# 想法

1. Banshee的替换算法对于SPEC CPU中lbm这种程序很不友好（缺失率70.9%），因此可以根据set dueling或者miss rate动态决定替换算法，提高lbm程序的性能。这类程序有个特点：spatial locality比较好，并且对于单个页面访问频率不高，导致FBR替换算法无法起到作用，也就是某个页面访问频率还没达到FBR的阈值就被替换出去了。
2. Banshee等，没有充分利用片外Dram的带宽，某些情况下性能比Cache-Only的好也是因为对片外Dram的带宽利用更充分。是否可以考虑推测的从DramCache写回Dram，充分利用Dram带宽。
   1. 方法1：Banshee只有在访问片上内存的TagBuffer后才能知道是否命中片上内存，如果把命中信息直接放到L3级别，当L3miss的请求知道是否命中片上内存时，可以不访问Tagbuffer，直接访问Dram，记录该访问频率，当达到阈值再把其放到片上内存中，这样可以充分利用片外内存。在L3处可以使用包含关系的stream register用来猜测是否命中片上内存。
   2. 方法2：对于写命令，在内存带宽紧张时，可以直接写到片外Dram，并记录下其写的位置，最后再跟片上Dram拼起来，不会导致错误。
3. TDC更适合Pointer chasing程序，Banshee更适合mcf\*16，但是在mcf\*4、mcf\*8时，性能也不如TDC，因此要在TDC和Banshee之间动态切换。
4. 通过DC的缺失率选择tagless还是banshee的替换算法及粒度，默认banshee的，如果缺失率高就切换成tagless以footprint为单位的next way替换算法，否则就一直默认banshee的方法。问题：如何从tagless切换回来？
5. 如何能让DC的性能比cacheonly好？只有当dram cache size不是限制的情况下才有可能，因为cacheonly是以dram cache无限大为例的。什么情况下有这种可能？当程序footprint足够小时，dram cache size不是性能瓶颈了，此时如果能把片外内存也用上，性能可能就比cacheonly的好。这种情况下如何充分利用片外dram进一步提升性能？如果从llc出来的请求就知道是否命中dc，可以绕过访问tagbuffer的延迟，在llc中存一个tagbuffer的缓存？ 或者在每一个cacheline中存一个bit标识是否命中dc，如果tagbuffer中有remap就把remap信息传回llc进行flush。或者用stream register，如果不命中则标识肯定不在dc里，此时可以直接访问dram，为了把热数据放到dc中，还需要在llc中维护计数器和页面地址，多次不通过dram cache访问片外dram时就把数据swap进dram cache
6. 把所有speccpu都测一遍，改完后以health16 lbm16 mcf16 mix3\*16测试

TDC和Banshee的主要区别：

|  |  |  |  |
| --- | --- | --- | --- |
|  | TDC | Banshee | 优化点 |
| 替换算法 | Next line | FBR | 动态选择 |
| 替换粒度 | Footprint | Page | 动态选择 |
| TLB coherence | No coherence | TagBuffer | LLC中添加stream Register消除延迟 |
| 充分利用片外DRAM | No | No | 有20%的片外内存带宽可以用 |
| 相连度 | 一个set，全相连 | 4路 |  |

我的改进方案：

|  |  |
| --- | --- |
|  |  |
| 动态变化的替换算法及粒度 | 动态选择：TDC/Banshee，在4路cache里选择一路做成footprint cache，用LRU替换算法以footprint为粒度，剩余的3路继续以页面为粒度使用FBR替换算法，如下图所示。 |
| 提前查DramCache Tag，减少延迟 | 在LLC中添加stream Register，或者在目录一致性协议中实现 |
| 通过预取器充分利用片外DRAM | 先用实验证明，把片外DRAM也利用起来时的效果；尽量减少Dram/DC的数据同步的开销即可。把用于记录页面冷热的计数器放到L3，减少为了访问片外计数器的次数  预取时，先区分片上DramCache是不是dirty，如果不是，则从片外Dram预取，充分利用片外Dram的带宽 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Footprint cache way selection | Footprint dirty bits | Footprint Valid bits |  |  |  |  |
| 01 | [15:0] | [15:0] | Way0 | Way1 | Way2 | Way3 |

# 问题：

1. 如果页表信息里提示不在DramCache中，但是事实上在DramCache中，Banshee是否解决了这个问题？是否模拟了这种情况？也就是通过页表发现DramCache miss，但是数据确实在DramCache中，
2. Tagbuffer清空的延迟如何计算的？没有详细展示，只是给了一个数据
3. TLB shootdown的代价在哪里模拟的？