文章编号：2095-2163(2021)06-0020-06 中图分类号：TP39 文献标志码：A

基于分类的预取感知缓存分区机制

陈玲玲[1]，焦 童[1]，汪 玲[2]，安 鑫[1]，李建华[1]
（1 合肥工业大学 计算机与信息学院，合肥 230009；2 安徽交通职业技术学院，合肥 230051）

摘 要：在多核处理器中，硬件预取技术是解决存储墙问题的主要技术之一，是对高速缓冲寄存器的优化。但是现有的预取技术大多只考虑内存密集型程序的性能优化，而忽视了非内存密集型程序因预取而受到的干扰。针对这个问题，本文提出基于分类的预取感知缓存分区机制，利用自适应预取控制和缓存分区技术，可以动态调整预取的激进程度和合理分配共享缓存，该机制使用 Champsim 进行仿真实验。实验结果表明该机制可以有效提高非内存密集型程序的吞吐量，减少核间干扰，提高系统的性能和公平性。

关键词：高速缓冲寄存器；多核处理器；共享缓存；硬件预取；缓存分区

Classification-based Prefetch-aware cache partition mechanism

CHEN Lingling[1], JIAO Tong[1], WANG Ling[2], AN Xin[1], LI Jianhua[1]
(1 School of Computer and Information, Hefei University of Technology, Hefei 230009, China;
2 Anhui Vocational and Technical College, Hefei 230051, China)

【Abstract】 In multi-core processor, hardware prefetching is one of the main technologies to solve the problem of memory wall, which is the optimization of cache registers. However, most of the existing prefetching techniques only consider the performance optimization of memory intensive programs, and ignore the interference of non memory intensive programs due to prefetching. To solve this problem, this paper proposes a Classification-based Prefetch-aware Cache Partition Mechanism. Using adaptive prefetcher control and cache partitioning technology, we can dynamically adjust the aggressiveness of prefetcher and reasonably allocate shared cache. The mechanism uses champsim for simulation experiments. Experimental results show that this mechanism can effectively improve the throughput of non memory intensive programs, reduce inter core interference, and improve the performance and fairness of the system.

【Key words】 cache memory; multicore processor; shared cache; hardware prefetching; cache partitioning

0 引 言

多核处理器(Chip Multiprocessor，简称为 CMP)是当前通用处理器的主流架构，该架构下多个核心共享最后一级缓存(Last-Level Cache，简称 LLC)和片外带宽等共享资源。多个核心共享 LLC 会导致访存干扰，从而影响系统性能和融合多核处理器的服务质量。具体来说，当不同的应用程序运行在多核处理器的不同核心上时，会同时发出内存请求，彼此相互冲突，增加内存访问延迟，使得每个应用程序比单独运行时更慢。

硬件预取是减少内存访问延迟的主流技术之一。通过学习当前程序的访存模式来预测程序之后一段时间内的访存行为，并在程序发出访存请求之前将数据预取进高速缓冲寄存器，来隐藏内存访问延迟[1]。但预取也可能会导致缓存层次结构污染，并在内存系统中产生过多的流量和争用，增加核间干扰。因此，配备预取机制的多核处理器，减少核间干扰是提升系统性能的主要瓶颈之一。

先前的大量工作提出了基于预取来减少核间干扰的一些方案。但是，大多数预取方案都没有考虑过非内存密集型程序的特殊性。由于其在运行期间发出的访存请求远低于其它程序，若不对其优先处理，就容易造成以下问题：

(1)发出的访存指令少导致预取器难以找到程序的访存规律，若非内存密集程序盲目的配置，过于激进的预取，由于其局部性不强，容易将其它潜在有用的数据踢出缓存，造成严重的缓存污染，从而影响

基金项目：国家自然科学基金青年基金(61402145，61673156)。

作者简介：陈玲玲(1996-)，女，硕士研究生，主要研究方向：缓存系统、硬件预取技术；焦 童(1995-)，男，硕士研究生，主要研究方向：高性能存储器；汪 玲(1997-)，女，硕士研究生，主要研究方向：片上系统；安 鑫(1986-)，男，博士，副教授，主要研究方向：嵌入式系统；李建华(1985-)，男，博士，副研究员，主要研究方向：计算机体系结构。

通讯作者：李建华 Email：jhli@hfut.edu.cn

收稿日期：2021-03-18

其它应用程序的性能。

(2)如果简单地将 LLC 平均分给各个应用程序,非内存密集型程序很容易被内存密集型程序发出的频繁访存请求挤出缓存,增加非内存密集型程序的缺失率。

因此本文提出了基于分类的预取感知缓存分区机制(简称 CPAP)。该机制基于非内存密集型程序的特点,进一步研究缓存分区和预取控制的方式。

1　相关工作

硬件预取是提升处理器性能的核心技术之一,之前已经有很多关于提升预取性能的研究[2]。下面将介绍与本文研究内容相关的一些预取的技术。

1.1　缓存分区

缓存分区可以将受预取干扰严重的程序单独放入一个分区中,避免数据块被挤出 LLC,导致内存访问延迟增大。Selfa 等提出了一种基于集群的缓存分区机制,以提高多核处理器的公平性,该机制根据二级缓存停顿周期数将核心上的应用归类为集群,为不同的应用分配不同的缓存路数[3];Qureshi 等提出了一种基于程序实用性的缓存分区机制(简称 UCP),在多个应用程序之间划分共享的 LLC,UCP 根据应用程序利用缓存空间的实用性高低为其分配合适数量的缓存路数,以提升缓存空间的使用效率[4];Sun 等提出了协调控制预取和缓存分区技术(简称 CMM)来提高多核系统性能,CMM 根据预取是否激进将其分为 2 类,为预取激进的程序分配较小数量的缓存路数,其它程序共享剩余缓存路数[5]。

1.2　全局预取控制

Ebrahimi 等提出了一种分层控制预取激进程度的方法来动态控制多核处理器中的硬件预取器,该方法可动态地识别出引起核心间干扰的应用程序,并限制其预取的激进程度,以减少对其它应用的干扰[6];Panda 等提出了一种协调控制预取器激进程度的机制来提高多核系统的公平性,该机制探索了预取器的控制决策之间的相互作用,并基于多核系统的公平性来控制预取器,协调了预取器与整体公平性的关系[7]。

1.3　其它预取相关的研究

除了控制预取的激进程度之外,如何提升预取的精度同样重要。本文并未提出新的预取技术,主要是基于步幅预取器的优化,可以将本文提出的机制运用到其它经硬件预取器上,比如反馈指导预取[8],沙箱预取[9]等等。Selfa 等[10]提出了激活/停用预提取器来正确处理内存带宽,从而改善性能。Navarro 等[11]提出了带宽感知的预取配置(简称 BAPC)来提高多程序工作负载的性能。Seshadri 等[12]提出了基于信息的预取块替换策略。该机制只将预测准确的预取块以高优先级方式插入到缓存中,从而缓解缓存污染。Huang 等[13]提出了通过控制预取距离来减少预取污染,该方法估算特定应用的预取距离上限,然后分析增加预取距离对共享缓存污染的影响,以此来减少共享的缓存污染。Lee 等[14]提出了预取感知 DRAM 控制器来动态调整预取请求优先级技术,该方法预取请求的有用性,并根据估算值动态调整其调度和缓冲区管理策略,实现最大程度地提高有用的预取的性能。Ebrahimi 等[15]提出了预取感知共享资源管理器,该方法既可以利用预取的优势,又可以管理多核芯片的共享资源以获得高性能和公平性。Wu 等[16]提出了预取感知缓存管理,该方法通过修改缓存插入策略和命中率提升策略来以不同方式对待需求和预取请求,消除了预取对替换策略的影响。

2　基于分类的预取感知缓存分区机制

2.1　CPAP 整体架构

CPAP 的目标是减少非内存密集型程序因预取引起的核间干扰而性能降低。CPAP 先识别非内存密集型程序,通过预取控制和缓存分区来限制其资源使用,以减少核间干扰,从而提高系统性能和公平性。CPAP 被设计成一种前端和后端分离的结构,如图 1 所示。前端负责检测,主要检测 2 类程序:非内存密集型程序和预取友好程序;后端负责控制,根据前端检测分类的结果,进行相应的预取配置和缓存分区方案。

CPAP 将应用程序的执行过程分为多个间隔(Interval),在每个执行周期结束时,该机制的前端会收集运行时的统计信息并检测当前程序是否为非内存密集型程序,判断预取的准确性的高低。随后,后端会通过该机制的类别做出相应的预取控制,下一个执行时期就会使用该预取配置进行操作。

2.2　前端:收集数据并检测

2.2.1　检测预取友好型程序

预取友好程序是指程序通过预取获得了很好的性能提升。根据这个特性,本文使用预取的激进程度(Aggressiveness,简称 Agg)和每周期执行指令数加速比(IPC Speedup,简称 IS)这一组合指标来判断预取是否友好。其中,预取的激进程度表示预取是否开启,IS 代表性能是否提升。

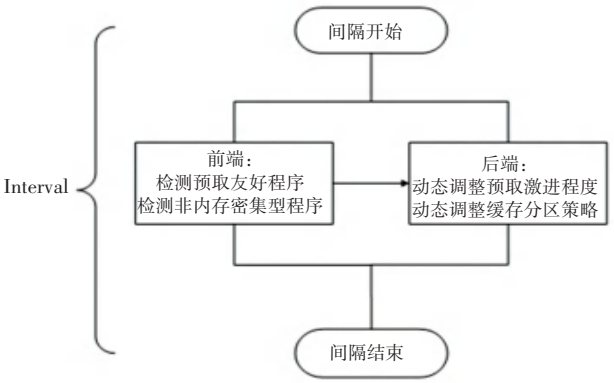

图 1 CPAP 整体架构

Fig. 1 CPAP architecture

在文中所提出的方案中,预取的激进程度主要取决于预取的准确性(Accuracy,简称 Acc),准确性计算主要使用 2 个计数器:

(1)预取计数器(pref-total):用于记录预取器发出的预取请求次数;

(2)有用预取计数器(use-total):记录预取的缓存块被 CPU 命中的次数。预取准确性的计算如公式(1)所示。

$$Prefetch\ Accuracy = \frac{use\ total}{pref\ total}, \tag{1}$$

IS 主要用来判断性能是否提升,如公式(2)所示,IPC_{current} 是当前间隔中获得的 IPC,IPC_{previous} 是在上一个间隔中获得的 IPC。

$$IPC\ Speedup = \frac{IPC\text{current}}{IPC\text{previous}}. \tag{2}$$

判断程序是否友好的具体流程,如图 2 所示。首先,判断预取是否打开(即预取的激进程度大于 0),若预取并未打开,则认为预取不友好;若预取打开,则判断当前预取是否带来性能提升,若能带来性能提升,则当前程序为预取友好。反之,当前程序预取不友好。

2.2.2 检测非内存密集型程序

非内存密集型程序是指该类程序在整个运行过程中,发出的访存指令明显少于其它程序。如果处理器的访存指令发生了缓存未命中时,必然会对下一级内存结构发出进一步的访问请求,所以可以依据每 1 000 条指令的缓存未命中数(Misses per 1K Instructions,简称 MPKI)这一指标用来计算该程序有多少访存指令需要访问下一级内存结构,从而判断该程序是否为非内存密集型程序。在每个间隔的结束,CPAP 判断当前核心上运行程序的 MPKI 是否小于预先设定的阈值(T_{MPKI}),若小于则将该程序设置为非内存密集型程序;反之,则设置为内存密集型程序。当前间隔检测的结果用于下一个间隔的运行控制和资源分配。

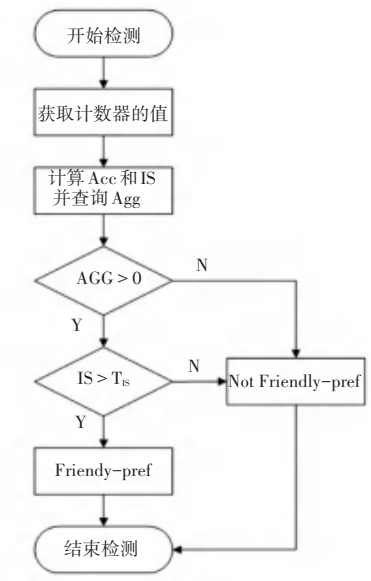

图 2 预取友好性程序检测流程

Fig. 2 Prefetch-friendly application detection process

2.3 后端:自适应调整机制

2.3.1 调整预取的激进程度

预取的激进程度是决定预取技术是否能带来性能提升的一个重要原因,预取的激进程度主要表现为预取距离(Prefetch Distance)和预取度(Prefetch Degree)2 方面。预取的激进程度越大,预取距离和预取度的值越大。表 1 给出了预取器的 4 种不同的预取激进程度配置,其中 OFF 代表关闭预取。

表 1 预取器激进程度

Tab. 1 Prefetcher aggressiveness

Case	Aggressiveness	Distance	Degree
1	OFF	-	-
2	Conservative	8	1
3	Middle	16	2
4	Aggressive	32	4

预取准确性是指预取器预测程序,即将访问的内存地址的准确程度的度量,对预取性能具有巨大影响,CPAP 主要通过预取的准确性来决定预取的激进程度。在间隔结束时,CPAP 会计算出预取的准确性,若准确性高于阈值,则在下一个间隔运行时提升预取的激进程度;反之,若低于阈值,则降低预取的激进程度。在本文中,步幅预取器的初始激进程度设为 2。

CPAP 还明确了何时打开预取,如何处理非内存密集型程序。

(1)若预取器处于关闭状态(OFF),但是性能却依旧处于下降趋势,这很可能是因为内存访问延迟过大而导致的性能下降,则应打开预取,避免性能进一步损失;

(2)非内存密集型程序发出的访存请求少,局部性不强,难以准确预测。所以非内存密集型程序的准确性阈值要高于其它程序,尽量减少非内存密集型程序长期配置过高的预取激进程度。同时,非内存密集型程序的最大预取激进程度不能超过“中等”配置。通过这 2 项限制条件,可以有效的控制非内存密集型程序的预取激进程度,使得其既可以获得预取带来的性能提升,又可以避免对其它应用程序的干扰,保证其它应用程序可以保持应有的性能。

2.3.2　调整缓存分区

非内存密集型程序发出的访存请求较少,对路数(way)的需求自然不高,较小的路数就可以保证性能。同时,预取友好的程序最多只需 2-way 就可以达到其最优性能的 90%,因此 CPAP 将非内存密集型程序放入一个较小的分区中,将预取友好型程序放入另外一个分区中,在保证性能的同时,避免相互干扰。

CPAP 的调整缓存分区根据前端收集的信息,首先,判断多核系统中是否有非内存密集型程序,其次,再考虑是否存在预取友好程序,最后,在图 3 中选择最合适的缓存分区方式,以此来减少其它程序对非内存密集型程序的影响。

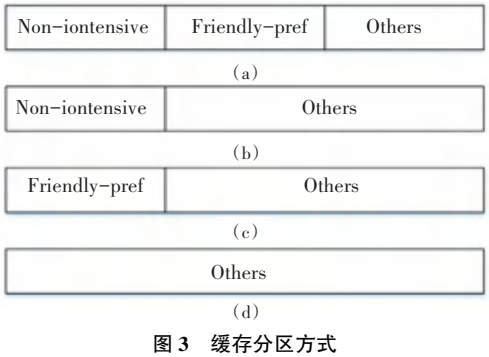

图 3　缓存分区方式

Fig. 3　Cache partitioning

3　实验及结果分析

3.1　实验方法

本文使用 ChampSim [19]仿真片上多核处理器系统对 CPAP 机制进行评估。使用的基准硬件预取器是步幅预取器,基准测试程序是 SPEC CPU 2017。表 2 给出了仿真的多核处理器系统的核心参数。因为本文所研究的问题主要关注片上共享资源分配的问题,为了避免 L1 和 L2 的预取请求对 LLC 预取策略产生干扰,本文的预取只发生在 LLC 与主存之间。

CPAP 方案的阈值设置,表 3 提供了用于实施 CPAP 机制的阈值。本文对所有的基准测试程序,基于 5 种不同的参数组合,分别运行了1 000次,就整体而言,表 3 的配置可以带来最佳的性能提升。其中, T_{nAcc} 表示的非内存密集型程序的准确性阈值;T_{Acc} 表示其它经程序的准确性阈值;T_{IS} 表示 *IS* 的阈值;T_{MPKI} 表示 *MPKI* 的阈值。

表 2　仿真参数

Tab. 2　Simulation parameters

Component	Configuration
Core	4 Cores, 4GHz
L1I	32 KB, 8-way, 3 cycles, 私有, LRU 替换策略
L1D	48KB, 12-way, 5 cycles,私有, LRU 替换策略
L2	512KB, 8-way, 10 cycles,私有, LRU 替换策略
LLC	2MB, 16-way, 20 cycles,共享, LRU 替换策略

表 3　CPAP 阈值

Tab. 3　CPAP threshold

T_{nAcc}	T_{Acc}	T_{IS}	T_{MPKI}
0.75	0.5	1.1	3

3.2　动态调整预取激进程度

首先,本文评估了 CPAP 机制的动态调整预取器激进程度的性能,与不包含动态反馈的 3 种传统配置进行比较:无预取,保守预取和激进预取。图 4 显示了 20 种基准测试程序在 4 种预取配置下的 IPC 性能变化。动态调整预取器的激进程度可在所有配置中提供最佳的平均性能。整体上看,使用动态地调整预取器的激进程度,与“激进”配置相比,平均 IPC 提高了 6.2%,与不预取相比,IPC 提高了 15%。几乎在所有基准测试中,动态调整预取激进程度的性能都非常接近每个基准测试性能最佳的传统预取器配置所实现的性能。因此,该动态机制能够基于每个基准检测,并为步幅预取器采用最佳性能的激进程度。

如图 4 所示,动态调整预取激进程度几乎完全消除了由于激进的预取而导致的非内存密集型测试程序性能大幅下降。在 3 种传统配置中,激进的预取器配置提供了整体最佳的平均性能,但对于 x264,deepsjeng 等非内存密集型程序而言,激进的预取会大大降低性能,与不预取相比,激进的预取使非内存密集型测试程序性能平均损失了 10%。因

为传统的激进预取忽视了非内存密集型程序发出的请求少,局部性不强的特性,盲目的使用激进的预取配置,导致内存结构污染,性能下降;相反,动态调整预取激进程度对非内存密集型的预取控制更加严格,使非内存密集型程序性能平均提高了 56%。

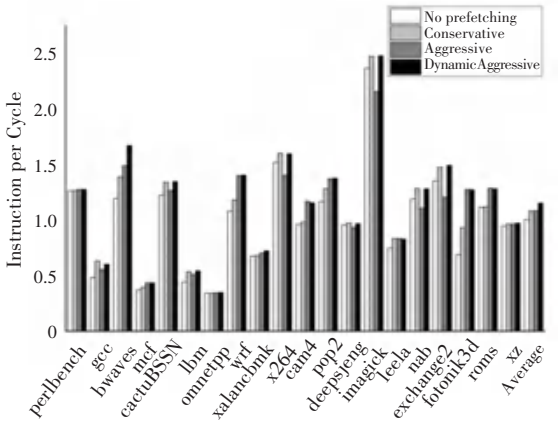

图 4 基于分类的自适应调整激进程度

Fig. 4 Adaptive adjustment of aggressiveness based on classification

3.3 动态调整缓存分区

为了更直观评估 CPAP 机制的动态调整缓存分区的性能,将 CPAP 的性能与 3 种静态分区策略进行比较:

(1)不预取且不分区;

(2)保守预取且基本分区(如图 3(b)所示);

(3)保守预取且不分区(如图 3(d)所示)。

图 5 显示了 20 种基准测试程序在 4 种缓存分区策略下 IPC 的性能变化。整体上看,与不预取且不分区相比,基本分区方式使 IPC 仅上升了 1%,不分区方式使 IPC 上升了 7%。基本分区方式虽然可以缓解非内存密集型程序受到其它经程序干扰,但整体的性能提升几乎可以忽略不计,这是因为当 CMP 中没有非内存密集型程序时,基本分区中 Non-intensive 部分不可使用,使得有效的 LLC 空间减少,导致 LLC 争用更加严重,cam4 和 roms 等内存密集型程序受到的影响最为严重。

CPAP 可以比任何静态分区策略提供更高的性能。整体上看,与基本分区相比,CPAP 实现了 16.5%的性能提高;与不分区相比,CPAP 的性能提高了 9.3%,CPAP 几乎为每个基准测试提供了最佳静态分区策略的性能。对于非内存密集型程序,CPAP 选择最优的分区策略(即基本分区),大大减少其它经程序对非内存密集型程序的干扰,使其 IPC 提高了 61%(与不预取相比)。因此,运行时通过对应用程序的分类来动态调整分区策略,可以选择出基于步幅预取器的性能最佳的分区方案。

3.4 合并:动态调整预取控制和缓存分区

本节研究了 CPAP 机制动态调整预取程序的激进程度和缓存分区的效果。图 6 显示 CPAP 与其 3 种机制的 IPC 性能比较,从左到右分别是:

(1)不预取;

(2)保守预取且基本分区;

(3)动态调整预取激进程度;

(4)动态调整预取激进程度和缓存分区(CPAP)。

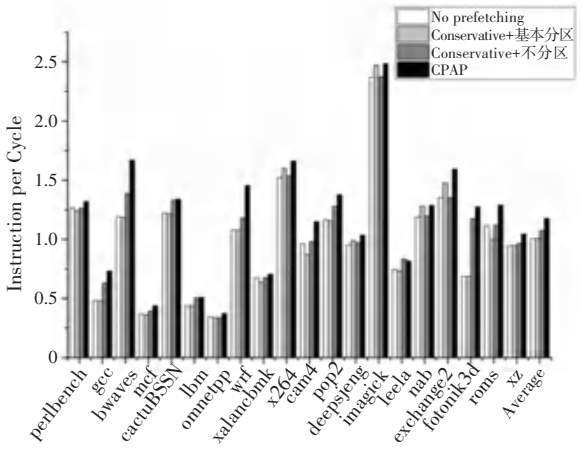

图 5 基于分类的自适应调整缓存分区

Fig. 5 Adaptive adjustment of cache partition based on classification

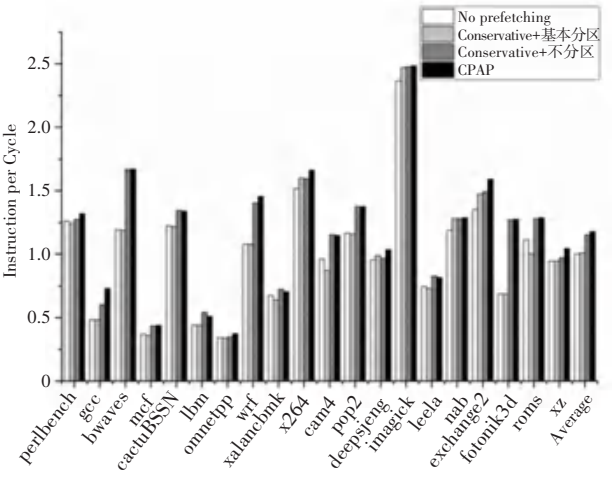

图 6 CPAP 机制的性能评估

Fig. 6 Performance evaluation of CPAP

整体上看,CPAP 可提供最佳性能,与不预取相比,可将 IPC 平均提高 17.8%,此性能提升要大于仅动态调整预取激进程度或最优的静态缓存分区策略所提供的性能提升。因此,动态调整预取器行为(预取激进程度和缓存分区策略)的 2 个方面都可以提供互补的性能优势。

对于一些内存密集型程序,如 omnetpp,对 LLC

有效空间大小变化不敏感，CPAP 可获得最佳性能提升。但对于大多数内存密集型程序，会因为 LLC 有效空间减少，导致性能明显下降，如 cam4，但 CPAP 可以通过动态调整分区策略，缓解这一问题，使性能得到明显提升，仅略低于动态调整预取激进程度，约 1%。

对于非内存密集型程序，CPAP 不仅减少了其它程序对其干扰，而且几乎完全消除了由于激进预取而导致的性能损失，如 exchange2，相比于动态调整激进程度，CPAP 提升了 6%。

4　结束语

在多核处理器中，硬件预取在掩藏内存延迟的同时，也会在内存系统中产生过多的流量和争用，增加核间干扰。其中，非内存密集型程序受到的来自预取的干扰十分严重。首先，由于其在运行期间发出内存请求指令少，局部性弱，直接给予过于激进的预取配置，会将潜在有用的数据挤出 LLC，增加缓存污染；其次，其它经预取友好的程序也会将非内存密集型程序挤出 LLC，增大其内存访问延迟。

为了既能保证预取所带来的的性能优势，又能消除预取给非内存密集型程序带来性能下降，本文提出了 CPAP 机制，一种基于分类的预取感知缓存分区机制，主要通过监视应用程序的预取/缓存行为，以动态和协调的方式管理硬件预取器和 LLC 分区方式。实验结果表明，使用缓存分区来隔离具有不同预取行为的应用程序，并结合预取限制，可以有效的减少预取器引起的核间干扰，以最大程度地提高系统性能。

参考文献

[1] LEE J, KIM H, VUDUC R. When Prefetching Works, When It Doesn't, and Why[J]. ACM Transactions on Architecture and Code Optimization. 2012, 9(1): 1-29.

[2] SMITH, JAY A. Cache Memories[J]. Association for Computing Machinery. 1982, 14(3): 473-530.

[3] SELFA V, SAHUQUILLO J, EECKHOUT L, et al. Application clustering policies to address system fairness with Intel's cache allocation technology[C]//Proceedings of the 26th International Conference on Parallel Architectures and Compilation Techniques. Portland, USA: PACT Press, 2017: 194-205.

[4] QURESHI K, PATT N. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches[C]//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Orlando, USA: IEEE Press, 2006: 423-432.

[5] SUN G, SHEN J, VEIDENBAUM V. Combining Prefetch Control and Cache Partitioning to Improve Multicore Performance[C]//Proceedings of 2019 IEEE International Parallel and Distributed Processing Symposium. Rio de Janeiro, Brazil: IEEE Press, 2019: 953-962.

[6] EBRAHIMI E, MUTLU O, LEE J, et al. Coordinated control of multiple prefetchers in multi-core systems[C]//Proceedings of the 42th Annual IEEE/ACM International Symposium on Microarchitecture. New York, USA: IEEE Press, 2009: 316-326.

[7] PANDA B. SPAC: A Synergistic Prefetcher Aggressiveness Controller for Multi-Core Systems[J]. IEEE Transactions on Computers, 2016, 65(12): 3740-3753.

[8] SRINATH S, MUTLU O, KIM H, et al. Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers[C]//Proceedings of IEEE 13th International Symposium on High Performance Computer Architecture. Scottsdale, AZ, USA: IEEE Press, 2007: 63-74.

[9] PUGSLEY S, CHISHTI Z, WILKERSON C, et al. Sandbox Prefetching: Safe run-time evaluation of aggressive prefetchers[C]//Proceedings of IEEE 20th International Symposium on High Performance Computer Architecture, Orlando, USA: IEEE Press, 2014: 626-637.

[10] SELFA V, AHUQUILLO J, GÓMEZ, et al. Efficient selective multicore prefetching under limited memory bandwidth[J]. Journal of Parallel and Distributed Computing. 2018, 120: 32-43.

[11] NAVARRO C, FELIU J, PETIT S, et al. Bandwidth-Aware Dynamic Prefetch Configuration for IBM POWER8[J]. IEEE Transactions on Parallel and Distributed Systems. 2020, 31(8): 1970-1982.

[12] SESHADRI V, YEDKAR S, XIN H, et al. Mitigating Prefetcher-Caused Pollution Using Informed Caching Policies for Prefetched Blocks[J]. ACM Transactions on Architecture and Code Optimization, 2015, 11(4): 1-22.

[13] HUANG Y, GU Z, TANG J, et al. Reducing Cache Pollution of Threaded Prefetching by Controlling Prefetch Distance[C]//Proceedings of IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum. Shanghai, China: IEEE Press, 2012: 1812-1819.

[14] LEE J, MUTLU O, NARASIMAN V, et al. Prefetch-Aware DRAM Controllers[C]//Proceedings of the 41th IEEE/ACM International Symposium on Microarchitecture. Lake Como, Italy: IEEE Press, 2008: 200-209.

[15] EBRAHIMI E, LEE J, MUTLU O, et al. Prefetch-aware shared-resource management for multi-core systems[J]. 2011 38th Annual International Symposium on Computer Architecture. 2011, 39(3): 141-152.

[16] WU C, JALEEL A, MARTONOSI M, et al. PACMan: prefetch-aware cache management for high performance caching[C]//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. New York, USA: IEEE Press, 2011: 442-453.