多层级预取和替换联合优化策略

郭裕彬[1)]

[1)]计网络空间安全学院,物联网工程专业,2114052

摘　要　预取与替换是影响 cache 性能的两个重要因素。本文测试了多种经典预取和替换策略组合后的性能表现，择出了表现较优的组合。本文尝试考虑预取与替换策略之间的关联性，提出了一种从 LLC-SRRIP 替换策略中改进的联合算法，简单与 LLC-next_line 预取策略结合后，在多种组合下有 2%-10%的性能提升。

关键词　预取策略；替换策略；cache；SRRIP；next line；

注：本实验的实验设计、联合算法设计部分与计算机学院计算机科学与技术专业的刘荟文(2114019)协作完成。

1 研究动机

Cache 在计算机体系结构中有着举足轻重的作用，学术界对于 Cache 的研究一直是热点。影响 Cache 性能的因素有很多，其中预取（prefetching）和替换（replacement）算法是最重要的两个因素。尽管研究 Cache 预取和替换算法的工作有很多，但是已有工作存在两个问题：（1）仅考虑 Cache 预取或者仅考虑 Cache 替换，没有考虑两者之间的关联性；（2）通常只考虑某个层级（例如，L2 或者 L3）的预取或者替换策略，没有考虑多个层级预取和替换策略之间的关联关系。本实验针对上述问题，研究多层级预取和替换联合优化策略，目标是最大化应用程序的性能。

2 相关工作

1）本实验使用的 champsim 模拟器环境提供了在 L1D、L2C 和 LLC 多级 cache 上的 next_line prefetcher、IP_stride(Instruction Pointer-based/Program counter-based Stride) prefetcher 等预取方法与 SRRIP(Static Re-Reference Interval Predictor) 和 SHiP(Signature-based Hit Predictor)这两种 LLC 替换策略，实验测试组合后的性能表现，寻找到比较优的算法组合。

2）考虑不同 cache 层级之间的预取策略以及 LLC cache 的预取和替换策略之间的关联性，在已有预取和替换策略算法的基础上进行修改，设计改进该方面指标的联合优化算法，进一步提高组合的性能。

3 方法描述

3.1 实验环境

实验测试环境运行在 Ubuntu22.04 64 位系统下，使用的 champsim 模拟器版来自 github 仓库（https://github.com/XDUFanYang/ChampSim），测试使用的 traces 为 462.libquantum-714B.champsimtrace.xz 、 482.sphinx3-1100B.champsimtrace.xz，预热指令数和模拟运行指令数都设置为 100M，评估指标为所有指令执行完后的 Cumulative IPC(instructions per cycle)数值。

3.2 预取算法

3.2.1 next_line

该预取算法设计简单，在每次产生数据需求后预取该数据其后的一个数据块的内容到 cache 中，该方法在顺序访存时很有效率，但非顺序甚至不规律的访存时会造成不可忽视的带宽浪费。

3.2.2 IP_stride

该预取算法在每次访存时记录与上一次该指令指针访问的地址之差，与同样作记录的上一次 stride 作比较，若相同则向前或向后预取同样 stride 的一个数据块。这种预取方法在遇见有跨度的规律性访存，如矩阵计算时具有较强的优势。

3.2.3 SPP

Signature Path Prefetcher（SPP）预取算法使用签名路径思想，维护一个记录页的签名表、一个记录特殊页访问模式的模式表和一个可能跨页访问的全局寄存器历史表，只有在预取周期结束时才开始预取下一个顺序地址。

3.3 替换策略

3.3.1 SRRIP

RRPV 值表示预测某一 cache 块被重新调用的次序，RRIP 链的前端（即 RRPV 值越小）表明该 cache 块很可能被再次调用，在 RRIP 链的后端则表示该 cache 块的下次调用间隔将会很长。SSRIP（静态再参考间隔预测策略）为了阻止再次调用间隔很久的 cache 块在 cache 长期存在、污染 cache，在每个 cache 块中记录该块的 RRPV 值。初始化时设置所有块的 RRPV 为 maxRRPV-1，替换时选择第一个 RRPV 值为 maxRRPV-1 的内存块进行替换，并将其值修改为 0；没有找到这个内存块则将所有内存块的 RRPV 值+1，再重复寻找。SRRIP 策略能够自动适应工作集中应用程序的大小，并事实性地降低 cache 缺失率。

3.3.2 SHiP

SHiP 是一种基于签名学习的缓存替换策略，结合了采样器（Sampler）和 SHCT（Sampler History Counter Table）来学习签名的重引用行为，进行缓存替换决策。命中时 SHiP 算法会增加 SHCT 表中该缓存行对应的签名的值。当一个缓存行要被替换出去并且在插入之后没有被重引用过，SHiP 会较少 SHCT 中对应的值。SHCT 表中的值代表着签名的重引用行为。如果值为 0，则说明这个缓存行很有可能不会被使用；如果 SHCT 中计数器的值是正的，说

表 1 实验数据

L1-pref	L2-pref	L3-pref	L3-repl	IPC-462	IPC-482	AVG-IPC
No	No	No	SHiP	0.50871	0.69395	0.60133
Next_line	No	No	SHiP	0.55487	1.05812	0.80650
No	Next_line	No	SHiP	0.5477	1.03398	0.79084
No	No	Next_line	SHiP	0.51425	0.85456	0.68440
No	IP_stride	No	SHiP	0.70430	1.11885	0.91158
No	No	IP_stride	SHiP	0.66252	0.87919	0.77086
No	SPP	No	SHiP	0.82640	1.03660	0.93150
No	IP_stride	Next_line	SHiP	0.72366	1.12639	0.92502
No	SPP	Next_line	SHiP	0.83287	1.09537	0.96412
Next_line	IP_stride	Next_line	SHiP	0.68953	1.22561	0.95757
Next_line	Next_line	Next_line	SHiP	0.64434	1.1605	0.90242
Next_line	SPP	Next_line	SHiP	0.64740	1.23325	0.94032
No	IP_stride	No	SRRIP	0.69243	1.02737	0.85990
No	No	IP_stride	SRRIP	0.65005	0.89159	0.77082
No	IP_stride	Next_line	SRRIP	0.71034	1.12937	0.91986
No	SPP	Next_line	SRRIP	0.84242	1.09692	0.96967
Next_line	IP_stride	Next_line	SRRIP	0.67658	1.23575	0.95616
Next_line	Next_line	Next_line	SRRIP	0.63579	1.17974	0.90776
Next_line	SPP	Next_line	SRRIP	0.64839	1.23740	0.94290

明相应的签名很有可能被命中。SHiP 算法可以为对于每一个插入的缓存行给出一个重引用间隔。在算法执行过程中，如果发生 cache 缺失，通过要插入的缓存行的签名在 SHCP 表中找到相应的值，如果这个值为零则表示该要插入的缓存行的重引用间隔很大，否则就认为重引用间隔较小，将会被访问。利用这些信息，替换策略可以选择是否要替换该行。相关研究表明，在 LLC 中使用 SHiP 策略对缓存的性能有很大的提升。

4 实验结果

实验结果如表 1 所示，可以看出，在两个数据集上表现均较好的算法组合有：

1） L1D Prefetcher:no
   L2C Prefetcher:SPP
   LLC Prefetcher:Next_line
   LLC Replacement:SHiP
2） L1D Prefetcher:Next_line
   L2C Prefetcher:IP_stride
   LLC Prefetcher:Next_line
   LLC Replacement:SHiP
3） L1D Prefetcher:no
   L2C Prefetcher:SPP
   LLC Prefetcher:Next_line
   LLC Replacement:SRRIP
4） L1D Prefetcher:Next_line
   L2C Prefetcher:IP_stride
   LLC Prefetcher:Next_line
   LLC Replacement:SRRIP

分析实验结果，单一的预取算法与替换策略相组合时都不能很好地发挥缓存的性能，当应用到多级 cache，甚至在不同 cache 上使用不同的预取策略时，缓存性能能够有 20%至 30%的显著提升。从结果上来看，使用 SRRIP 还是 SHiP 对于相同预取策略的组合来说结果是独立的，或者说不论是哪种替换策略，其对整个测试的影响都来自于它本身，与预取策略的选择没有关系。这也符合本实验所讨论的“预取策略和替换策略之间没有考虑关联性”的情景。

5 联合算法设计

5.1 设计思路

考虑 L3 层级的 Next_line 预取策略和 SRRIP 替换策略，预取器总是预取正在使用的数据块的下一个内存块，这会在运行时产生相当一部分将来永远不会用到的无效块缓存在 cache 中，如果能在替换时优先考虑这些“预取进入 cache 而从未被使用”的数据块进行换出，在整体上就有更大的概率使得原本要被换出但可能在之后会被使用的非预取原因进入 cache 的内存块在之后被引用。结合 SRRIP 的算法的 RRPV 值，即可修改出与预取策略相关联的 new_SRRIP 替换策略：

1） 定义一个 LLC_pref 和 LLC_repl 共享的数组 my_priority 记录 cache 中每一个数据块的预取-优先级，初始化全为 0。
2） 每次预取成功后，将 cache 放入位置对应 my_priority 数组位置的值记录为 1。
3） 每次 hit 命中后，将命中的 cache 块位置对应 my_priority 数组位置的值改为 0。
4） SRRIP 在遍历 cache 中 RRPV 值时，选择 RRPV 值为 maxRRPV-1 的候选块中选择 my_priority 数组对应位置为 1 的块换出；当不存在这样一个数据块时，再直接从候选块中选第一个换出。

这样一个策略设计的假设是，当一个预取的内存块在接下来的数次替换（也即更多次的内存访问）中都没有 HIT，就可以认为这个内存块的预取是失败的，在相当长的一段 时间之内都不会再访问到。由于 cache 大小具有不变性，因此换出时可以优先考虑这些数据块而不是相同换出优先级（RRPV 值）但是曾经向前一段时间内被访问、之后也有可能被访问的数据块。

表 2 改进测试数据

L1-pref	L2-pref	L3-pref	L3-repl	IPC-462	IPC-482	IPC-AVG
No	No	Next_line	SHiP	0.51425	0.85456	0.684405
No	No	Next_line	SRRIP	0.50901	0.87178	0.690395
No	No	Next_line	New_SRRIP	0.52354	0.93161	0.727575
No	IP_stride	Next_line	SHiP	0.72366	1.12639	0.925025
No	IP_stride	Next_line	SRRIP	0.71034	1.12937	0.919855
No	IP_stride	Next_line	New_SRRIP	0.74626	1.15282	0.94954
No	SPP	Next_line	SHiP	0.83287	1.09537	0.96412
No	SPP	Next_line	SRRIP	0.84242	1.09692	0.96967
No	SPP	Next_line	New_SRRIP	0.85864	1.13125	0.99494
Next_line	IP_stride	Next_line	SHiP	0.68953	1.22561	0.95757
Next_line	IP_stride	Next_line	SRRIP	0.67658	1.23575	0.956165
Next_line	IP_stride	Next_line	New_SRRIP	0.71808	1.25068	0.98438
Next_line	SPP	Next_line	SHiP	0.64740	1.23325	0.94032
Next_line	SPP	Next_line	SRRIP	0.64839	1.23740	0.94290
Next_line	SPP	Next_line	New_SRRIP	0.67735	1.27039	0.97387

5.2 测试结果

从表 2 可以看出，考虑了预取策略关联关系的 New_SRRIP 在相同的预取策略组合下，相比于其他替换策略在不同 traces 下和不同组合下 Cumulative IPC 性能有 2%-10%的提升，但平均提高率仅有 4%左右，这与算法设计简单有关，但相对应地，改进后的算法对硬件开销的增量不多，几乎可以忽略不计。