通过减少位翻转来优化NVM系统

摘要

Introduction

- 对PCM的优化方式
  - 通过减少写次数优化PCM
    - 设计需要更少写次数的数据结构
    - 通过硬件技术比如cache来减少写
  - 然而，对于PCM来说，应该减少位翻转次数而非写次数
- BNVM如PCM遇到的两个问题：能耗和磨损，这也是应用中的主要成本来源。
  - 翻转的能耗远大于读写，因此许多控制器仅在不同于旧数据时翻转，但不能消除软件更新数据结构时的翻转需求。
  - PCM单元是单独修改的，由于不平衡写入，可能一些单元会很快被磨穿。
  - 减少比特翻转，可以节省能量并增加BNVM的寿命
- 通过优化编译器和选择数据结构比修改硬件更好
  - 实现了三种数据结构，最多可减少3.56倍翻转，同比减少能耗和增加寿命，而不需要修改硬件。
- 贡献
  - 在全周期精准的模拟器环境下实现比特位翻转计算，并研究比特翻转行为。
  - 经验证据表明，减少内存写不会按比例减少位翻转
  - 测量内存分配和堆栈帧使用等操作所需位翻转，以及减少所需位翻转的建议。
  - 修改了三种数据结构：链表，哈希表，红黑树，来减少比特翻转，评估了这种技术的有效性。
- 文章组织
  - 2:背景介绍比特位如何影响能耗和寿命
  - 3：讨论了减少比特翻转的软件技术
  - 4：针对位翻转进行评估
  - 5：评估它们的性能
  - 6：讨论了结构
  - 7：未来工作
  - 8：总结

BNVM and Bit Flips

- 存储技术优化
  - 现存研究忽略了不对称的读写能耗和避免重写未改变的单个单元能力。
- PCM和DRAM的能耗
  - DRAM功耗与内存成正比，PCM功耗与单元变化率成正比。
- 磨穿
  - 避免不必要的写增加了单元的平均寿命，但可能会加剧磨损不均衡问题，因为这种技术会增加写操作的偏差。
  - 一些现有的硬件技术可以在一个字上进行扩展，在一个字上分布"热"位，这允许我们通过比特翻转来减少磨损，减少引入的偏写问题。
- 减少BNVM中比特翻转的影响
  - 之前的工作聚焦于硬件编码：重新编码高速缓存线路。由于还需保存编码信息，性能有限。
  - 软件技术利用软件中定义的语义知识，但在内存控制器对单个高速缓存路线的有限视图不可见。
  - 从三个方面研究如何减少位翻转
    - 数据结构设计
      - 围绕指针距离的思想定义新数据结构，而不直接存储指针。数据结构比数据修改频率更高，减少也较方便。
    - 程序运行效果
      - 堆栈写入返回地址，寄存器的保存和溢出。
    - 缓存层的效果
      - 由于数据先经过缓存，了解不同缓存层和缓存大小如何影响位翻转非常重要。更加复杂的是BNVM的一致性问题，程序更频繁的将cacheline刷新到主存，使用直写缓存，或者将更复杂、硬件支持的缓存刷新协议。
  - 软件上减少比特翻转
    - XOR链表
      - 每个节点存储兄弟节点地址的异或值XOR，可以双向遍历同时指针数减少了一半。但不允许O(1)删除一个仅有一个结点指向的节点。
    - XOR哈希表
      - 使用改进的XOR链表做链式哈希，每个指针与所在节点进行XOR，实际上存储的是下一个节点的距离，以NULL结尾。
      - 第二点优化，将空链表用最低有效位为1或数据指针为空标记。初始时数据为0，删除后下一个节点指针为1。
    - XOR红黑树
      - 使用xchild代替child，xchild=child XOR parent，通过一个节点的同或来获取其余节点。迭代的调用函数以遍历树。
      - 上述技术可以推广到任何有序KV树，关键在于消除指针的额外位或使用排序属性。
    - 堆栈帧
      - 栈满或程序将状态保存在BNVM上时，寄存器以及返回地址可能写入主存。也可能使用写穿缓存减少一致性复杂性。
      - 两个函数可以以不同顺序入栈相同寄存器，如果两个函数交替将寄存器入栈，修改的比特位较少。建议制定一个寄存器入栈框架，通过保留栈使得每个元素到栈的指定位置。

不足的总结

- 优化效果不太明显，虽然提出了一个新的优化方向，但是具体的优化手段目前比较少，仅能通过数据结构和栈帧的方式进行优化，优化方案具有很大的局限性。
除此以外，对数据结构的使用也增加了代码复杂度，极大的降低了以链表为例的数据结构的独立性（即给定x大小数据，就可以直接访问全局，但优化后的代码需要给定2x数据，才能进行有效操作。）

未来工作

- 有更多的数据结构需要被优化，归结为数据移动最小化。对实际BNVM的评估来研究位翻转有启发性。

结论

- 软件消减位翻转
  - 通过数据结构来优化位翻转，而除位翻转（BNVM上新的优化目标）外，并没有理论进步或性能优势。这个新型研究理论目标不是性能，而是减少功耗同时增加寿命。任何具有显著读写差异和位级更新的内存都会收益。这种设计具有很好的向后兼容性，不破坏现有的ABIs
- 缓存影响
  - 缓存只是扩展了数据结构的预热期，因此缓存的影响时间可预测的。虽然XOR链表较双链表产生更多写入，但XOR链表的额外写总是命中了缓存。

性能分析

- 在减少位翻转的同时不能产生很大的性能影响
- XOR链表、哈希表上性能影响不大。
在红黑树上，延迟xrbt<rbt<xrbt-big

内存特性结果

- 目标
  - 证明自己的位翻转优化时有效的
  - 理解不同的系统和程序组件如何影响位翻转
  - 使用clwb指令模拟XOR链表和堆栈写实验的一致性特点。
- 内存分配

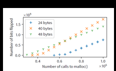


Figure 3: Bit flips due to calls to malloc. Allocation size of 16 bytes is not shown because it matches with 24 bytes.

  - 分配器存储分配的元数据，返回指针也有助于位翻转。我们按照数据结构进行了大小为16，24，40，48字节的100000次分配。每次分配的位翻转如图3所示。更大的空间导致更多的位翻转，40字节先少于48字节，后期由于不对称而大于48字节。在缓存影响较小后，趋势成为线性。24和16字节与48有相同的位翻转/每分配。16，48字节每次花费了2位翻转，40每次花费了3次位翻转。这个优化较小。
- XOR链表

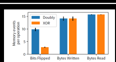


Figure 4: Memory characteristics of of XOR linked lists compared to Doubly-Linked Lists.

  - 以5：1的比例随机在头插入和在尾pop测试XOR链表和双链表。XOR链表减少了3.56x位翻转，而写入字节数相同。XOR链表的读取比双链表更多，但因为相同，说明读取总发生在cache中。
- XOR哈希表

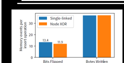


Figure 5: Memory characteristics of XOR hash table variants under Zipfian workload.

  - 对比单链表和XOR链表实现的哈希表，运行Zipfian负载，其中80%的更新发生在20%上，每次迭代删除已有键，插入新键。XOR链表产生的位翻转较少。这是由于哈希表通常是短链。减少的量主要来自于最低有效位的指示列表有效。这种数据结构几乎没有优化空间。
- XOR红黑树

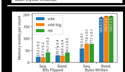
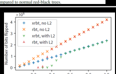
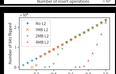


Figure 6: Memory characteristics of XOR red-black trees compared to normal red-black trees.

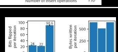
  - 在随机/顺序插入测量了正常RBT和双节点XRBT以及膨胀到正常大小的大XRBT。每项包括到一百万和一个随机值。每个xrbt和xrbt-big减少了1.92x的位翻转在顺序写上，减少了1.47x位翻转在随机写上。可以看到写数目和位翻转不相关。
- 缓存影响



  - 研究L2缓存存在与否的影响，随着数据规模的变大，L2达到饱和后，两种数据结构的位翻转都进入稳定的线性增长。这表明长期来说，L2无法减少位翻转。



  - 改变缓存大小的影响。
对于不同大小的L2，延迟的大小和缓存大小成正比，但饱和后呈线性增长，这证实了L2只短期影响位翻转。
- 栈帧



  - 对三种操作进行两两组合循环：
x:按序push7个寄存器，y:乱序push7个reg，s:按s序push2个reg。
乱序相比同序大了4倍左右的位翻转，表明帧的布局一致性具有显著影响。