多核存储系统体系结构课程项目报告

Sangam：一种多组件缓存预取算法的实现

组员：陈麒先 2001213040

组员：黎卓 2001213041

学院：信息科学技术学院

2021 年 1 月

郑重声明

本项目由本小组成员合作完成。抄袭行为在任何情况下都是不能容忍的(**COPY is strictly prohibited under any circumstances**)!

由抄袭所产生的一切后果由抄袭者承担，勿谓言之不预也。

陈麒先、黎卓

目录

摘要

数据预取是一项非常重要的技术，目前，该技术已得到广泛应用。几乎所有的商用处理器中均采用了一些数据预取的策略。数据预取的主要目的在于消除某些大数据的访问延迟，进而提高程序整体的运行性能。而缓存预取是数据预取当中一种常用的策略，在本文中，我们主要参考了 Sangam: A multi-component core cache prefetcher 这篇论文，对其中实现的预取策略在 ChampSim 模拟器上进行了复现。

在 Sangam 中，主要设计了对 L1D 缓存、L2C 缓存和 LLC 的数据预取策略。使用三种不同组件，构建了一个分层的组织架构，以应对复杂多变的数据访问场景。对于每一层级而言预取器应用不同的数据流和控制流信息，例如，第一级预取器利用指令指针（以下简称 IP）以及数据流和控制流的增量在访问流中的联合特性，能够以一个较高的置信度来预测增量序列。为了保证高覆盖率，当基于 IP 增量的预取器不能提供一个高置信度的预测时，则会插入并启用第二级预取，即一种基于 IP 步长的预取器。若第二级失效，则会在预取器第三级触发一个仅基于数据流的，具有自适应度的下一行预取器。将三级预取器集成在 L2 缓存中，并在与 L1 缓存协同工作时进一步改善性能。此外还增加了三级 L1 缓存预取器，使用一种优化 L1 缓存查找带宽的技术来优化预取所消耗的带宽。最终，在 ChampSim 模拟器上所运行的结果表明，使用 Sangam 预取后的模拟仿真后的 IPC 要优于不使用预取策略的情况。

关键词：缓存预取，分层预取策略，ChampSim，Sangam

一、项目背景

随着计算机硬件的发展，CPU 主频已由过去 MHz 级发展到了 GHz 级，而常用硬盘的存取速率还不到 100M/S。并且根据摩尔定律，微处理器的速度以及单片集成度每 18 个月就会翻一番，但像磁盘这样的机械电子设备，存取速率每年仅增加约 8%。由此看来，磁盘 I/O 无疑是整个性能提升的瓶颈，而且磁盘的访问速率与 CPU 的速度差距还在持续扩大。常用的存储介质从光盘、磁盘、内存到高速缓存，存取速度越来越快，但是成本也越来越高。为了在成本和性能之间进行平衡，现代计算机体系架构往往选择使用少量性能高但成本也高的存储器作为速度慢而成本也低的存储器的缓存。所以整个存储层次如同一个金字塔结构。现代处理器速度的快速发展和存储器速度的慢速发展导致处理器要花费大量的时间等待存储器数据的返回，这就是存储墙问题。例如，Alpha 21264 667 MHz 的工作站一次访存失效造成的开销就高达 128 个时钟周期。为了解决这些问题，已提出多种技术方案，其中最主要的有缓存和预取技术两种。缓存技术是利用局部性原理，使速度更快的上层存储器成为下层存储器的缓冲。预取技术可以通过计算和访存的重叠，隐藏因为缓存延时而引起的缓存失效，被认为是解决容量失效和强制性失效的有效手段。

在当今处理器的多级缓存层次结构中，根据设计，预取器可以将数据块带入缓存层次结构的不同层次。然而，由于 L1 缓存是所有缓存级别中最接近处理器、最小的，因此设计高精度、高覆盖率的高效 L1 缓存预取器非常重要。及时地将未来需要访问的数据块提取到 L1 缓存中，可以提供更佳的性能。此

外，在研究中注意到 L1 缓存是学习整个指令流访问行为的最佳位置，这是因为在 L1 缓存中可以观察到未经过滤的访问流，这使得模式学习过程最为可靠。本项目在一个三级使用 L1、 L2 和 L3 缓存的缓存层次结构，在 L1 缓存中加入一个基于 IP 的四度跨步预取器，在 ChampSim 项目中的 dpc3_trace 数据集下，可以有效提高 IPC。在 L2 缓存中使用同样的预取器只能实现有限的加速效果。有两个原因可以解释这种现象。首先，L2 缓存预取器从过滤的访问流中学习步长的变化规律，导致步长规律的总结过程是不可靠的。其次，预取块不会一直被保留并传输到 L1 缓存。这种错误模式的学习可以通过简单的途径进行解决，即将 L1 缓存中的预取器进行合并，并将预取到的数据块传送到 L2 中进行缓存。这种优化将上述基于 IP 的步长预取器实现的加速效果进一步提高。基于上述事实的启发，Sangam 预取算法专注于设计一个 L1 缓存预取器，试图将所预取的数据块传送到 L1 缓存进行缓存。

Sangam 预取策略中的预取器包含三个不同的组件，并且以分级层次结构的方式利用各个组件中不同数量的控制流和数据流信息。第一级预取器组件是一个基于 IP- 增量联合特征的增量序列预取器，第二级则是基于 IP 的步长（stride）的预取器，第三级组件为一个仅基于数据流触发的具有一定自适应度的下一行（next line）预取器。我们通过两个重要的技术进一步增强了 L1 缓存预取器的预取效果。第一种是优化预取所消耗资源的 L1 缓存查找带宽技术，第二种是在 L1 缓存控制器资源短缺的情况下保持 L1 缓存预取的激进的新机

制。总体而言，使用 Sangam 预取策略后，在 ChampSim 上模拟的 IPC 结果整体上得到了明显改善。

二、相关研究

近年来，人们在数据预取领域做出了大量贡献，这些预取器具有深入的前瞻性和更好的及时性。这些预取方案通常设计增量预测器得到预取序列。

Jinchun Kim 等人提出了签名路径预取器（Signature Path Prefetcher，SPP）[2]。SPP 首先基于压缩历史记录的方案来准确预测复杂的地址模式。接着当地址模式在物理页面之间转换时，SPP 会在物理页面边界上跟踪复杂的模式，并在它们移至新页面后继续进行预取。最后，SPP 使用其预测的置信度以预取流为基础自适应调整自身。SPP 预期器在在覆盖范围和准确性之间取得了一定的折中。沙箱预取通过将预取地址添加到布隆过滤器中，而不是实际将数据提取到缓存中，从而在运行时评估简单的主动偏移预取器。针对布隆过滤器的内容对后续的缓存访问进行测试，以查看评估中的主动式预取器是否可以准确地预取数据，同时测试是否存在可预取的流。当评估的预取器的准确性超过阈值时，将执行实际的预取。此方法结合了全局模式确认和立即预取操作的思想，以实现高性能。

Seth H Pugsley 等人提出了沙盒预取器（Sandbox Prefetching）[3]，在运行时为当前正在执行的程序确定适当的预取机制。沙箱预取通过将预取地址添加到布隆过滤器中，而不是实际将数据提取到缓存中，从而在运行时评估简单的主动偏移预取器。针对布隆过滤器的内容对后续的缓存访问进行测试，以查看评

估中的主动式预取器是否可以准确地预取数据，同时测试是否存在可预取的流。当评估的预取器的准确性超过阈值时，将执行实际的预取。此方法结合了全局模式确认和立即预取操作的思想，以实现高性能。

Pierre Michaud 发现沙盒方法虽然有效但没有考虑预取的及时性，在此基础上提出了最佳偏移量预取器（Best-Offset prefetcher，BOP）[4]，BOP 对预取偏移量进行学习，从而动态选择考虑了时效性的预取偏移量，实现更好的及时性。

Manjunath Shevgoor 等人提出了可变长度增量预取器（Variable Length Delta Prefetcher，VLDP）[5]。VLDP 建立了物理页面内连续高速缓存行未命中之间的增量历史记录，然后使用这些历史记录来预测新页面中高速缓存行未命中的顺序。VLDP 的显着特征之一是它使用多个预测表，每个预测表都根据输入历史记录的不同长度存储预测。较长的历史记录通常会产生更准确的预测，因此 VLDP 倾向于根据具有匹配条目的最长历史记录表进行预测。

另一部分方案则研究了服务器工作负载上下文中的指令特征和数据预取技术。

Michael Ferdman 等人提出了最后触摸相关数据流预取器（last-touch correlated data streaming，LT-cords）[6]，这是一种实用的地址相关预测器。LT-cords 的关键思想是按需要使用的顺序在片外记录相关数据，并在需要它们之前不久将它们流式传输到实际大小的片上表中，从而避免了对可扩展的片上表的需求。启用低延迟查找。

现在已经有学者研究出了能感知不同线程的预取器，以用于多线程工作负载。Laith M. AlBarakat 等人提出了可感知多线程的硬件预取器[7]，该预取器利用线程花费在等待同步语义上的时间在关键部分之前运行，以推测和预取独立于同步语义之外的加载指令数据。它可以通过显着减少预取需求的干扰来从芯片多处理器中进行预取，从而大大提高性能。

Akanksha Jain 等人提出了不规则流缓冲区预取器（Irregular Stream Buffer，ISB）[8]，它是一种针对时间相关的内存引用的不规则序列的预取器。 关键思想是使用额外的间接级别，将任意对相关物理地址转换为新结构地址空间中的连续地址，该结构地址仅对 ISB 可见。 此结构化地址空间使 ISB 可以组织预取元数据，以便同时对其进行时间和空间排序，从而在覆盖范围，准确性和内存流量开销方面产生了技术优势。

Anant Vithal Nori 等人提出了关键性感知分层缓存层次结构（Criticality Aware Tiered Cache Hierarchy，CATCH）[9]，该方法利用对硬件中程序关键性的准确检测，并使用一组新颖的缓存间预取器，确保在关键执行路径上进行片上数据访问。最快的 1 级（L1）缓存的延迟。最后一级缓存（LLC）的目的是减少缓慢的内存访问，从而使大型 L2 缓存对于大多数应用程序而言都是冗余的。然后，通过消除 L2 缓存节省的区域可用于创建更高效的处理器配置。

Carole-Jean Wu 等人表示 LLC 管理与硬件预取之间的交互很少受到关注，在这样的背景下他们提出了预取感知缓存管理（Prefetch-Aware Cache

Management，PACMan）[10],通过修改缓存插入和命中率提升策略来以不同方式对待需求和预取请求，从而动态估计并减轻预取引起的缓存干扰的程度。

三、预取器设计

本文预取策略总体包括如下三个组件预取器，基于 IP-增量序列预测器，基于 IP 步长预测器和具有自适应程度的下一行预取器。预取器整体流程框架如下图 1 所示。每次访问 L1 高速缓存时，都会按需查找 L1 高速缓存预取器组件，并生成预取序列。对于所有三个组件，都维持一个通用的基本预取度 d，这意味着在每个 L1 缓存需求访问中最多注入 d 个预取。接下来本文将讨论具体细节。

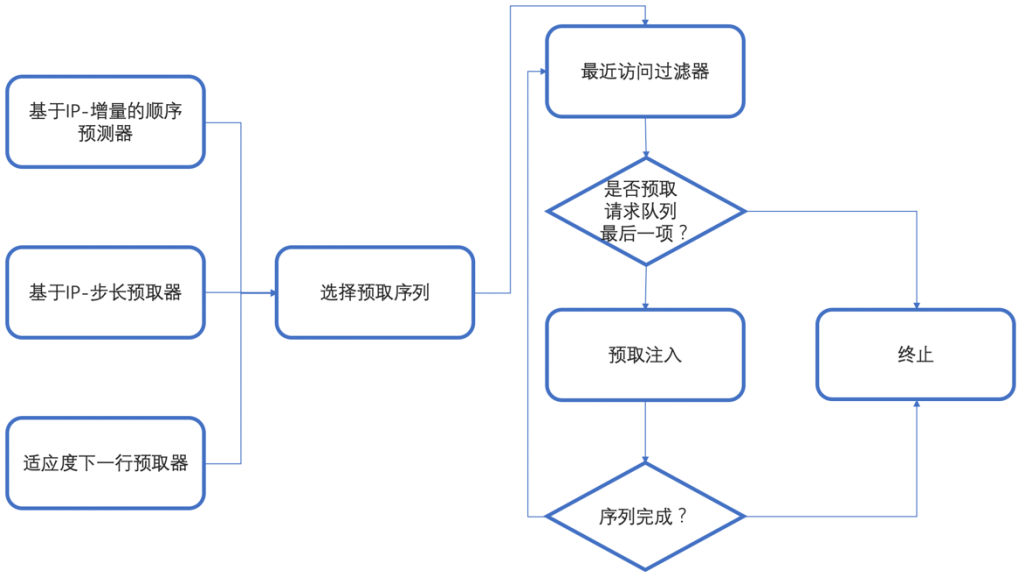


图 1：sangam 预取器总体流程框架

3.1 基于 IP-增量的序列预测器

基于 IP-增量的序列预测器是 sangam 预取器的第一级组件。该组件首先根据请求访问的源 IP 对所有请求访问进行分类。而后，对于来自特定 IP 源的给定访问，并且相对于来自同一 IP 的最后一次访问有一个增量，预测器预测一个接下来的增量序列。两次访问之间的增量定义为当前访问的偏移量(即页面内的缓存块编号)与上一次访问的偏移量之间的差值。使用这个预测器的动机来自于这样的事实: 即在来自特定 IP 的访问中，在给定增量Δ 之后出现的顺序序列通常在相同的增量后重复出现，例如$\Delta, \delta1, \delta2, \delta3, \delta4 \ldots, \Delta, \delta1, \delta2, \delta3, \delta4 \ldots$ 这种形式。

该预测器需要两个标记的集合关联表。第一个表称为 IP 表，这些表以用于对表进行索引的实体命名，使用当前请求访问的 IP 查找。IP 表的每一项都存储了一个 FIFO（先进先出）队列，其中包含了最后看到的与索引值 IP 相对应的 d + 1 个增量。它还存储最后一个偏移量，以便计算下一个增量。除此之外，每一项都有一个部分标记、有效位和 LRU 状态。如果一页中有 n 个缓存块，我们将使用 1 + log2(n) 个位对每个增量进行编码，其中大多数增量的有效位对其符号进行编码。

第二个表称为 IP-增量表，它使用当前需求访问的 IP 和由该 IP 源的访问流中的当前增量的串联来查找。给定索引的源 IP，该表的每个条目都存储一个 d-增量序列，该 d-增量位于用于对该表进行索引的增量之后。每个 d-增量还有一个两位置信度计数器。每个条目还存储一个部分标记、一个有效位和 LRU

状态。这个表采用如下策略进行更新。首先，使用当前 IP 查找 IP 表，并更新相应的 d + 1 个增量组成的 FIFO 队列，每次更新时在尾部插入一个新增量，替换最久未使用的增量。

接着，对于每个 k，使用 FIFO 队列的第 k 个增量 (其中，k = 0 表示是最久未被使用的)和 IP 来为 IP-增量表建立索引。相应的 IP-增量表条目的第 d-k 个增量项被更新为 FIFO 队列中新插入的增量。如果新增的增量与 IP-增量表条目的位置 d-k 处的增量相匹配，则该位置的置信度计数器将增加 1，否则该位置的置信度计数器将重置为零。 通过使用当前 IP 和增量查找 IP-增量表来获得预测结果。

增量序列从匹配的 IP-增量表项中读出，从而避免了 miss 事件的产生。如果读出序列中的 增量 的置信度低于阈值τ_c，那么增量的预测值为零。这个增量序列用于生成一个预取序列直到增量的预测值为 0 为止。当遇到增量为 0 且生成的预取数量小于 d 时，将检查增量序列中最后两个非零增量是否相同。如果它们是相同的，将继续对该增量序列进行预取，否则预取序列将提前终止。如果在 IP-增量表或 IP 表中出现错误或异常，则无法生成预测结果。测试结果表明，使用最后几个 IP 的散列以及最后几个增量来索引 IP-增量 表有效地提高了预测器的准确性，但是显著降低了表命中率。因此，在最终的具体实现中选择使用单个 IP 和单个基于增量的 IP-增量表索引方案。

3.2 基于 IP 步长的预取器

当基于 IP-增量的序列预测器由于 IP-增量表的错误或预测置信度低而不能提供预测时，我们则改为利用 IP 表的步长进行预测。这种策略避免了覆盖率的损失。这种预测不会产生任何额外的开销，因为无论如何都需要查询 IP 表。在通过插入当前增量来更新匹配 IP 表条目的 FIFO 队列之后，如果 FIFO 队列中最新的两个增量是相同的，那么当 IP-增量表由于缺失或置信度低而不能提供预测时，这个增量用于生成长度为 d 的预取序列。因此可以使用从 IP 表中预测的增量来完成一个由于 IP-增量表的预测序列中的低置信度增量而过早终止的预取序列，如前文所述。

3.3 下一行（next line）预取器

IP-增量表预测和 IP 表的步长预测包含了一个固有的置信度概念，这对于 L1 缓存预取器避免局部缓存变量污染非常重要。但是在这些预测机制都不能提供预测的情况下，就应该使用第三级预取组件，即下一行预取器。在反馈机制的帮助下，可以有效避免使用不准确的下一行预取导致 L1 缓存被污染。在具体实现上，则表现为在一个小型的完全关联的下一行缓冲区(NL buffer)中插入所有达到 d 级的下一行预取候选块。该缓冲区的每一项都包含一个标记、log2(d)位插入标记的预取度、一个有效位和 LRU 状态。每个需求访问都会查找这个缓冲区中匹配的标记。对于每个预取度 d，需要维护两个计数器，一个用于计算度 d 上 NL 缓冲区插入的次数，另一个用于计算度 d 上插入项需要的命中次数。当需要访问命中一个条目时，它会根据匹配项中记录的程度增加命

中计数器。此时，匹配项从 NL 缓冲区清除，标记为失效项。在向 NL 缓冲区中插入度为 d 的预取候选项时，如果插入的标记尚未存在于 NL 缓冲区中，则累加相应的插入计数器的值。如果要插入的标记已经存在于缓冲区中，并且过去插入该标记的预取度更高，我们将条目中的度设置为 d，增加与度 d 相对应的插入计数器，并减少与旧的较高度 d 相对应的插入计数器。当从基本需求地址生成一个距离或度为 d 的下一行预取时，查找度为 d 的命中和插入计数器的比率，如果该比率超过阈值τ_d，只将预取插入到预取队列中即可。

3.4 最近访问过滤器

理想情况下，每个预取请求都应该是准确的，并且应该从 L1 缓存中产生一个新的未命中。这样可以确保预取消耗的 L1 缓存查找带宽不会被浪费。但是，在不查找缓存的情况下，很难确定 L1 缓存中哪些预取会被触发。本文大致上通过维护一个小型的全相连最近访问缓冲区来解决这个问题，这个缓冲区用于跟踪最近在需求访问流中看到的标记和最近插入的预取。这些标记最有可能停留在 L1 缓存或 L1 缺失状态保持寄存器(对缺失来说)。一个新生成的预取请求查找这个缓冲区，在命中的情况下，预取被丢弃。我们注意到在这个估计中，需要保守地进行删除操作，因为删除一个在 L1 缓存中错过的真正的预取会降低覆盖率。因此，最近访问缓冲区的容量很小。

3.5 处理存储资源短缺

L1 缓存控制器通常没有配备存储资源来实现激进地预取。预取使用的一个重要资源是预取请求队列(prefetch request queue，PQ)。一个新的预取插入到 PQ

的尾部。对于一个小的 PQ，预取的激进性很可能会因为 PQ 经常充满而降低。一种选择是将预取器移动到存储空间更大的 L2 缓存。然而，正如已经指出的，从 L2 缓存获得的过滤流中学习访问模式是不可靠的。本文提出了一个简单的解决方案——利用 L1 缓存预取器和 L2 缓存之间的通信。在注入预取时，L1 缓存预取器检查 PQ 的占用情况。如果只剩下一个插槽，它将再注入一个预取，并随之附加有关剩余预取(如果有的话)的信息，这些信息是由于缺少 PQ 条目而无法注入的。L2 缓存预取器检查搭载的信息(作为编码在预取包中的 32 位元数据传递)，对信息进行解码，并注入剩余的预取。这些预取信息将被填充到 L2 缓存中，而不会传播到 L1 缓存中。这个搭载工具用于基于 IP 增量序列的预取和基于 IP 步长预取。如果下一行预取器耗尽了 PQ 空间，它就不会被使用，因为这个预取器组件一般来说远不够精确。背载信息使用如下两种可能的编码方式之一: 增量序列(用于基于剩余 IP 增量序列的预取)和常量增量(用于剩余的基于 IP 步长预取)。32 位元数据中最重要的一位被用来区分这两种编码。剩下的 31 位要么编码一系列增量，要么只编码一个步长。

3.6 L2 缓存预取器

虽然一个精确的高覆盖率的 L1 缓存预取器本身可以非常有效，但是一个 L2 缓存预取器在 L1 缓存未命中时触发进一步的预取可以提供非常深入的前瞻性。然而，一个非常深入的前瞻很少保持准确。尽管如此，我们还是在 L2 缓存中复制了我们的 L1 缓存预取器，并进行了如下两个简化。首先，在 L2 缓存预取器中没有最近访问缓冲区，因为 L1 缓存的高命中率使得 L2 缓存访问

带宽很大。其次，L2 缓存不会将任何剩余的预取信息搭载到 L3 缓存，主要是因为在 L2 缓存 PQ 相当大的情况下，这种设施几乎不会需要。本文注意到 L2 缓存在预取访问中，预取器可能首先注入来自 L1 缓存的编码格式的剩余预取，然后调用 L2 缓存预取器来注入更多的预取。在这两种情况下，L2 缓存使用一个基本的预取度，预取的总数可以达到基本预期度的两倍。

3.7 存储开销

我们假定 4KB 的页和 64B 的缓存块来计算预取器的存储开销。因此，每个页面有 64 个缓存块，因此，每个增量可以用 7 位编码表示。对于单核和多核，我们将 L1 缓存基预取度设置为 4，L2 缓存基预取度根据单核和多核分别设置为 3 和 2。通过适当调整部分标签的大小，IP 表的每个条目被限制在 63 位。同样，IP 增量表的每个条目都被限制为 64 位。NL 缓冲区的每个条目是 73 位，而最近访问缓冲区的每个条目是 71 位。表 1 和表 2 分别显示了 L1 和 L2 缓存预取器的存储开销。每个核的总开销是 63.1 KB。我们的设计使用了两个阈值参数，其值被列在下表中。

表 1 L1 缓存预取器存储开销

结构	存储
模拟内核数	1
IP 表	128 组，15 路，120690 bits
IP-增量表	256 组，8 路，131072 bits
NL 缓冲区	64 项，4672 bits

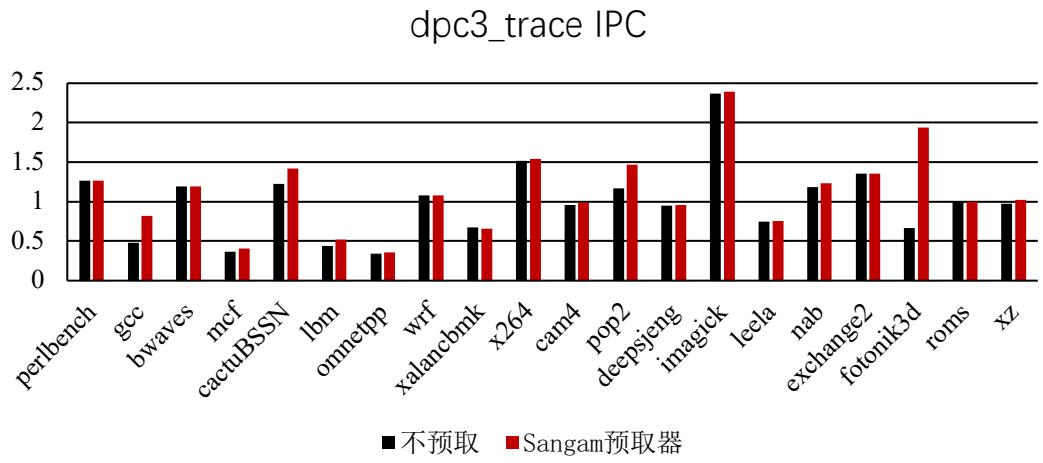
最近访问缓冲区	40 项，2840 bits
辅助计数器与寄存器	316bits
总计	259870bits

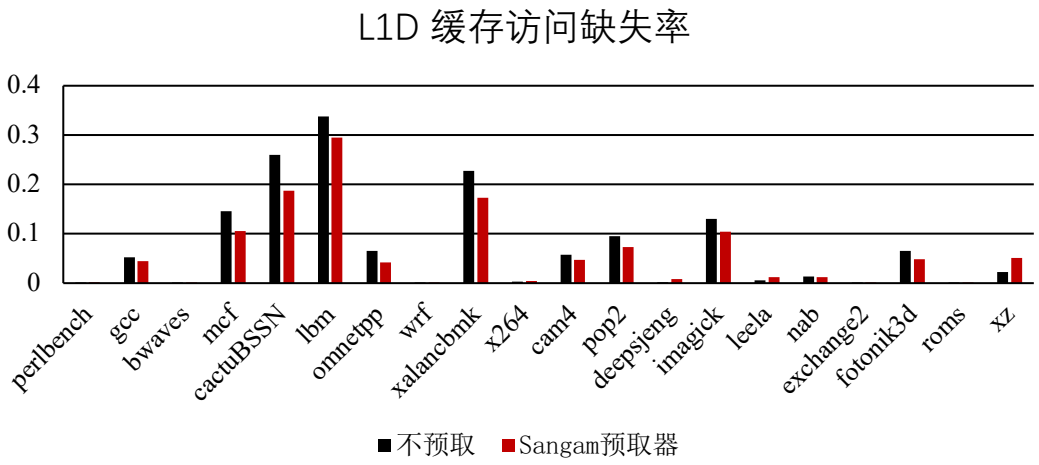
表 2 L2 缓存预取器存储开销

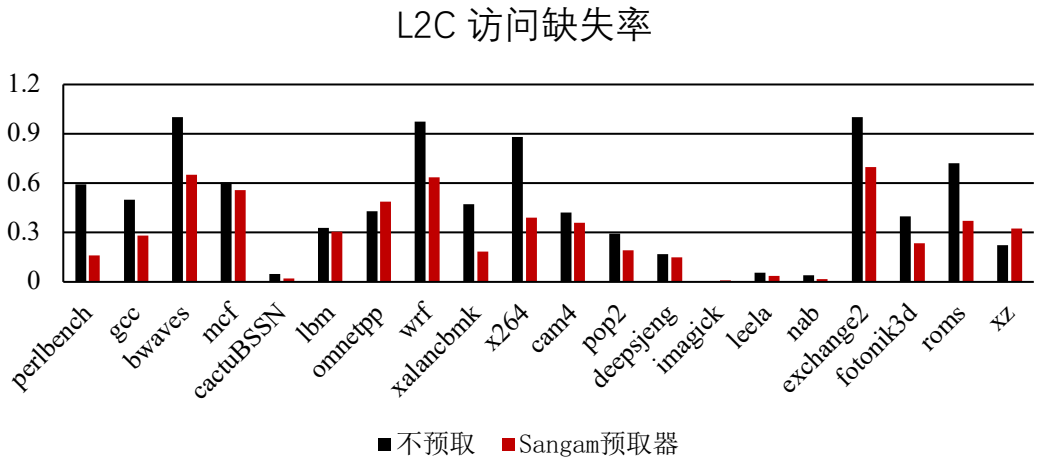
结构	存储
模拟内核数	1
IP 表	128 组，15 路，120690 bits
IP-增量表	256 组，8 路，131072 bits
NL 缓冲区	64 项，4672 bits
辅助计数器与寄存器	248 bits
总计	256952 bits

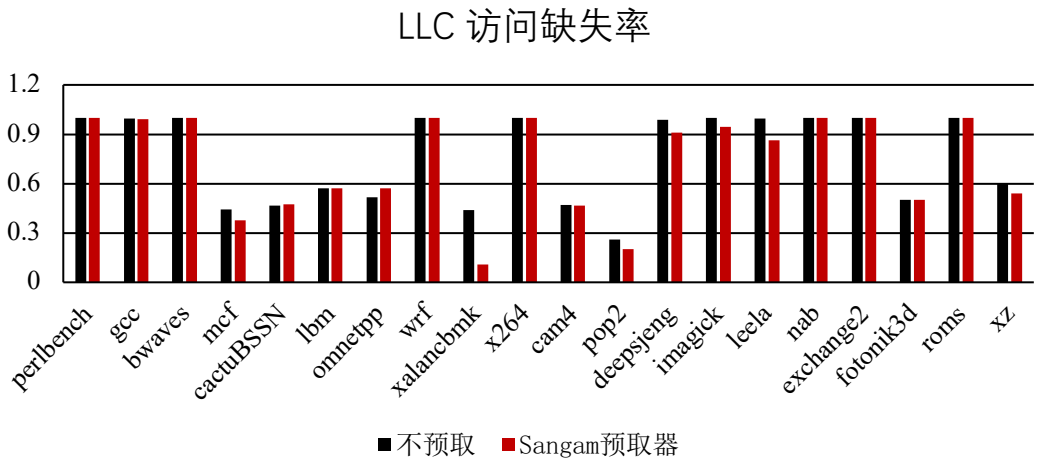
四、实验结果

本文使用 ChampSim 模拟器评估 Sangam 预取器的效果，它是第三届据预取冠军赛（DPC-3）[1]中使用的模拟器。本文对一个内核进行建模，其参数可以在表 1 和表 2 中找到。ChampSim 是基于踪迹的模拟器，本文使用的踪迹是 DPC3 官方踪迹，该踪迹是通过追踪 SPEC CPU 2017 得到的。本文一共对 20 个踪迹进行了实验，其中包括 10 个整型用例和 10 个浮点用例。具体名称分别如下：perlbench、gcc、bwaves、mcf、cactuBSSN、lbm、omnetpp、wrf、xalancbmk、x264、cam4、pop2、deepsjeng、imagick、leela、nab、exchange2、fotonik3d、roms、xz。为了进行性能评估，本实验为内核预热了 1M 条指令，并通过其他 10M 条指令收集了结果。本文选择的性能评估指标分别为：CPU 每一时钟周期内所执行的指令数（Instruction per Clock, IPC）、L1D 缓存访问缺失率、L2C 访问缺失率、LLC 访问缺失率。本文将实验结果绘制成如下柱形图。




L1D 缓存访问缺失率
0.4
0.3
0.2
0.1
0
perlbench
gcc
bwaves
mcf
cactuBSSN
lbm
omnetpp
wrf
xalancbmk
x264
cam4
pop2
deepsjeng
imagick
leela
nab
exchange2
fotonik3d
roms
xz
不预取
Sangam预取器


L2C 访问缺失率
1.2
0.9
0.6
0.3
0
perlbench
gcc
bwaves
mcf
cactuBSSN
lbm
omnetpp
wrf
xalancbmk
x264
cam4
pop2
deepsjeng
imagick
leela
nab
exchange2
fotonik3d
roms
xz
不预取
Sangam预取器


LLC 访问缺失率
1.2
0.9
0.6
0.3
0
perlbench
gcc
bwaves
mcf
cactuBSSN
lbm
omnetpp
wrf
xalancbmk
x264
cam4
pop2
deepsjeng
imagick
leela
nab
exchange2
fotonik3d
roms
xz
不预取
Sangam预取器

结论

由于处理机和存储器发展速度不一致的原因，存储中的数据访问速度逐渐成为限制程序性能的重要瓶颈。数据预取是一种消除某些大数据的访问延迟，进而提高程序整体的运行性能的重要技术手段。本文中主要参考 Sangam 预取策略，实现了一种分层级多组件预取器，在 ChampSim 模拟器上进行了复现，并通过实验验证了预取效果。

Sangam 的三层架构分别使用了三种不同组件，从而使得多样的数据访问场景能够得到满足。该预取器的第一级组件利用 IP 的增量进行预取，获得一个置信度较高的预测序列。第二级组件在 IP-增量预取器给出的预测序列出现置信度低的情况时启用，该级组件是一种基于 IP 步长的预取器。第三级预取器组件则是一个仅基于数据流的，具有自适应度的 next line 预取器。三级预取器主要集成在 L2 缓存中。此外，还通过优化预取所消耗资源的 L1 缓存查找带宽技术，以及在 L1 缓存控制器资源短缺的情况下保持 L1 缓存预取的激进的新机制，进一步增强了 L1 缓存预取器的预取效果。最终在 ChampSim 模拟器上的试验结果表明，使用了 Sangam 预取策略后，IPC 指标取得了显著改善。

参考文献

[1] The 3nd Data Prefetching Championship (DPC3).http://comparch-conf.gatech.edu/dpc3/.

[2] J. Kim, et al. Path Confidence based Lookahead Prefetching. In MICRO 2016.

[3] S. H. Pugsley, et al. Sandbox Prefetching: Safe Run-time Evaluation of Aggressive Prefetchers. In HPCA 2014, pages 626–637.

[4] P. Michaud. Best-offset Hardware Prefetching. In HPCA 2016, pages 469–480.

[5] M. Shevgoor, et al. Efficiently Prefetching Complex Address Patterns. In MICRO 2015, pages 141-152.

[6] M. Ferdman and B. Falsafi. Last-Touch Correlated Data Streaming. In ISPASS 2007, pages 105–115.

[7] L. M. AlBarakat, P. V. Gratz, D. A. Jimenez. MTB-Fetch: Multithreading Aware Hardware Prefetching for Chip Multiprocessors.

[8] A. Jain and C. Lin. Linearizing Irregular Memory Accesses for Improved Correlated Prefetching. In MICRO 2013, pages 247–259.

[9] A. Nori, et al. Criticality Aware Tiered Cache Hierarchy: A Fundamental Relook at Multi-Level Cache Hierarchies. In ISCA 2018, pages 96–109.

[10] C.-J. Wu, et al. PACMan: Prefetch-aware Cache Management for High Performance Caching. In MICRO 2011, pages 442–453.

[11] In Computer Architecture Letters, 17(2): 175-178 (2018).

[12] Chaudhuri, M., & Deshmukh, N. (2019). Sangam: A multi-component core cache prefetcher. 3rd Data Prefetching ChampionshIP.

[13] A. Mirhosseini, A. Sriraman, and T. F. Wenisch, “Enhancing Server Efficiency in the Face of Killer Microseconds,” Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA), 2019.

[14] M. Hashemi, E. Ebrahimi, O. Mutlu, and Y. N. Patt, “Accelerating Dependent Cache Misses with an Enhanced Memory Controller,” in Proceedings of the International Symposium on Computer Architecture (ISCA), pp. 444–455, 2016.

[15] H. Kang and J. L. Wong, “To Hardware Prefetch or Not to Prefetch?: A Virtualized Environment Study and Core Binding Approach,” in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems

(ASPLOS), pp. 357–368, 2013.

[16] Bhatia E, Chacon G, Pugsley S H, et al. “Perceptron-based prefetch filtering,” in Proceedings of the International Symposium on Computer Architecture (ISCA), pp. 1-13, 2019.

[17] S. Kumar and C. Wilkerson, “Exploiting Spatial Locality in Data Caches Using Spatial Footprints,” in Proceedings of the International Symposium on Computer Architecture (ISCA), pp. 357–368, 1998.

[18] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, “Spatial Memory Streaming,” in Proceedings of the International Symposium on Computer Architecture (ISCA), pp. 252–263, 2006.

[19] Y. Solihin, J. Lee, and J. Torrellas, “Using a User-Level Memory Thread for Correlation Prefetching,” in Proceedings of the International Symposium on Computer Architecture (ISCA), pp. 171–182, 2002.

[20] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos, “Accurate and Complexity Effective Spatial Pattern Prediction,” in Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA), pp. 276–287, 2004.

[21] Y. Ishii, M. Inaba, and K. Hiraki, “Access Map Pattern Matching for Data Cache Prefetch,” in Proceedings of the International Conference on Supercomputing (ICS), pp. 499–500, 2009.

[22] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, “Efficiently Prefetching Complex Address Patterns,” in Proceedings of the International Symposium on Microarchitecture (MICRO), pp. 141–152, 2015.

[23] M. Hashemi, O. Mutlu, and Y. N. Patt, “Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads,” in Proceedings of the International Symposium on Microarchitecture (MICRO), pp. 61:1–61:12, 2016.

[24] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, “Scale-Out Processors,” in Proceedings of the International Symposium on Computer Architecture (ISCA), pp. 500–511, 2012.

[25] W. A. Wulf and S. A. McKee, “Hitting the memory wall: Implications of the obvious,” SIGARCH Comput. Archit. News, vol. 23, pp. 20–24, Mar. 1995.