计算机系统结构 Cache 替换策略设计实验

ZhaohengLi 201750025
cainetatum@foxmail.com
15801206130

2020 年 3 月 18 日

1 实验背景

在终端设备上随着游戏的画质、特效、帧率需求的日渐增长，使得这些场景下的访存密集型特点越发显著，对内存的带宽需求不断增长，最终造成游戏场景下芯片功耗的大幅增加。现有处理器主要采用 Cache-Memory 层次存储结构，通过 Cache 缓存频繁访问的程序和数据，减少对内存的访问。然而目前在游戏场景下，SoC 芯片 System Cache 的命中率非常低，需要重新优化 Cache 替换策略，提高 Cache 的命中率。

2 实验目的

深入理解 Cache 结构设计及 Cache 替换策略的原理，阅读 Cache 替换策略方面的最新研究进展，设计自己的 Cache 替换策略，提高游戏场景下的 Cache 命中率。

3 主流替换策略设计思想

3.1 Random 替换策略

随机替换策略不考虑 Cache 存储的情况，简单地根据一个随机数选择一块替换出去。

随机替换策略在硬件上容易实现，速度非常快，缺点则是命中率方面表现较差。

3.2 FIFO 替换策略

FIFO(First in First out) 算法核心思想为先进先出，即如果一个数据最先进入缓存中，则应该最早淘汰掉。也就是说，当缓存满的时候，应当把最先进入缓存的数据给淘汰掉。

这种替换策略实现简单，操作快速，但是在命中率方便表现一般。

3.3 LRU 替换策略

LRU(Least Recently Used) 算法是把 CPU 近期最少使用的块替换出去。这种替换方法需要随时记录 Cache 中各块的使用情况，以便确定哪个块是近期最少使用的块。每块设置一个计数器，Cache 每命中一次，命中块计数器清零，其他各块计数器增 1。当需要替换时，将计数值最大的块换出。

这种替换策略在命中率方面表现很好，但是在每一次访问 Cache 时需要较大的计算量，时间复杂度较高。

3.4 LFU 替换策略

LFU(Least Frequently Used) 算法将一段时间内被访问次数最少的那个块替换出去。每块设置一个计数器，从 0 开始计数，每访问一次，被访块的计数器就增 1。当需要替换时，将计数值最小的块换出，同时将所有块的计数器都清零。

这种替换策略将计数周期限定在对这些特定块两次替换之间的间隔时间内，不能严格反映近期访问情况，新调入的块很容易被替换出去。

3.5 RRIP 替换策略

RRIP(Re-Reference Interval prediction) 算法核心思想为将访问间隔较小的块留在 Cache 中，从而提高命中率。维护 Mbits 记录 PPRV，选择 PPRV 值为 2 的 M 次方减一的项替换出去，如果没有该项，则每项 PPRV 值 +1，之后重复扫描。新进块 PPRV 置为 2 的 m 次方-2，命中块清零。采用此种方法描述访问间隔并通过访问间隔来预测访问概率。

这种替换策略在命中率方面表现很好，缺点则是每一次访问 Cache 时需要较大的计算量，时间复杂度较高。

3.6 Clock 替换策略

Clock 算法具体的实现方式为定义环形页表，指针指向初始位置，当命中时，将对应页表中的访问位定义为真。当缺失时，从指针位置开始进行判断，如果当前位置访问位为真，将访问位置为假，指针指向下一位，直到指针指向位置的访问位为假，则当前位置为要进行替换的位置，并将指针指向下一位。

这种替换策略综合考虑了 FIFO 替换策略和 LRU 替换策略，先进先出替换策略完全不考虑过去的做法，而 LRU 考虑的时间较长，时钟替换策略既考虑过去，考虑的时间又不是那么长。时钟替换策略的思路是对页面的访问情况进行大致统计。

4 设计替换策略

4.1 替换策略核心思想

访问 Cache 的目的是为了缩短获取数据的时间，这就意味着所有的替换策略都需要在提高命中率与减少访问计算量之间进行衡量取舍。

我设计的 Swing 替换策略如下：

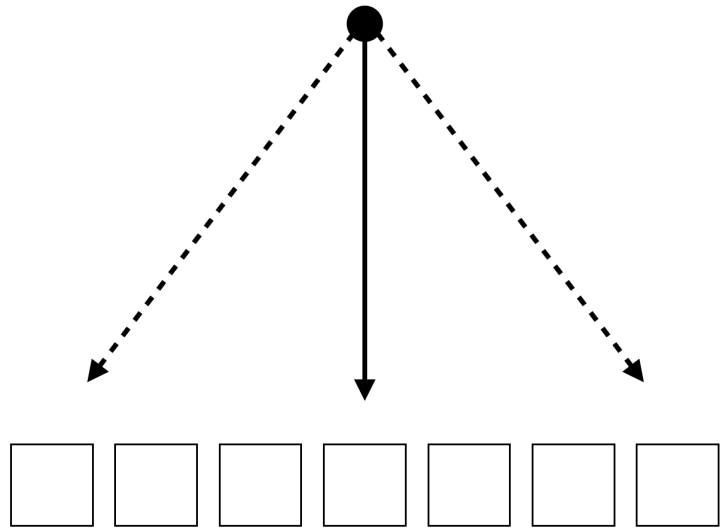


图 1: Swing 替换策略示意图

图中的矩形代表着 Cache 块，Cache 块上方为“淘汰摆”，其像重力摆一样做简谐运动，“淘汰摆”当前所指向的位置是下一个待替换的 Cache 块，当访问 Cache 未命中时，“淘汰摆”所指向的 Cache 块被替换，同时“淘汰摆”指向下一个 Cache 块。

当 Cache 初始化时，“淘汰摆”指向 Index 为 0 的 Cache 块，向此 Cache 块装入内容后，“淘汰摆”指向 Index 为 1 的 Cache 块，类似地不断进行下去。

假设 Cache 共有 n 个，当“淘汰摆”指向 Index 为 n 的 Cache 块时，向此 Cache 块装入内容后，“淘汰摆”指向 Index 为 n-1 的 Cache 块，由此之后，“淘汰摆”向相反方向摆动。

类似地，当“淘汰摆”再次指向 Index 为 0 的 Cache 块时，向此 Cache 块装入内容后，“淘汰摆”指向 Index 为 1 的 Cache 块，像重力摆一样做简谐运动。

由于在每次访问 Cache 的时候没有任何额外计算量，该替换策略可以保持极快的访问速度。命中率方面的表现将在下一小节进行具体分析。

- **时间复杂度** $O(1)$（每次访问 Cache 时无额外计算量。）
- **空间复杂度** $O(N)$（N 为 Cache 组相连路数，用于记录“淘汰摆”指向的位置。）

4.2 适用场景分析以及命中率分析

在一般的文件系统中，对文件的访问方式有以下几种:

- **顺序访问：**ABCDEFGHIJKLMNO...
- **循环访问：**ABCDEABCDEABCDE...
- **聚簇访问：**ABCDEDCBABCDEDC...
- **概率访问与随机访问**

对于顺序访问、概率访问、随机访问这三种形式，Cache 替换策略的设计对综合访问速度的影响不大，因此 Cache 替换策略设计的越简单越好，以减少每次访问时的计算量。另外，对这些情况来说，增加 Cache 的容量是提升综合访问速度较为有效的方法。

接下来对 Swing 替换策略在循环访问、聚簇访问和随机访问三种访问方式进行命中率分析：

4.2.1 循环访问方式测试

Swing 替换策略时间复杂度与 FIFO 替换策略的时间复杂度相同，再此将 Swing 替换策略和 FIFO 替换策略进行比较。

下面进行循环访问序列的理论分析。

不难发现，当循环序列长度小于 Cache 块个数时，所有替换策略均可以保证在循环序列加载完毕后的访问一直命中，因此不分析这种情况。

我们假设 Cache 块共有 4 个，循环序列长度为 5。模拟的访问过程如下图所示，为了节省空间，最初的装填过程未在图中表示出，从 Cache 装满开始的访问过程如下图表格所表示。

访问请求		E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
Cache	0	A	E	E	E	E	D	D	D	D	C	C	C	C	B	B	B	B	A	A	A	A	E	E	E	E	D	D	D	D	C	C	C	C	B	B	B	B	A	A	A	A	E
	1	B	B	A	A	A	A	E	E	E	E	D	D	D	D	C	C	C	C	B	B	B	B	A	A	A	A	E	E	E	E	D	D	D	D	C	C	C	C	B	B	B	B
	2	C	C	C	B	B	B	B	A	A	A	A	E	E	E	E	D	D	D	D	C	C	C	C	B	B	B	B	A	A	A	A	E	E	E	E	D	D	D	D	C	C	C
	3	D	D	D	D	C	C	C	C	B	B	B	B	A	A	A	A	E	E	E	E	D	D	D	D	C	C	C	C	B	B	B	B	A	A	A	A	E	E	E	E	D	D
FIFO选择项		0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0
缺页状态		E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E

访问请求		E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
Cache	0	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A	A	A	A	A	B	B	B	B
	1	B	B	B	B	B	C	C	C	C	C	C	C	C	A	A	A	A	A	A	A	A	A	A	A	A	A	A	C	C	C	C	B	B	B	B	B	B	C	C	C	C	C	C	C	C
	2	C	E	E	E	E	E	E	E	E	E	E	E	E	E	E	C	C	C	C	C	C	D	D	D	D	D	D	D	D	D	C	C	E	E	E	E	E	E	E	E	E	E	E
	3	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	E	E	E	E	E	E	E	E	E	E	E	E	E	E	D	D	D	D	D	D	D	D	D	D	D	D	D
"淘汰摆"指向		2	1	1	1	0	0	0	0	1	1	1	1	2	2	3	3	2	2	2	2	1	1	1	1	0	0	0	1	2	3	2	1	1	1	0	0	0	0	1	1	1
缺页状态		E			C				B				A		C		E				D				C			A	B	C	D	E			C				B			

图 2: FIFO 替换策略（上）和 Swing 替换策略（下）在访问循环序列的表现

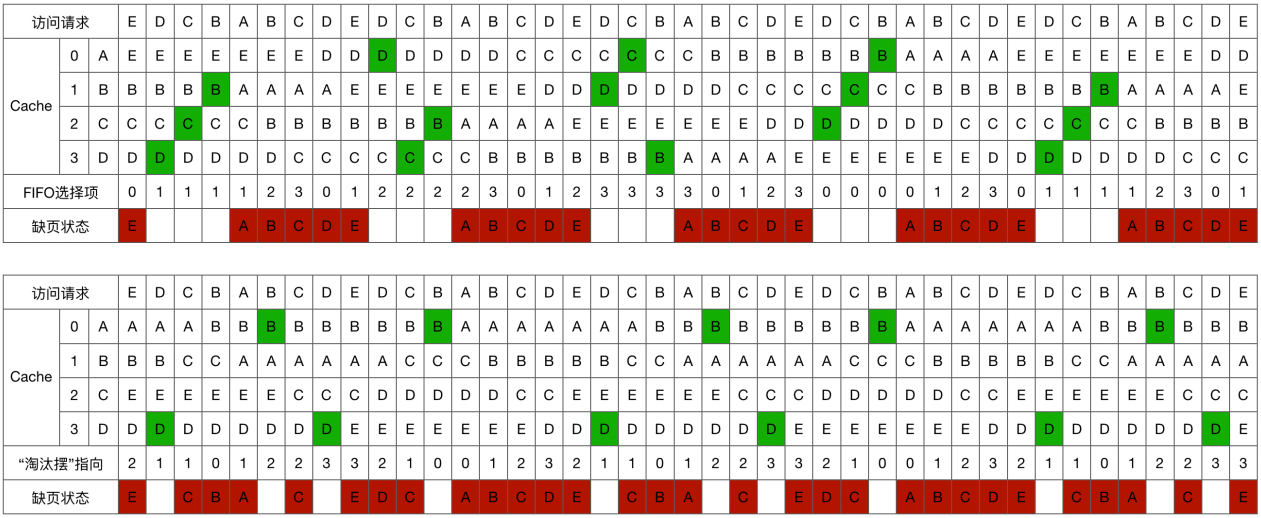
可以发现，FIFO 在进行循环序列的访问时有一个致命缺陷。当循环序列的长度大于 Cache 块的长度时，会导致访问 Cache 一直不命中。然而在 Swing 替换策略中，这个缺陷被修复了，而且在访问循环序列的过程中保持着很高的命中率 (63.4146%)。同时可以发现，使用 Swing 替换策略访问循环序列时，访问过程是也是有循环规律的，这就保证了命中率不会有大幅波动，在不断循环中命中率会趋向一个稳定的数值。

设 Cache 块个数为 n，循环序列长度 m，需要注意的是，当 $m > 1.5n$ 时，Swing 所表现出来的优势也消失了。

4.2.2 聚簇访问方式测试

同样使用 Swing 替换策略与 FIFO 替换策略进行比较。不难发现，当聚簇序列长度小于 Cache 块个数时，所有替换策略均可以保证在循环序列加载完毕后的访问一直命中，因此不分析这种情况。

我们假设 Cache 块共有 4 个，聚簇序列长度为 5。模拟的访问过程如下图所示，为了节省空间，最初的装填过程未在图中表示出，从 Cache 装满开始的访问过程如下图表格所表示。



访问请求		E	D	C	B	A	B	C	D	E	D	C	B	A	B	C	D	E	D	C	B	A	B	C	D	E	D	C	B	A	B	C	D	E	D	C	B	A	B	C	D	E
Cache	0	A	E	E	E	E	E	E	E	D	D	D	D	D	D	D	D	C	C	C	C	C	C	B	B	B	B	B	B	B	A	A	A	A	E	E	E	E	E	E	E	D	D
	1	B	B	B	B	B	A	A	A	A	E	E	E	E	E	E	E	D	D	D	D	D	D	D	C	C	C	C	C	C	B	B	B	B	B	B	B	A	A	A	A	E
	2	C	C	C	C	C	C	B	B	B	B	B	B	B	A	A	A	A	E	E	E	E	E	E	E	D	D	D	D	D	D	C	C	C	C	C	C	C	B	B	B	B
	3	D	D	D	D	D	D	D	C	C	C	C	C	C	C	B	B	B	B	B	B	B	A	A	A	A	E	E	E	E	E	E	E	D	D	D	D	D	D	D	C	C	C
FIFO选择项		0	1	1	1	1	2	3	0	1	2	2	2	2	3	0	1	2	3	3	3	3	0	1	2	3	0	0	0	0	1	2	3	0	1	1	1	1	2	3	0	1
缺页状态		E				A	B	C	D	E				A	B	C	D	E				A	B	C	D	E				A	B	C	D	E				A	B	C	D	E

访问请求		E	D	C	B	A	B	C	D	E	D	C	B	A	B	C	D	E	D	C	B	A	B	C	D	E	D	C	B	A	B	C	D	E	D	C	B	A	B	C	D	E
Cache	0	A	A	A	A	B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	B	B	B	B	B	B
	1	B	B	B	C	C	A	A	A	A	A	A	C	C	C	B	B	B	B	B	C	C	A	A	A	A	A	A	C	C	C	B	B	B	B	B	C	C	A	A	A	A	A
	2	C	E	E	E	E	E	E	C	C	C	D	D	D	D	D	C	C	E	E	E	E	E	E	C	C	C	D	D	D	D	D	C	C	E	E	E	E	E	E	C	C	C
	3	D	D	D	D	D	D	D	D	D	E	E	E	E	E	E	E	D	D	D	D	D	D	D	D	E	E	E	E	E	E	E	D	D	D	D	D	D	D	D	D	D	E
“淘汰摆”指向		2	1	1	0	1	2	2	3	3	2	1	0	0	1	2	3	2	1	1	0	1	2	2	3	3	2	1	0	0	1	2	3	2	1	1	0	1	2	2	3	3
缺页状态		E		C	B	A		C		E	D	C		A	B	C	D	E		C	B	A		C		E	D	C		A	B	C	D	E		C	B	A		C		E

图 3: FIFO 替换策略（上）和 Swing 替换策略（下）在访问聚簇序列的表现

可以发现，FIFO 替换策略在进行聚簇序列的访问时要好于 Swing 替换策略，这是因为 FIFO 替换出的是最先进入 Cache 的块，这序列的特点十分吻合，因此取得了不错的命中率 (36.5854%)。

Swing 替换策略的命中率略低，为 (26.8293%)，这是由于在 “淘汰摆” 摆动到两端的时候，经常会把刚刚进入的 Cache 块再次替换出去，导致 Swing 替换策略与聚簇序列的特点发生冲突，通过增大 Cache 块的数量可以有效缓解这一现象。

可以发现，Swing 替换策略在访问聚簇序列的时候，访问过程有循环规律，这就保证命中率不会大幅波动，在不断循环中命中率会趋向一个稳定的数值。

4.2.3 随机访问方式测试

我们假设 Cache 块共有 4 个，从操作系统课程的案例中取一段随机访问序列，随机序列中有 6 个可能出现的数值。模拟的访问过程如下图所示，为了节省空间，最初的装填过程未在图中表示出，从 Cache 装满开始的访问过程如下图表格所表示。

访问请求		E	B	D	C	F	A	B	C	D	F	B	C	D	E	C	D	E	A	B	C	D	E	A	B	C	D	E	F	C	D	C	D	E	F	A	C	B	D	E	F	D
Cache	0	A	E	E	E	E	E	E	E	C	C	C	C	C	C	C	C	C	C	A	A	A	A	E	E	E	E	D	D	D	D	D	D	D	D	D	D	D	C	C	C	C	F	F
	1	B	B	B	B	B	B	F	F	F	F	D	D	D	D	D	D	D	D	D	D	B	B	B	B	A	A	A	A	E	E	E	E	E	E	E	E	E	B	B	B	B	B	B
	2	C	C	C	C	C	C	C	A	A	A	A	F	F	F	F	F	F	F	F	F	F	C	C	C	C	B	B	B	B	F	F	F	F	F	F	F	F	F	D	D	D	D	D
	3	D	D	D	D	D	D	D	D	B	B	B	B	B	B	B	E	E	E	E	E	E	E	D	D	D	D	C	C	C	C	C	C	C	C	C	C	C	A	A	A	A	E	E	E
FIFO选择项		0	1	1	1	1	2	3	0	1	2	3	3	3	3	0	0	0	0	1	2	3	0	1	2	3	0	1	2	3	3	3	3	3	3	3	0	1	2	3	0	1
缺页状态		E				F	A	B	C	D	F				E				A	B	C	D	E	A	B	C	D	E	F							A	C	B	D	E	F		

访问请求		E	B	D	C	F	A	B	C	D	F	B	C	D	E	C	D	E	A	B	C	D	E	A	B	C	D	E	F	C	D	C	D	E	F	A	C	B	D	E	F	D
Cache	0	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	F	F	F	F	F	F	F	F	F	F	F	F	F	F
	1	B	B	B	B	B	B	F	F	F	F	F	F	F	F	E	E	E	E	A	A	A	A	A	A	A	A	E	E	E	E	E	E	E	E	E	A	A	A	A	A	A	A
	2	C	E	E	E	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	E	E	E	E	D	D	D	D	D	D	D	D	D	D	D	D	B	B	E	E	E
	3	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	C	C	C	C	C	C	C	C	C	C	C	C	C	D	D	D	D
“淘汰摆”指向		2	2	2	2	1	0	0	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	2	1	0	1	1	1	1	1	1	2	2	3	2	1	1
缺页状态		E			C	F		B						E				A				E			C	D	E	F							A		B	D	E		

图 4: FIFO 替换策略（上）和 Swing 替换策略（下）在访问随机序列的表现

可见 Swing 替换策略的命中率 (63.4146%) 高于 FIFO 替换策略的命中率 (39.0244%)。

4.2.4 Swing 替换策略优势分析

- Swing 替换策略时间复杂度为 $O(1)$。
- Swing 替换策略可以很好的应对循环序列访问。
- Swing 的“淘汰摆”在摆动到 Cache 块队列端点时有“长短时自适应”性质。

长短时自适应性质：

当“淘汰摆”即将摆动到 Cache 块队列端点时，如图所示。

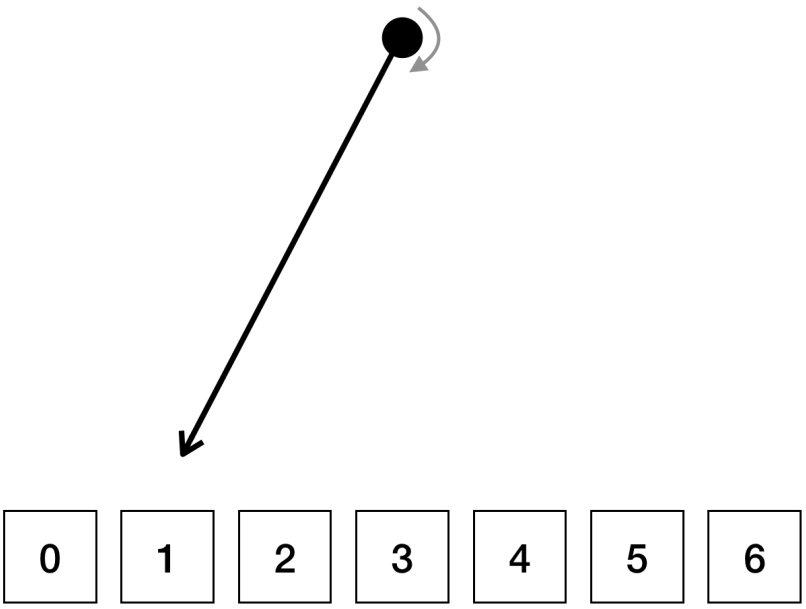


图 5: “淘汰摆”当前指向 Index 为 1 的 Cache 块并处于向左摆动的过程中

如果新装入的 Cache 块属于偶尔使用 1 次就不再使用的块，“淘汰摆”在摆动到左端点后会继续向右摆动，从而能够很快的替换掉这一个 Cache 块。

如果新装入的 Cache 块属于接下来会很频繁使用的块，“淘汰摆”在装入这一个 Cache 块后，虽然有可能在后续的向右摆动中替换掉这一块，但是因为这个 Cache 块会频繁使用，因此会很快在被装入，当再次装入这一块时，“淘汰摆”会继续向右摆动，摆动到右端点才会回来，这会让这一个 Cache 块保留很长时间。

4.3 实际测试与结果分析

5 实验总结

本次实验中，我深入学习了主流替换策略的核心算法，并在模拟器上实现了自己的替换策略。实验证明，该策略相比 LRU 来说缺失率相近，相比随机算法来说缺失率降低，在某些数据上表现比较好。通过本次实验，我对 Cache 的工作方式理解更加深刻了。