doi:10.3772/j.issn.1002-0470.2024.03.003

基于行内局部性的内存控制器端预取①

周叔欣②[*,**,***] 张见齐[****] 王焕东[****] 章隆兵[*,**,***]

([*]处理器芯片全国重点实验室(中国科学院计算技术研究所) 北京 100190)

([**]中国科学院计算技术研究所 北京 100190)

([***]中国科学院大学 北京 100049)

([****]龙芯中科技术股份有限公司 北京 100190)

摘 要 本文提出一种基于行内局部性的内存控制器端预取。采用位图的数据结构记录行内每个数据块的状态;并且对每一行进行区域划分,量化每个区域的访问局部性;根据区域内的局部性高低决定预取的激进程度。对于局部性较低的区域,预取区域内未被访问过的数据块;对于局部性较高的区域,同时采用跨区域的预取。通过动态调整区域规模的大小来适应局部性程度的变化。上述预取方法在龙芯 3A6000 处理器上实现并评测,评测程序采用 SPEC CPU2006 访存密集型应用。评测结果显示本文的预取方法将每周期指令数(IPC)平均提升 6.51%,将单线程 IPC 最高提升 46.80%(bwaves),将双核四线程 IPC 最高提升 26.22%(lbm)。

关键词 内存控制器; 预取; 局部性

“内存墙”[1]问题是由于处理器的性能提升速度明显高于主存,导致处理器与主存之间的性能差距越来越大。在多核处理器中,“内存墙”问题由于以下几个原因更加严重:(1)内存控制器内排队的请求增多,排队延迟增大;(2)不同应用的请求交织在内存芯片上,破坏了单个应用的行缓冲局部性,引起更多的行冲突,导致访存延迟增大。

预取技术可以覆盖访存延迟,常用来缓解“内存墙”问题。预取是在程序访问某些数据之前,提前发射这些地址的访存请求,使得指令在执行时能够在相应的存储层次找到需要的数据,不需要等待访存请求返回,从而达到“隐藏”延迟的目的。预取带来的性能提升是吞吐率(单位时间内存系统完成的访存请求的个数)的提升,单个访存请求的延迟没有变化,只是提前发射了所需的访存请求,使得访存延迟相互覆盖,通过提高内存系统吞吐率来提升性能。

预取首先学习以往访存请求的访问模式,然后预测将来可能要访问的地址。按照预取数据的存放位置不同,预取技术分为 2 类:“处理器端”的预取和“内存控制器端”的预取。其中,处理器端的预取更为常见,预取到的数据存放在处理器的高速缓存(Cache)中,通过覆盖 Cache 失效引起的长延迟来提高性能。以往的很多预取技术是处理器端的预取,通常采用对指令地址的分析和学习,预测后续指令可能要访问的地址。但是,当处理器端的预取不够准确时,预取到的数据会污染 Cache,并且增加访存带宽压力,造成性能下降。

另一种预取是“内存控制器端”的预取,通过由内存控制器发出的预取请求,将可能用到的数据提前取到内存控制器中。这种预取可以避免不准确的预取对 Cache 的影响,同时覆盖片外访问延迟,从而提高处理器性能。代价是在内存控制器内的硬件开销。但由于内存控制器收到的访存请求是最后一级

① 国家重点研发计划(2022YFB3105100)资助项目。

② 女,1996 年生,博士生;研究方向:计算机系统结构,处理器设计;联系人,E-mail:zhoushuxin20b@ict.ac.cn。

(收稿日期:2022-12-05)

Cache 的失效请求,访存地址间的局部性很大程度上已经被 Cache 过滤掉了,传统的顺序预取方法不能达到很好的效果。

本文提出一种基于行内局部性的内存控制器端预取方法。采用位图的方式记录访存地址模式,对 Bank 的行进行进一步的区域划分,并量化每个区域的访问局部性,使用区域的局部性来估计这一行的访问局部性。当局部性较高时,通过内存控制器发出预取命令,预取同一个区域内或者跨区域的数据。预取到的数据保存在内存控制器中,当真正需要预取的数据时,不需要再进行片外访问,覆盖了访存延迟,提高了处理器性能。本文在龙芯 3A6000 处理器中实现了这种预取方法,并评测了其性能和硬件开销。

1 相关工作

本节介绍预取技术的相关工作。

1.1 处理器端预取技术

处理器端的预取技术一般通过指令的地址进行访问分类和分析。一类是基于空间模式的预取,另一类是基于时间模式的预取。

基于空间模式的预取技术学习访存地址的空间关系,包括简单的模式(如连续、固定间隔等)和复杂的模式(如周期性间隔)。典型的空间模式预取有以下几种。(1)流预取(stream buffer prefetching, SBP)[2]是一种经典且被广泛使用的识别简单空间访问模式的预取技术。流预取根据访问地址不同对访问进行分类,跟踪记录各个内存区域的访问历史,一旦识别到访问流(地址连续的序列),则向下一级存储层次发出预取请求。(2)基于位图的预取技术[3]是一种识别复杂访问模式的技术。采用位图的存放方式记录内存地址的访问情况,同时采用一种快速的、低开销的逻辑进行访问模式的对比匹配。当某种访问模式出现 3 次,那么对这种模式进行预取。Bingo[4]在文献[3]的基础上进行改进,降低了位图的存储开销。(3)基于地址签名的预取(signature path prefetching, SPP)[5]提出了一种签名机制,压缩存储内存访问模式。通过将签名与地址增量关联,SPP 可以快速、准确地学习简单和复杂 2 类访问模式。同时签名的机制还可以检测连续 2 个物理页之间的访问模式。

基于时间模式的预取技术学习访存地址的先后关系,预测未来可能访问的地址。根据时间局部性原理,已经访问过的地址在未来也可能会被访问。典型的基于时间模式的预取有以下几种。(1)基于时空流的预取[6](spatial-temporal memory streaming, STMS)是一种同时考虑时间访问模式和空间访问模式的预取技术。文献指出单独考虑空间访问模式的预取只考虑了固定规模的内存区域的访问规律,而 STMS 同时记录内存区域内的时间访问顺序,并考虑了区域内和跨区域的空间访问模式的重复,提高了预取准确率和覆盖率。(2)基于时间访问模式的预取技术记录历史的访问地址和预取信息,需要较大的硬件开销。预取准确率指的是预取的数据中被实际用到的部分占所有预取数据的比例;预取覆盖率指的是预取的数据中被实际用到的部分占所有访问请求的比例。预取准确率和预取覆盖率是衡量预取算法的 2 个重要指标。

1.2 内存控制器端预取技术

内存控制器端的预取技术由文献[7]提出,该文献首先指出处理器端预取技术的劣势:不准确的预取会对片上 Cache 造成污染,同时延长请求在内存控制器中排队的延迟,反而引起性能下降。在此背景下,文献提出将预取的数据存放在内存控制器内,不污染 Cache,并且不增加片上网络负载。

1.3 预取反馈与控制

衡量预取带来性能好处的另一个重要指标是预取及时性,指的是预取请求是否及时完成,以便后续访问可以直接取走预取到的数据。文献[8]指出,预取准确率和预取及时性影响预取带来的性能提升,在预取准确率和及时性很差的情况下,预取甚至会带来性能下降。所以该文献提出了根据上述 2 个指标来动态调整预取激进程度的方法,其中,用预取距离和预取度来反映预取的激进程度。预取距离是指预取请求与正常访问请求地址之前的距离;预取度是指一次发出的连续预取请求的个数。通过周期性地统计预取请求的准确率和及时性,以对预取激进程度进行调整。文献[9]也采用了类似的方法对

预取进行控制。而文献[10,11]采用动态学习的方式适时地决策最合适的预取距离。文献[11]同时考虑多个预取距离,根据预取效果反馈决策选用相应的预取距离。

不同于文献[3],本文提出的预取方法将位图的结构应用于内存控制器端预取,用于记录行内数据块的访问情况。并提出采用划分区域的方式估计行内访问局部性程度,根据局部性高低进行预取,将预取到的数据存放在内存控制器内。在龙芯 3A6000 处理器[12]上实现了该方法,并评测了 SPEC CPU2006[13]评测程序的性能。评测结果表明,本文提出的预取方法将每周期指令数(instruction per clock cycle,IPC)平均提升 6.51%,将单线程 IPC 最高提升 46.80%(bwaves),将双核四线程 IPC 最高提升 26.22%(lbm)。

2　基于行内局部性的预取方法

预取技术包括对访问模式的跟踪记录、生成预取请求、对预取的调整和控制 3 个主要部分。本节介绍本文提出的预取技术——一种基于行内局部性的内存控制器端预取。首先介绍基于位图的访问模式的记录与分析,然后介绍如何生成预取请求,最后介绍对预取的调整与控制。

2.1　基于位图的访问模式的跟踪记录

文献[7]指出,SPEC CPU2006 中行内的局部性很高,应用访问同一行内相邻的一个数据块的概率平均可达 36.4%,访问同一行内后续 4 个数据块以内的概率平均可达 43.7%。考虑上述发现,本文提出一种基于行内局部性的预取方法,由访存请求触发行内其他地址的预取请求,针对同一行内的数据进行预取。预取器在内存控制器的位置如图 1 所示。

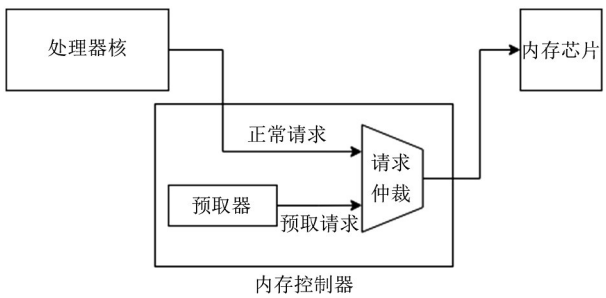


图 1　预取器在内存控制器中的位置

首先对到达内存控制器的请求按行进行预取资源的分配,预取资源包括记录数据块状态的位图、预取请求生成逻辑(见图 2)。预取资源的分配按照行地址进行分配,最多支持 16 个不同行(见图 3)。对于到达内存控制器的访存请求,如果同一行的请求已经分配了预取资源,那么不为这一行重新分配。而对于没有分配过预取资源的一行,当行内的第一个请求到达内存控制器时,为它分配一份预取资源,跟踪记录这一行的访问局部性。如果内存控制器内 16 份预取资源全都已经被分配,那么无法分配预取资源,访存请求直接发往动态、随机存取存储器(dynamic random access memory,DRAM)。对于已经被分配的预取资源,如果一段时间内没有收到该行的访问请求,将这份预取资源释放,提供给其他行。

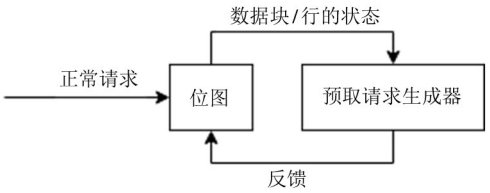


图 2　预取资源

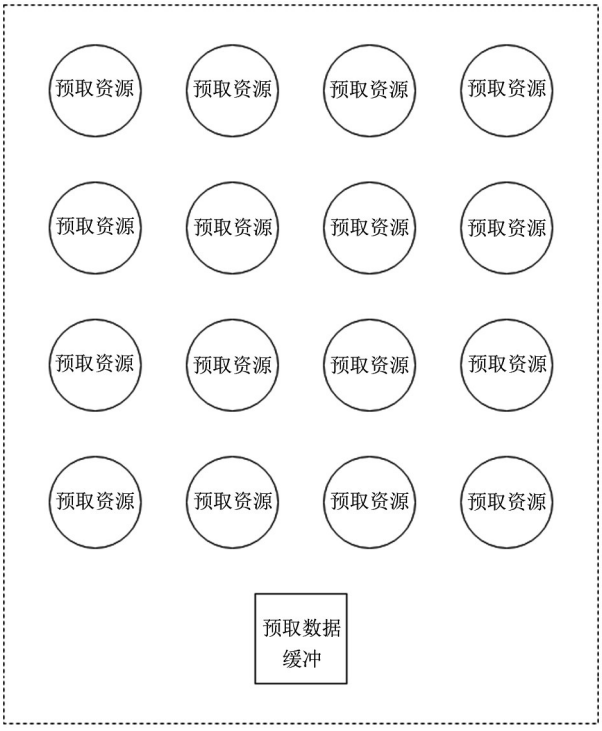


图 3　预取器整体结构

分配预取资源后,后续这一行的访存请求将被

记录在相应的位图中。位图用于量化这一行的访问局部性强度。

本节介绍对访问模式的跟踪记录。首先介绍基于位图的记录方式,然后介绍区域的划分和分类。

2.1.1 基于位图的记录和数据块状态

为了挖掘和利用行内的局部性,首先应对局部性进行量化的统计。本文提出的预取方法采用访存请求触发预取请求的方式,所以应对数据块的状态进行跟踪记录,根据数据块的状态对访问局部性进行统计,然后决定是否要发出预取请求以及预取哪些数据块。

用于记录数据块状态的存储结构有多种形式。由于只关注预取相关的状态,同时为了节约硬件开销,采用位图的结构。下面对位图进行详细介绍。

位图记录了分配预取资源的一行的访问情况(见图4)。位图中的每个表项对应一个数据块(64字节,与 cache line 大小一致),表项内的信息表示这个数据块在近期的访问情况,表项宽度为 2 bits,对应数据块的不同状态。一共有 Init、Access 和 Prefetch(I,A,P)3 种不同状态。

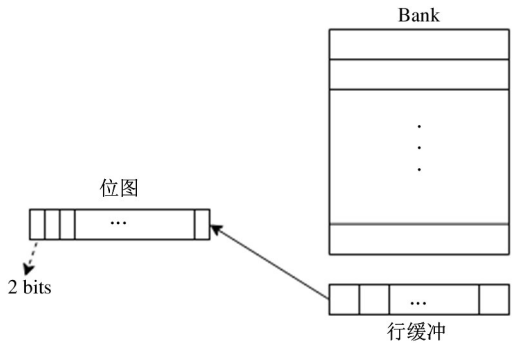


图4 位图与 Bank 的对应关系

当分配一份预取资源给一行时,位图(Row Map Table)内所有数据块的初始状态均为 Init。如果某个数据块对应的预取请求被发出时,这个数据块的状态由 Init 变为 prefetch。当内存控制器收到某个数据块的访存请求时,这个数据块的状态变为 Access。也就是说,“Prefetch”状态表示这个数据块即将被预取,而“Access”状态表示这个数据块被访问了(见图5)。

Row Map Table 内每个表项表示对应数据块的状态(见图6)。如果数据块状态不是 Init,比如是 Prefetch 状态或者 Access 状态,这意味着这个数据块对应的预取请求已经被发射给内存芯片,或者是已经被访问。这时不需要发射这个数据块的预取请求。所以,在本文的预取方案中,只需要发射状态为 Init 的数据块的预取请求。

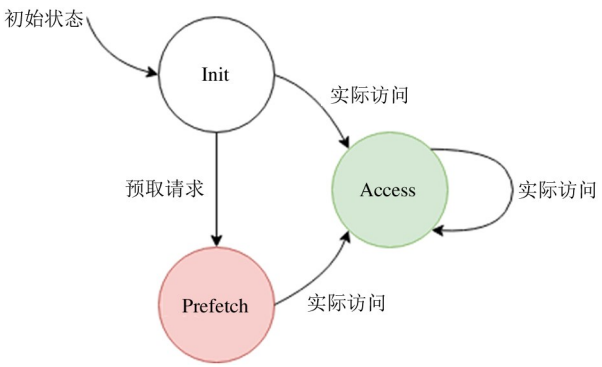


图5 数据块状态

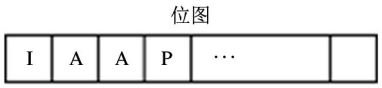


图6 Row Map Table

另外,由于位图的表项的状态变化是单向的,随着时间推移,处于 Access 状态的数据块会越来越多,处于 Init 状态的数据块越来越少,也就是说,内存控制器发射的这一行的预取请求也会越来越少。当 Row Map Table 中处于 Init 状态的数据块数量为 0 时,本文的预取方案不会继续发射预取请求。

内存控制器每次收到访存请求,都查询预取器内对应行的 Row Map Table。如果访存请求所在的行在预取器内没有被分配预取资源,那么先试图为其分配预取资源,将预取资源内的 Row Map Table 中每个表项状态初始化为 Init。如果访存请求所在行在预取器内已经分配了预取资源,那么将 Row Map Table 读出,根据 Row Map Table 中表项的状态决定是否预取。

同时,对于每个访存请求,查询 Data Buffer 中是否有对应的已经预取回来的数据块。如果有,那么直接返回这个请求,不需要将请求再发给内存芯片。

2.1.2 区域

获得每个数据块的状态后,为了统计行内局部

性程度,以及为了避免不准确的预取,将一行再划分为几个区域(zone),进行精确的局部性量化统计,通过细致的局部性量化保证预取准确度。区域的大小以数据块为单位,其规模是动态变化的。区域内访问过的数据块越多,说明在这个区域内的访问局部性越高。另一方面,区域规模越大,越接近一行的规模,说明应用在行内的局部性越高。这样通过区域的局部性可以推测应用在一行内的局部性。

进一步地,根据局部性高低,确定是否预取以及预取的激进程度。使用区域的大小和是否跨区域的预取来控制预取激进程度。对于访问局部性一般的区域,进行区域内部的预取。对于访问局部性较高的区域,进行跨区域的预取。

为了对区域的局部性程度进行区分,对区域的状态进行分类。本文提出的预取方法根据区域内数据块的状态确定区域的状态。按照区域内访问过的数据块数量占区域所有数据块的比例对区域的状态进行分类。

具体而言,区域的状态分为 2 种。一种是活跃(active)状态,收到过访问请求的区域都处于活跃状态。也就是区域内有至少一个数据块的状态为 Access。这一类区域被访问的数据块较少,属于局部性一般的区域,预取将限制在区域内部。另一种是热点(hot)状态,区域内至少一半的数据块被访问过,这一类区域被访问的数据块较多,是局部性较高的区域,将预取包括区域内部和下一个区域的数据块。例如,假设区域的大小为 S 个数据块,一行包含 N 个区域。如图所示,当前 $S=4$,$N=4$,一行内所有数据块的状态如图 7(1)所示。

对于第 1 个区域,只有一个数据块被访问过,属于局部性一般的情况,状态为 active。预取第 1 个区域内其他未被访问过的数据块。而对于第 2 个区域,超过一半的数据块被访问过,表现出较高的访问局部性,状态为 hot。进行较为激进的预取,内存控制器将预取第 2 个区域内部和第 3 个区域的数据块。对于第 4 个区域,由于没有数据块被访问过,其局部性较低,所以不对其进行预取。

2.2　生成预取请求

对于每个到达内存控制器的访存请求,首先获得其所在行地址,查询预取器内是否有这一行的预取资源。接下来更新访存请求的数据块的状态,和这一行的 Row Map Table,获取这一行所有数据块的状态。然后根据当前 S 和 N 的值对行内所有区域进行状态分类。对于处于 active 状态的区域,预取其区域内部未访问过的数据块;对于处于 hot 状态的区域,除了预取它内部的数据块,还要生成对下一个区域的数据块的预取请求。生成的预取请求也会使得 Row Map Table 进行数据块状态的更新,将预取的数据块状态标记为 Prefetch。到此预取器对一个访存请求的处理结束。

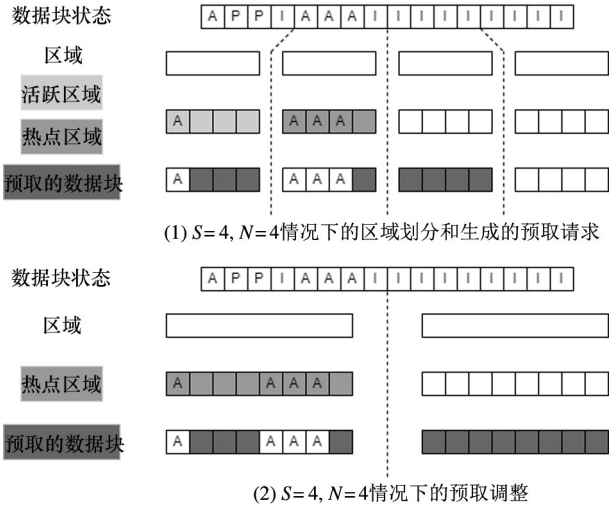


图 7　区域划分和生成的预取请求

2.3　预取调整与控制

为适应不同访问局部性的应用,同时适应应用随着执行阶段而变化的访问局部性,避免多余的预取请求,预取的激进程度应该随着局部性程度的变化而变化。

在本文的预取方法中,区域的大小与预取度和预取距离直接相关。对于 hot 状态的区域,如果触发了跨区域的预取,预取请求的数量越多,预取距离也大。另一方面,区域规模越大,不论是否跨区域预取,预取请求的数量也越多。所以调整区域的大小即可调整预取度和预取距离。对于处在 active 和 hot 状态的区域,其局部性相较其他区域较高。在一行内,这 2 个状态的区域数量越多,说明应用的行内访问局部性越高,应用就有更大的可能在更大地址范围内也有良好的访问局部性,区域的规模可以进

一步扩大(区域规模实际上是预取器对应用的访问局部性的一种量化的统计)。

在本文的预取方法中,通过动态调整区域的大小调整预取的激进程度。具体方法如下:预取器实时计算区域状态为 active 和 hot 的区域占所有区域数量的比例。当发现处于上述 2 种状态的区域个数超过 50% 时,将区域的规模扩大为原来的 2 倍;如果其占比不超过 50%,区域规模稳定不变。在本文的预取方法中,倾向于选择更大的区域规模。也就是说,如果 2 个区域规模都满足 50% 以上的区域处于 active 和 hot 状态,本文的预取方法将选择更大的区域规模。例如,如图 7(2)所示,对于 $S=4(N=4)$ 的区域划分,有 50% 的区域状态为 active 或者 hot,此时检查 $S=8$ 的情况($N=2$),也有 50% 的区域处于 active 或者 hot 状态,那么选择 $S=8$。这样可使预取请求覆盖更大的地址范围。

3 实验结果与分析

3.1 实验平台

在龙芯 3A6000 处理器的仿真环境中进行评测。龙芯 3A6000 处理器是基于 LoongArch 指令系统的超标量乱序高性能处理器。主要参数见表 1。

评测程序采用 SPECCPU 2006。本文评测了双核四线程和单线程的结果。

表 1 龙芯处理器主要参数

处理器	Loongson3A6000
核数	4
主频	2.0 GHz
指令集	LoongArch 1.0
L1 Cache	I-Cache 64 kB D-Cache 64 kB
L2 Cache	256 kB
LLC(Shared)	8 MB
内存芯片	2×8 GB UDIMM DDR4-3200

3.2 实验结果

在龙芯 3A6000 处理器上评测了 2 个对比方案的 IPC:其一是不采用内存控制器端预取的方案,即基础方案;其二是采用本文提出的内存控制器端预取的方案。评测程序使用 SPEC CPU2006 的访存密集型应用,测试集为 train。针对 2 个方案分别评测单核和双核四线程的性能。相比于基础方案,本文提出的预取方案带来的性能提升如图 8 和图 9 所示。

评测结果显示,在单核情况下,本文的预取方案可以显著提高性能,平均提高 6.51%,个别应用性能略有下降,如 soplex。几个应用性能几乎没有变化,如 omnetpp, milc, leslie3d。在双核四线程情况下,本文的预取方案评价提高性能不明显,平均提高

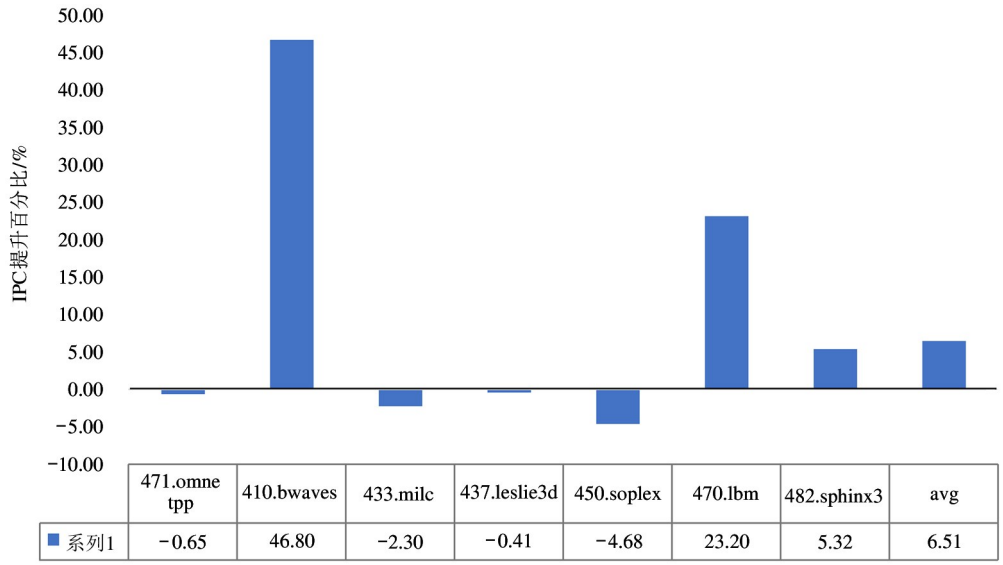
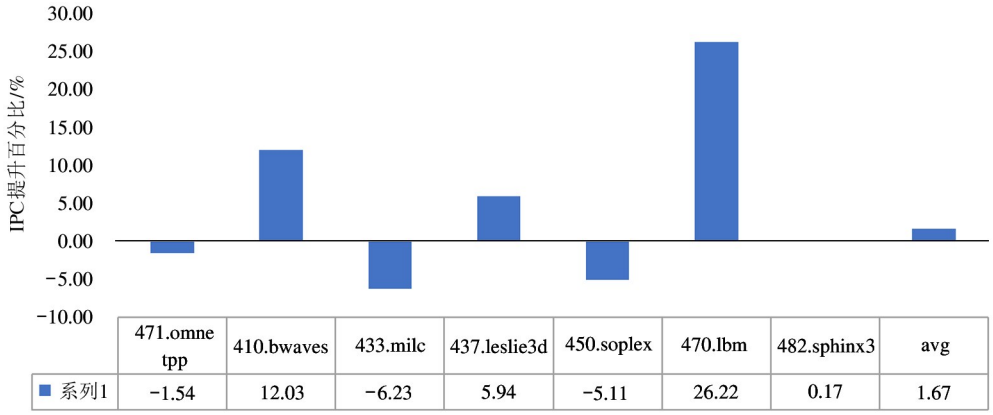


图 8 单线程性能提升



	471.omnetpp	410.bwaves	433.milc	437.leslie3d	450.soplex	470.lbm	482.sphinx3	avg
系列1	-1.54	12.03	-6.23	5.94	-5.11	26.22	0.17	1.67

图 9　双核四线程性能提升

1.67%，甚至可能给一些应用带来性能下降，如 omnetpp，milc，soplex，这是因为多个应用的访存请求在 Bank 上相互交织，破坏了单个应用的行内局部性。

4　结　论

“内存墙”问题在多核处理器中更为严重，预取是一种常用的通过覆盖访存延迟来提高性能的方法。按照预取的数据存放的位置不同，分为处理器端的预取和内存控制器端的预取。尽管以往的很多预取技术都是处理器端的预取，但是当预取准确度较低时，预取的数据对 Cache 造成污染，反而使得性能(IPC)降低。而内存控制器端的预取不会影响片上 Cache。同时，研究发现，SPEC CPU2006 中，在一个访存请求后，访问同一行内后续 4 个数据块以内的概率平均可达 43.70%，说明行内的局部性较高。本文提出一种基于行内局部性的内存控制器端预取方法。采用位图的数据结构记录行内每个数据块的状态；对每一行进行区域的划分，量化每个区域的访问局部性；根据区域内的局部性高低决定预取的激进程度。对于局部性较低的区域，预取区域内剩余的数据块；对于局部性较高的区域，采用跨区域的预取。同时，动态地调整区域规模的大小来适应变化的局部性程度。上述预取方法在龙芯 3A6000 处理器上实现并评测，评测程序采用 SPEC CPU2006。评测结果显示本文的预取方法将性能平均提升6.51%，将单线程 IPC 最高提升 46.80%(bwaves)，将双核四线程 IPC 最高提升 26.22%(lbm)。

参考文献

[1] WULF W A, MCKEE S A. Hitting the memory wall: implications of the obvious[J]. ACM SIGARCH Computer Architecture News, 1995,23(1):20-24.

[2] PALACHARLA S, KESSLER R E. Evaluating stream buffers as a secondary cache replacement[C]//Proceedings of the 21st Annual International Symposium on Computer Architecture. Valencia, Spain: IEEE, 1994:24-33.

[3] ISHII Y, INABA M, HIRAKI K. Access map pattern matching for high performance data cache prefetch[J]. Journal of Instruction-Level Parallelism, 2011,13:1-24.

[4] BAKHSHALIPOUR M, SHAKERINAVA M, LOTFI-KAMRAN P, et al. Bingo spatial data prefetcher[C]//2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). Washington, USA: IEEE, 2019:399-411.

[5] KIM J, PUGSLEY S H, GRATZ P V, et al. Path confidence based lookahead prefetching[C]//2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Taibei, China: IEEE, 2016:1-12.

[6] SOMOGYI S, WENISCH T F, AILAMAKI A, et al. Spatio-temporal memory streaming[J]. ACM SIGARCH Computer Architecture News, 2009,37(3):69-80.

[7] YEDLAPALLI P, KOTRA J, KULTURSAY E, et al. Meeting midway: improving CMP performance with memory-side prefetching[C]//Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques. Edinburgh, UK: IEEE, 2013:289-298.

[8] SRINATH S, MUTLU O, KIM H, et al. Feedback directed prefetching: improving the performance and bandwidth-efficiency of hardware prefetchers[C]//2007 IEEE 13th International Symposium on High Performance Computer Architecture. Scottsdale, USA: IEEE, 2007:63-74.

[9] HEIRMAN W, BOIS K D, VANDRIESSCHE Y, et al. Near-side prefetch throttling: adaptive prefetching for high-performance many-core processors[C]//Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques. Atlanta, USA: PACT. 2018:1-11.

[10] PUGSLEY S H, CHISHTI Z, WILKERSON C, et al. Sandbox prefetching: safe run-time evaluation of aggressive prefetchers[C]//2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA). Orlando, USA: IEEE, 2014:626-637.

[11] MICHAUD P. Best-offset hardware prefetching[C]// 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). Barcelona, Spain: IEEE, 2016:469-480.

[12] 胡伟武, 高翔, 张戈. 龙芯指令系统架构及其软件生态建设[J]. 信息通信技术与政策, 2022,48(4):43-48.

[13] HENNING J L. SPEC CPU2006 benchmark descriptions [J]. ACM SIGARCH Computer Architecture News, 2006,34(4):1-17.

Memory controller-side prefetching based on intra-row locality

ZHOU Shuxin[*,**,***], ZHANG Jianqi[****], WANG Huandong[****], ZHANG Longbing[*,**,***]

([*] National Key Laboratory of Processor Chips, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***] University of Chinese Academy of Sciences, Beijing 100049)

([****] Loongson Technology Corporation Limited, Beijing 100190)

Abstract

This paper proposes a memory controller-side prefetching based on intra-row locality. The data structure of the bitmap is used to record the state of each data block in the row. And each row is divided into regions, and the access locality of each region is quantified. The aggressiveness of prefetching depends on the locality in the region. For areas with low locality, unaccessed data blocks in the area will be prefetched, and for areas with high locality, cross-area prefetch will be adopted at the same time. It adapts to changes in the degree of locality by dynamically adjusting the size of the region scale. The above prefetching method is implemented and evaluated on the Loongson 3A6000 processor using SPEC CPU2006 memory-intensive applications. The evaluation results show that the prefetching method in this paper improves the instruction per clock cycle (IPC) by 6.51% on average (up to 46.80% for single-thread, up to 26.22% for dual-core four-thread).

Key words: memory controller, prefetch, locality