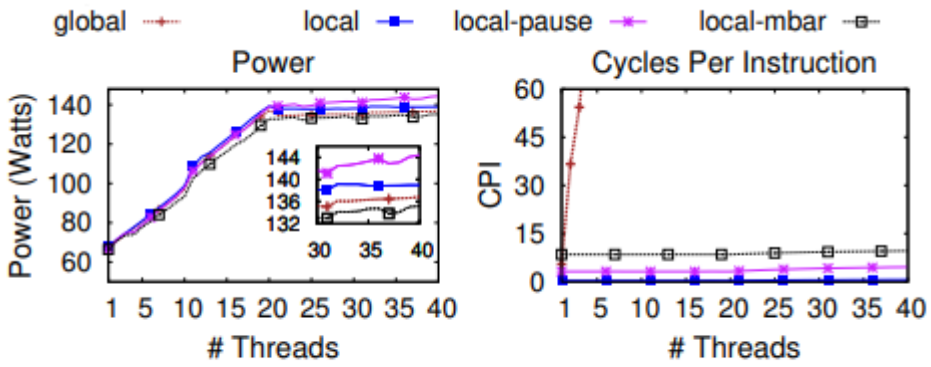
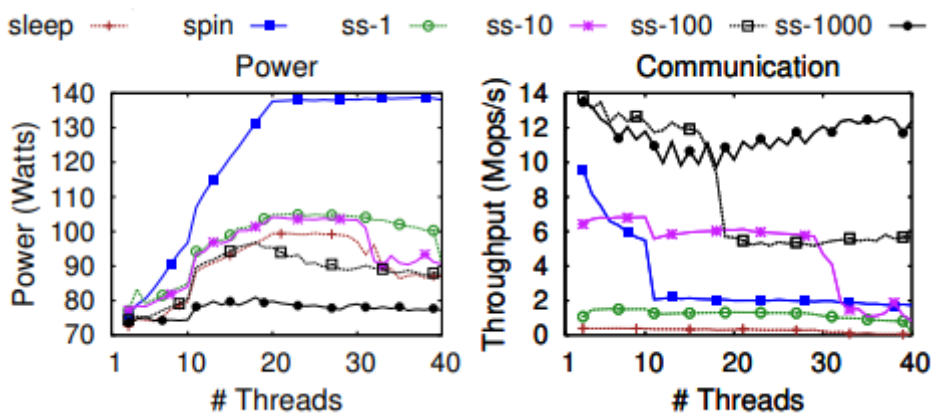


Figure 4: Power consumption and CPI while spinning.

- 本地旋转的CPI非常低的原因是现代处理器的猜测执行（speculatively execute）和乱序执行。
- 避免投机性执行的一种方法是插入一个满的或负载的内存屏障。这样，load只在前一个load退出时执行，并且依赖于它的指令(test和jump)也被暂停。
- 结果(local-mbar)表明，该内存屏障将局部自旋的功耗降低到比全局自旋更低的程度(7%)。

- 加长忙等时间
  - 实验场景：两个线程忙等，其它线程睡眠，每个线程的忙等都有一个quota， quota消耗完后后唤醒其他线程，下图展示了不同策略下的power消耗和吞吐量。其中策略ss-T表示每执行T次在忙等中获得锁后执行一次futex调用



  - 结果表明，执行越不公平，吞吐量越高，能源消耗越小。在T = 1000时， 大部分线程休眠2M个cycle，而直接sleep的策略，睡眠周期小于9w个cycle

- mutexee:一个优化后的mutex锁

	MUTEX	MUTEXEE
lock	for up to ~ 1000 cycles spin with `pause`	for up to ~ 8000 cycles spin with `mfence`
	if still busy, sleep with `futex`	
unlock	release in user space (`lock->locked = 0`)	**wait in user space**
	wake up a thread with `futex`	

  - 相比mutex锁，mutexee自旋8000个cycle
  - Spin witeh mfence
  - 在放锁后调用futex wake前等待一会。
    - 在mutex中，放锁后会立即唤醒睡眠的线程，但如果有其它线程在被唤醒的线程能够执行前拿锁，被唤醒的线程可能迟迟拿不到锁又进入到睡眠状态。
  - 模式切换，如果执行futex_wake和执行busy waiting获取锁的比例很高，切换到mutex模式
  - 降低尾时延

- 在高争用的情况下可能会产生高尾时延，但论文声明在实际系统中未观察到尾时延的大幅上升
- 提供降低尾时延的方法，在futex_wait时加入timeout，长时间睡眠后唤醒，唤醒后一直自旋不会再次睡眠
- 该方法会大大降低吞吐量

Lock	Throughput Kacq/s	TPP Kacq/Joule	Max Latency Mcycles
MUTEX	317	4.0	2.0
MUTEXEE	855	10.9	206.5
MUTEXEE timeout	474	6.5	12.0

效果

实验环境：有40个硬件线程的intel机器

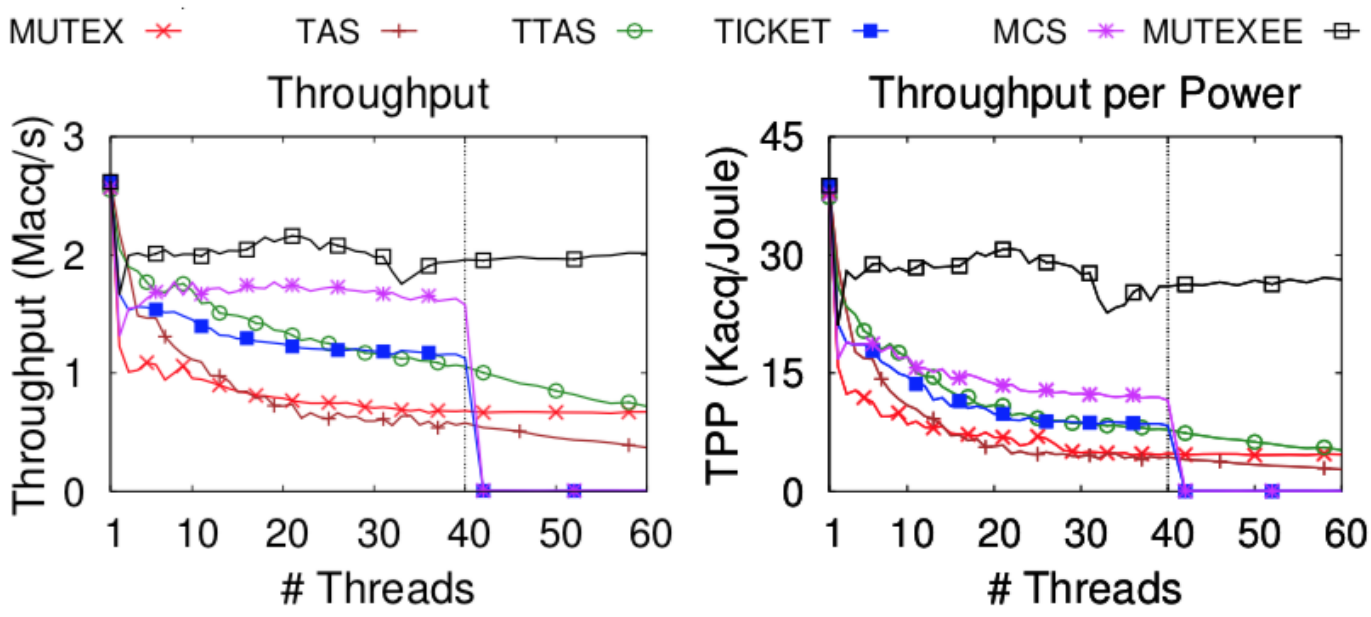


Figure 11: Using a single (global) lock.

- 左图为吞吐量，右图为能源效率TPP（throughput per power）
- 图中是不带timeout的MUTEXEE。可以看到在性能和能效上都有不错的表现与可扩展性。但是在thread比较多时，MUTEXEE拥有远超MUTEX的Tail latency（因为没有timeout）。mcs lock在40个核的时候出现性能与功耗下坠，这是由于硬件线程数量限制
- MUTEXEE的不公平性在实际系统中可能不是一个主要问题:MUTEXEE只有在极端争用场景下才会导致高尾延迟，这在设计良好的系统中必须避免

工作评价

1. 在复现的时候，没能够复现出论文中吞吐量达到2.5倍的情况，只有在高争用情况下一个线程反复获取锁时有着2.5倍甚至更高的吞吐量
2. 原本代码加入timeout还是尾时延高的原因可能是唤醒时没有获取锁继续睡眠，这一点在论文中也提到了并通过在放锁时等待一段时间。修改后唤醒后的锁不会进入到睡眠状态能够很好的降低尾时延，并且吞吐量未有明显下降。总体上能够提高吞吐量30%-50% 且尾时延未有明显提升