先进数据预取算法的洞察报告

【摘要】数据预取，即预测应用程序未来的内存访问并获取那些远离片上缓存中的数据，是一种众所周知且广泛使用的方法，用于隐藏内存访问的过长的延迟从而使得减少内存访问的瓶颈。如今，几乎每个高性能处理器都集成了一些数据预取器，以捕捉应用程序的各种访问模式；在本洞察报告中，我们讨论了数据预取的基本概念，并研究了最先进的硬件数据预取方法。

一、介绍

不论是个人电脑、手机，还是服务器工作负载，大规模的数据流动比如媒体流和网络搜索服务数百万用户，被视为重要的应用程序类别。此类工作负载运行在大规模数据中心基础设施上，依赖于为低延迟和服务质量保证而调整的处理器。这些处理器通常包括少量高时钟频率、积极推测和深度流水线的核心，以尽可能快速地运行服务器应用程序，满足最终用户的延迟要求。

而令处理器设计师苦恼的是，内存系统的瓶颈阻止了服务器处理器在服务器应用程序上获得高性能。由于服务器工作负载处理大量数据，它们产生的活跃内存工作集远远超过服务器处理器片上缓存的容量，驻留在片外内存中；因此，这些应用程序频繁地错过片上缓存中的数据，必须访问长延迟内存来检索数据。这种频繁的数据丢失使得服务器处理器无法达到其峰值性能，因为核心在等待数据到达时是闲置的。[1]

系统架构师提出了各种策略来克服频繁内存访问的性能损失。数据预取是这些策略之一，已显示出显著的性能潜力。数据预取是预测未来内存访问并在核心明确请求之前获取那些不在缓存中的数据的艺术，以隐藏内存访问的长延迟。如今，几乎每个高性能计算芯片都使用一些数据预取器来捕捉各种应用程序的规则和/或不规则内存访问模式。同样，研究文献中也有大量关于数据预取的提案，每个提案都以特定方式提高预取效率。

二、背景

在过去的几十年中，技术制造的进步伴随着电路级和微架构的改进，极大地提升了处理器的性能。同时，内存系统的性能却没有跟上处理器的速度，导致处理器与内存系统之间形

[1] A Survey on Recent Hardware Data Prefetching Approaches with An Emphasis on Servers

成了巨大的性能差距。因此，提出了许多方法来通过弥合处理器和内存性能差距来提高应用程序的执行性能。硬件数据预取就是这些方法之一。硬件数据预取通过主动在核心请求之前获取数据来弥合差距，以消除处理器等待内存系统响应的空闲周期。在本节中，我们简要回顾了硬件数据预取的背景。

2.1 预取缓存技术介绍

预取缓存（Prefetching Cache）技术是一种用于提高计算机系统性能的关键技术。它通过在数据实际被请求之前将其提前加载到缓存中，从而减少数据访问的延迟时间，提升系统的整体效率。预取缓存技术主要应用于处理器、存储器和网络系统中，以提高数据访问的速度和响应时间。

2.1.1 预取缓存技术的原理

预取缓存技术基于以下两个核心原理：

空间局部性（Spatial Locality）：指数据访问在空间上具有聚集性，即如果某个地址被访问，那么相邻地址也很可能在不久后被访问。预取缓存可以利用这一特点，将相邻的数据块提前加载到缓存中。

时间局部性（Temporal Locality）：指数据访问在时间上具有聚集性，即如果某个地址被访问，那么该地址很可能在不久后再次被访问。预取缓存可以利用这一特点，将频繁访问的数据块保留在缓存中。

2.1.2 预取缓存技术的类型

1. **Stride and stream prefetcher**：针对缓存行访问步幅进行预取的机制，通过记录地址和间距实现，尤其适用于数组和密集矩阵访问。具体实现时，需要动态调整置信度以避免错误预取；还可转而记录相同 PC 访存指令地址间距，以适用于矩阵向量乘法和跨步幅访问序列。
2. **Address-correlated prefetching**：通过记录重复访问序列和发现模式进行预取，特别适用于指针数据；典型实现是马尔科夫预取器，通过马尔科夫模型分析程序内存访问模式，基于当前状态和历史状态概率预测下一次访问的数据，并提前加载这些数据到高速缓存以降低访问延迟；还包括其他实现如距离预取，使用未命中地址间的距离作索引。
3. **Spatially correlated prefetching**：针对常规数据结构设计相关预取策略，利用数据访问的规律性和重复性，尤其对面向对象编程中的对象、数据库中固定大小字段等规律数据块效果显著。全局历史缓冲区的局部 PC 增量相关变体在存储效率和覆盖范围上十分有

效，可通过在索引表存储 GHB 指针实现索引，解耦历史信息存储，扩大预取深度和宽度，有效解决马尔科夫预取器深度与存储效率间的矛盾。

4. **Execution-based prefetching**：在 CPU 乱序阶段之前，利用 CPU 的闲置周期和硬件资源识别和预取数据，不倚赖内存地址访问模式。[1]

玄铁 C910 实现的数据预取机制属于上面的第一类，包括全局预取和多流预取两种模式，前者适用于连续一致的数据流，后者适用于对同一条指令的多次执行。

2.1.3 预取缓存技术的优点

减少数据访问延迟：通过提前加载数据，预取缓存技术可以显著减少数据访问的延迟时间，提升系统的响应速度。

提高系统吞吐量：预取缓存技术可以减轻主存储器的负担，减少内存访问次数，从而提高系统的整体吞吐量。

优化资源利用：通过减少数据传输的等待时间，预取缓存技术可以优化计算资源和存储资源的利用，提高系统效率。

2.2 数据预取器的位置

预取可以用于将数据从内存层次结构的较低级别移动到任何较高级别。先前的工作在所有缓存级别使用了数据预取器，从主数据缓存到共享的最后一级缓存。

数据预取器的位置对其整体行为有深远的影响。一级缓存中的预取器可以观察到所有内存访问，因此能够发出高度准确的预取请求，但代价是为记录元数据信息施加了大量存储开销。相比之下，最后一级缓存中的预取器观察到的是在内存层次结构的较高层次过滤后的访问序列，导致预测准确性较低，但存储效率较高。

2.3 预取危害

对数据预取器的简单部署不仅可能不会提高系统性能，还可能显著损害性能和能源效率。数据预取的两个众所周知的主要缺点是：(1) 缓存污染和 (2) 片外带宽开销。

缓存污染：数据预取可能通过用无用的预取数据替换有用的缓存块来增加需求丢失，从而损害性能。当一个激进的预取器表现出低准确性和/或在多核处理器中一个核心的预取请求与其他核心的需求访问竞争共享资源时，缓存污染通常会发生。

---

[1] Mittal S (2016) A survey of recent prefetching techniques for processor caches. ACM Comput Surv CSUR 49(2):1–35. https://doi.org/10.1145/2907071

带宽开销：在多核处理器中，一个核心的预取请求可能会由于争夺内存带宽而延迟另一个核心的需求请求。这种干扰是多核处理器中使用数据预取器的主要障碍，随着核心数量的增加，这个问题变得更加棘手。

三、先进的硬件预取器（SOTA）

3.1 Bingo 空间数据预取器[1]

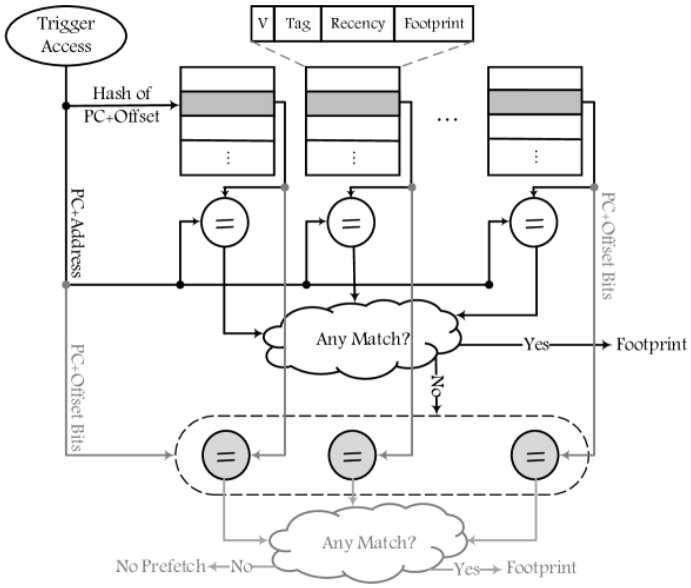
TAGE 使用多个具有不同信息的历史表来预测未来的事件。这些历史表存储了长短不同的事件历史，并根据这些历史来做出预测。Bingo 借鉴了这一策略，将观察到的访问模式与多个事件（包括长短事件）关联，这样在预取时可以考虑到更多的上下文信息，从而提高预测的准确性和覆盖率。

TAGE 在需要进行预测时，会依次检查多个历史表，从最长的历史表开始（最准确的但最少发生的），如果无法做出预测，则切换到下一个较长的历史表。Bingo 在预取时也采用了类似的逻辑，首先使用最长的事件来查找匹配的历史记录，如果没有找到，则使用次长事件继续查找。

但是 Bingo 注意到在传统的 TAGE 预测器中，多个历史表之间存在大量冗余信息，即多个表可能为相同的预测提供相同的元数据。为了避免为每种事件类型分别维护不同的历史表从而有效消除这种冗余，Bingo 提出了一个统一的历史表，该表通过不同的索引和标签机制实现了每个元数据项只存储一次，但是可以用于多个事件的预测。

[1] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran and H. Sarbazi-Azad, "Bingo Spatial Data Prefetcher," 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, DC, USA, 2019, pp. 399-411, doi: 10.1109/HPCA.2019.00053.

上一段中奖道德索引机制和标签机制具体如下：



图表 1 BINGO 的架构实现

索引与标签:历史表通过使用最短事件的哈希值来索引,这样可以用一个较短的事件(例如，仅基于偏移量 Offset）来查找表项。然而，表项是通过最长事件的所有位来标记的（例如，PC+Address)。这意味着每个表项都与一个特定的长事件相关联。

存储新足迹：当需要存储新的足迹时，它与相应的长事件（PC+Address）关联，并使用短事件(PC+Offset)的哈希值来确定在历史表中存储该信息的位置。使用替换算法(如 LRU）来选择要替换的条目。

查找匹配项：在需要进行预测时，首先使用长事件（PC+Address）来查找历史表。如果找到匹配项，则使用对应的足迹元数据来发出预取请求。如果没有找到匹配项，则使用短事件（PC+Offset）来查找表。

冗余消除：由于长事件包含了短事件的信息（PC+Address 中自然包含了 PC+Offset），因此不需要单独存储与短事件关联的元数据，这样就自动消除了冗余。

多匹配处理：如果在查找短事件时找到多个匹配项，Bingo 采用启发式方法来决定基于哪个足迹发出预取请求。例如，可以选择基于最近足迹的信息，或者为所有匹配项足迹中至少 20%指示的块发出预取请求。

论文中通过一系列大数据应用的详细评估表明，Bingo 在没有数据预取器的基线系统上平均提高了 60%的性能，并且在某些情况下，性能提升高达 285%。

与之前表现最好的空间数据预取器相比，Bingo 平均提高了 11%的性能。

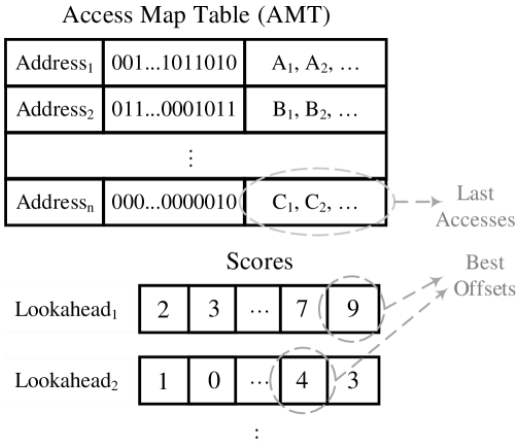
3.2 多前瞻偏移预取器（MLOP）[1]

多前瞻偏移预取器（MLOP）是最先进的偏移预取器。偏移预取实际上是步进预取（第1.3 节）的演变，其中预取器不尝试检测步进流。相反，每当核心请求缓存块（例如，A）时，偏移预取器会预取与该块相距k个缓存行的缓存块（例如，A+k），其中k是预取偏移量。换句话说，偏移预取器不会将访问地址与任何特定流相关联；相反，它们单独处理地址，并根据预取偏移量为每个访问的地址发出预取请求。偏移预取器已被证明在提供显著性能提升的同时施加较小的存储和逻辑开销。

最初的偏移预取提案名为 Sandbox 预取器（SP），尝试找到能产生准确预取请求的偏移量。为了找到这样的偏移量，SP 评估了几个预定义偏移量（例如，-8，-7，…，+8）的预取准确性，并最终允许预取准确性超过某一阈值的偏移量发出实际预取请求。后来的工作名为最佳偏移预取器（BOP），调整了 SP 并将及时性设为评估标准。BOP 基于这样的见解，即准确但延迟的预取请求不会像及时请求那样加速应用程序的执行。因此，BOP 寻找能产生及时预取请求的偏移量，以使预取块在处理器实际请求它们之前准备就绪。

MLOP 进一步提出了一种新的偏移预取器。MLOP 基于这样的观察，即虽然 BOP 能够生成及时的预取请求，但它由于依赖单一的最佳偏移而丧失了覆盖缓存丢失的许多机会。BOP 评估了几个偏移量，并认为能生成最多及时预取请求的偏移量是最佳偏移；然后，它仅依赖这个最佳偏移来发出预取请求，直到另一个偏移量变得更好，从而成为新的最佳偏移。事实上，这是一种二元分类：预取偏移量被认为是及时的或延迟的。分类后，预取器不允许所谓的延迟偏移量发出任何预取请求。然而，可能有许多其他合适的偏移量，它们虽然不那么及时，但具有隐藏大量缓存丢失延迟的价值。

为了克服先前工作的缺陷，MLOP 提出在评估各种预取偏移量期间，不是二元分类它们，而是有一个各种预取偏移量的及时性光谱。在评估各种预取偏移量期间，MLOP 考虑每个预



图表 2 MLOP **的架构实现**

[1] M. Shakerinava, M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, “Multi-Lookahead Offset Prefetching,” The Third Data Prefetching Championship, 2019.

取偏移量的多个前瞻：每个偏移量能以哪个前瞻覆盖特定的缓存丢失？为了实现这一点，MLOP 为每个偏移量考虑了多个前瞻，并分别为每个偏移量的每个前瞻分配分数值。最后，当预取时间到来时，MLOP 为每个前瞻找到最佳偏移，并允许其发出预取请求；然而，优先考虑并首先发出较小前瞻的预取请求。通过这样做，MLOP 确保预取器能够发出足够的预取请求（即，利用各种预取偏移；高丢失覆盖率），同时考虑到及时性（即，预取请求是有序的）。

图 2 展示了 MLOP 的概述。为了从访问模式中提取偏移量，MLOP 使用访问地图表（AMT）。AMT 跟踪几个最近访问的地址，并为每个基地址配备一个位向量。位向量中的每个位对应于该地址附近的一个缓存块，指示该块是否已被访问。

MLOP 考虑一个评估期，在此期间评估几个预取偏移量，并选择合格的偏移量以在以后发出预取请求。对于每个偏移量，它考虑多个级别的分数，其中每个级别对应于特定的前瞻级别。偏移量在前瞻级别 X 的分数表示偏移预取器能够至少提前 X 次访问预取访问的情况数。例如，偏移量在前瞻级别 1 的分数表示偏移预取器能够预取未来访问的情况数。

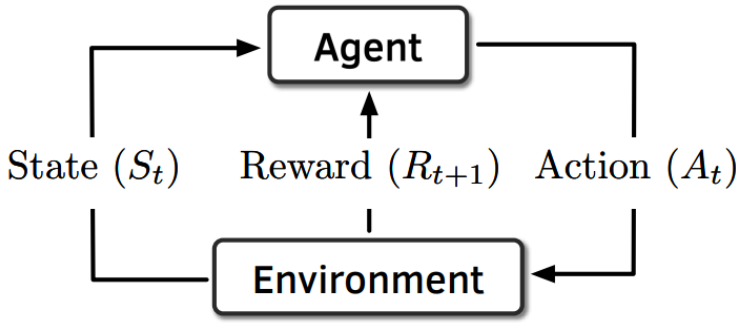
为了有效缓解所有可预测缓存丢失的负面影响，MLOP 从每个前瞻级别中选择一个最佳偏移量。然后，在实际预取期间，它允许所有选择的最佳偏移量发出预取请求。这样，MLOP 确保选择足够的预取偏移量（即，不像先前的工作那样抑制许多合格的偏移量），并将覆盖可预测的缓存丢失的很大一部分。为了解决及时性问题，MLOP 尝试以应用程序没有预取器的方式发送预取请求：MLOP 从前瞻级别 1 开始（即，预计最早发生的访问）并使用其最佳偏移发出相应的预取请求，然后转到上一级；这个过程重复进行。通过这种优先级排序，MLOP 尽可能隐藏所有可预测缓存丢失的延迟。

3.3 Pythia 算法[1]

Pythia 是一种硬件预取框架，通过利用在线强化学习（RL）来进行预取决策。传统的预取技术通常依赖单一类型的程序上下文信息（如程序计数器、缓存行地址或缓存行地址之间的差值）来预测未来的内存访问。这些技术往往忽视了预取器对系统整体（如内存带宽使用）的不利影响，导致在不同工作负载和系统配置下性能不稳定。Pythia 旨在设计一个全面的预取算法，利用多种类型的程序上下文和系统级反馈信息进行预取。

[1] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. 2021. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21).

Pythia 将预取器设计为一个强化学习代理。对于每个需求请求，Pythia 观察多个不同类型的程序上下文信息来做出预取决策。每个预取决策都会获得一个数值奖励，该奖励根据当前内存带宽使用情况来评估预取质量。Pythia 利用这个奖励来加强程序上下文信息与预取决策之间的关联，从而在未来生成高精度、及时且系统感知的预取请求。



图表 3 Pythia **的奖励机制图示**

Pythia 作为一种基于强化学习的硬件预取框架，其硬件构成主要包括 Q 值存储（QVStore）和评估队列（EQ）这两个部分。QVStore 用于记录所有状态-动作对的 Q 值，而 EQ 用于跟踪最近的预取动作及其对应的预取地址。通过这两个组件的协同工作，Pythia 实现了高效、准确的预取决策。

QVStore 是 Pythia 的核心组件之一，它记录了所有状态-动作对的 Q 值。为了提高存储效率和查找速度，QVStore 采用分层组织结构。具体来说，QVStore 分为多个 Vault，每个 Vault 对应一个程序特征，记录该特征和动作对的 Q 值。例如，一个 Vault 可能专门记录程序计数器（PC）与动作对的 Q 值，而另一个 Vault 则记录缓存行地址与动作对的 Q 值。每个 Vault 进一步分为多个 Plane，Plane 中的每个条目存储部分 Q 值。这种设计借鉴了 Tile Coding 的思想，通过多次量化特征空间来平衡特征值的分辨率和泛化能力。

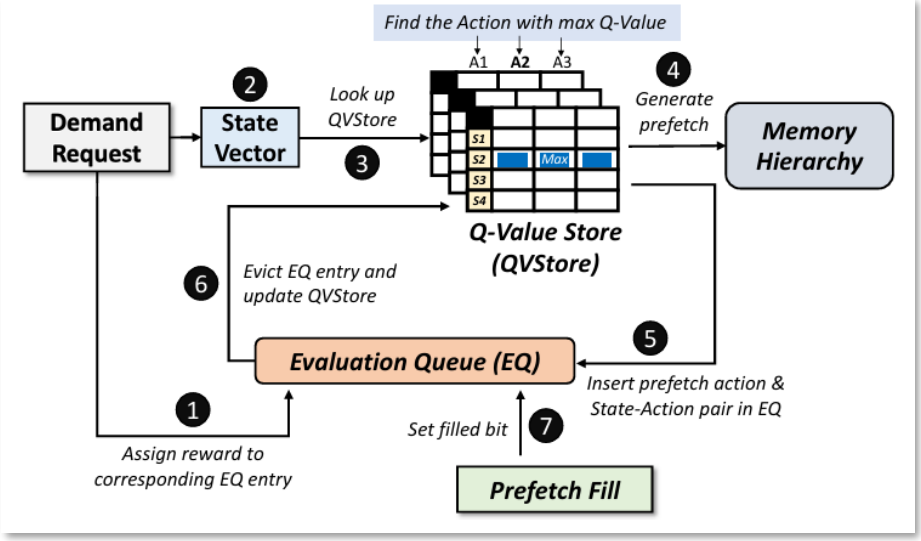
EQ 是一个先进先出的队列，用于跟踪最近的预取动作及其对应的预取地址。每个预取动作在生成后都会被插入到 EQ 中，EQ 条目包括预取动作、对应的预取地址、状态向量和填充位（filled bit）。在 EQ 条目被创建或在队列中时，Pythia 根据预取动作的效果分配相应的奖励。

在 EQ 条目被逐出时，Pythia 使用 SARSA 算法更新 Q 值。这个过程包括提取被逐出条目的状态和动作对，计算当前状态-动作对的 Q 值，并根据奖励和下一状态的 Q 值更新当前状态-动作对的 Q 值。

为了实现上述功能，Pythia 的硬件设计需要满足高吞吐量、低延迟和低功耗的要求。Pythia 采用多级流水线设计，以提高 Q 值检索和更新的效率。每个流水线级负责特定的计算步骤，如索引生成、Q 值检索、Q 值计算和最大值比较。此外，通过并行处理多个 Vault 和 Plane，Pythia 能够同时检索多个 Q 值，大大缩短了预测延迟。QVStore 和 EQ 需要一定的存储空间来记录 Q 值和动作信息。Pythia 的设计中，每个 Vault 的存储需求是固定的，

存储容量可以根据系统需求进行调整。

QVStore 存储需求:QVStore 需要 24KB 的存储空间,用于记录所有状态-动作对的 Q 值。QVStore 通过分层和 Plane 结构，实现了高效的存储和检索。EQ 存储需求：EQ 需要 1.5KB 的存储空间，用于记录最近的预取动作及其对应的预取地址。总体硬件开销：Pythia 的总



图表 4 Pythia 的硬件实现

体硬件开销仅为 25.5KB。这些存储需求在现代处理器的设计中是可以接受的。

此外,通过对 Pythia 的硬件设计进行实际的综合(synthesis)和验证(verification),发现其增加的芯片面积开销仅为 1.03%，而功耗开销仅为 0.37%。

Pythia 在多个实验中展现了优异的性能表现。与现有的最先进预取器 MLOP 和 Bingo 相比，Pythia 在单核、多核和带宽受限的系统配置下均表现出显著的性能提升。具体效果包括：在单核配置中，Pythia 比 MLOP 和 Bingo 分别高出 3.4%和 3.8%的性能。多核性能提升：在 12 核配置中，Pythia 比 MLOP 和 Bingo 分别高出 7.7%和 9.6%的性能。带宽受限系统中的性能提升：在带宽受限的系统中，Pythia 比 MLOP 和 Bingo 分别高出 16.9%和 20.2%的性能。

四、总结

数据预取技术在弥合处理器与内存系统性能差距方面展现了巨大的潜力。随着处理器性能的不断提升，内存系统成为性能瓶颈的问题愈加突出，数据预取作为一种有效的解决方案得到了广泛关注和深入研究。

先进的数据预取器如 Bingo、MLOP 和 Pythia，通过不同的策略和算法，实现了显著的性能提升。Bingo 预取器通过统一历史表和索引机制，有效提高了预取预测的准确性和覆盖率。MLOP 预取器通过考虑多个预取偏移量和前瞻级别，确保了预取请求的及时性和覆盖率，从而最大限度地减少缓存丢失的延迟。Pythia 预取器则利用在线强化学习（RL），结合多种程序上下文和系统级反馈信息，生成高精度、及时且系统感知的预取请求，大幅提升了处理

器在各种工作负载下的性能。

未来，随着计算需求的不断增加和应用场景的多样化，数据预取技术必将继续发展和演进。进一步优化预取算法，结合更多的上下文信息和系统反馈，开发更加智能和高效的数据预取器，将是未来研究的重要方向。同时，随着硬件设计和制造工艺的进步，数据预取器的实现也将更加高效和低功耗，为高性能计算提供强有力的支持。

总体而言，数据预取技术通过隐藏内存访问延迟，减少内存系统对处理器性能的制约，显著提升了计算系统的整体性能。未来的研究和发展将进一步挖掘其潜力，为更高效的计算架构奠定基础。

五、其他论文阅读心得

5.1 CSPM：

CSPM: A Coordinated Software Prefetching Mechanism For Multi-Level Caches

一种用于多级缓存的协调软件预取机制

（1）确定预取候选项

a. 事例分析

```
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
a[i][j]=a[i][j+1]+b[j][0]
```

对于数组 a[i][j]和 a[i][j+1]两个元素，预取 a[i][j+1]是没有必要的，因为在 j++之后的 a[i][j]与 j++之前的 a[i][j+1]表示的是同一个数据。

对于 b[j][0]来说，内环空间局部性好，外环时间局部性好。所以在每次大循环结束之后，它都会被“误导”，多进行一次错误的预取。

b. 由此，预取不一定会让性能变好，我们确定预取候选是十分有必要的。本文作者采用 T.C. Mowry 等人提出的技术确定预取候选。即仅当对数组的访问之间不存在局部性时，才会发出预取指令。局部性分析使用如下公式：（个人理解是综合考虑代码中数据的重复性以及本地资源的大小得出预取对象的确定）Data Locality=DataReuse∩Localized Iteration Space，简化，使用 FIFO。

（2）选择协调预取策略

采用根据接入流的数量来采用合适的策略。（解决预取候选项的存放问题）

具体策略：采用基本块模型，

block-insns / block-mem-refs > LxCACHE-INSNS-REFS-RATIO

block−mem−refs×α<LxCACHE−MSHR

其中 block_insns 是基本块中的指令数目，block_men_refs 是基本块中的访存次数。LxCACHE-INSNS-REFS-RATIO 是 cache 中计算和存储的最小并行度。第一个式子表征取数、计算这一过程在这层 cache 的承受范围之内。

其中α是一个优化系数(考虑到同一时间访存失效个数与此基本块个数存在偏差),LxCACHE-MSHR 是 cache 中包含的 MSHR 硬件资源，第二个式子表征 MSHR 资源可以供基本块开销。

当满足这两个条件时，预回写！

综上，达到存算两方面资源的合理分配。

（3）计算预取距离

执行预取指令和需求取取指令之间花费的时间称为预取距离。

a.计算循环中每次迭代的执行时间：

LIET 表示每次迭代时间，nrefs 表示访存的指令数目，〖Lat〗_L1 表示从 L1 访存的延时，Tc 表示执行计算指令的时间，Tb 表示执行分支指令的时间。

b.考虑到多级缓存，计算对应的延时，以及预取距离

〖PT〗_L1 表示 L1 层 cache 的延时，它就是访存 L1 和 L2 cache 时间之差。用这个延时除以 LIET 即是预取距离。

c.考虑到接入模式、MSHR 利用率、访问步幅、矢量化和循环展开

显然 CSPM 是一套有效的软件预取体系。

5.2 使用感知器学习的数据缓存预取

Data Cache Prefetching with Perceptron Learning

0、相关背景

感知器:

感知器（Perceptron Learning Algorithm）是一个二分类器，输入为特征空间，输出表示所属类别。感知机模型可以表示为:

f(x)=sign(w·x+b)

其中， 输入 x 是 n 维的特征向量，输出 y 是 1 或者-1，w,b 为感知机的模型参数。感知机模型时一种线性分类器，属于判别模型。

感知机模型的几何解释：线性方程 w·x+b =0 对应于特征空间 R^n 中的一个超平面 S，其中 w 是该平面的法向量，b 是超平面的截距。这个超平面将该空间分为两个部分，位于不同部分的实例属于不同的类别，位于相同部分的实例属于相同的类别。

内存污染:

2、主要思想

在上述文章中，作者提出了一个两级预取器的结构。第一级是之前基于表格的预取器与全局历史缓冲区（GHB）的结合，提供关于应该预取哪个缓存行的初步建议，同时将相关信息传递给下一级。在第二级中，感知器将利用这些定量信息来确定是否将预取请求发送到缓

存读取队列。

感知器的输入向量如图所示，这些特征包括：

预取距离（Prefetch distance）：预取距离指的是在数据被处理器需求访问之前，数据块已经被提前从内存预取到缓存中的距离。如果一个数据块的预取距离很高，意味着该数据块已经在缓存中待了很长时间，但却还没有被处理器使用。这种情况被称为缓存污染。

转移概率（Transition probability）：当发生缓存丢失时，新条目将被推送到 GHB 中。靠近上次缺失的缓存行的条目在未来内存访问中有很高的可能性被使用。靠得越近，可能性越大。使用马尔可夫图中使用的转移概率来量化这个特征。最后一次缺失的缓存行出现了 k 次。然后，在 GHB 时间序列中，每个条目都被赋予权重 2n-m/k，其中 m 表示该条目与最后一次缺失的缓存行之间的距离。如果预取器建议将 A 预取到缓存中，那么与 A 相关的所有权重之和将作为转移概率的表示。

块地址位异或程序计数器（Block address bits XOR program counter）：此指标显示了数据块的时间和空间距离。

出现频率（Occurrence frequency）：如果一个块经常出现在 GHB 中，这意味着处理器多次需要该块，但它被错误地从缓存中替换。因此，值越高，块应该被预取的可能性越高。

固定输入（Fixed input）：感知器需要一个阈值来做出判断。如果输出超过某个阈值，那么就确定预取该块，反之则不进行预取。然而，找到适用于所有情况的固定阈值是不可能的。因此，为了获得最佳性能，应该为不同的工作负载提供不同的阈值。文章的解决方案是提供一个额外的输入设置为 1，并相应地分配一个新的权重。这个新的权重可以作为阈值，同时适用于各种工作负载。

计算量化特征和权重乘积的总和作为 y_out。如果 y_out 超过 0，则决定预取该数据块。相反，如果 y_out 不超过 0，则决定不预取该数据块

预取机制数据路径如图所示。一旦发生缓存未命中，GHB 将推送一个新的条目。同时，预取器将被触发，并提供预取地址的建议。在 GHB 中进行搜索，将相关信息从 GHB 中提取，并作为输入传递给感知器。

经过量化后，感知器计算每个输入和权重的总和，得到 yout。

如果 yout 超过 0，将采纳此建议，并通过缓存读取队列将预取请求发送到下一级缓存或主存储器。同时，这一接受记录也将被推送到接受表中。否则，建议将被拒绝，并记录在拒绝表中。接受表和拒绝表中的记录都将作为感知器的训练数据，以便对可能的内存访问模式变化进行调整。

对于接受表中的条目，如果在 256 个缓存引用内没有被访问，则被视为错误预测。对于拒绝表中的条目，如果在 32 个缓存未命中（32 个预取触发）内没有被访问，则被视为正确的拒绝。这些信息将作为感知器在线训练的反馈。

3、总结

将感知器与传统的数据预取方式相结合是先进数据预取的方向之一。上述论文提出了一种新的两级预取方案，第一级采用基于表的预取工作，用于提供建议并提供必要的相关信息，第二级采用感知器学习工作，用于做出最终决策。与使用固定模式不同，感知器学习结合了局部和全局、时间和空间历史，可以动态检测和跟踪程序的内存访问模式。在 A Two Level Neural Approach Combining Off-Chip Prediction with Adaptive Prefetch Filtering 中，更进一步地提出了两级感知器的数据预期结构，说明了感知器在数据预取中广阔的应用前景。

5.3 签名路径预取器(Signature Path Prefetcher, SPP)

实现高准确率与覆盖率。

① SPP 使用基于**压缩历史**的方案，准确地预测复杂的地址模式。
签名表(ST)捕捉 4KB 物理页面内的内存访问模式，并将该页面内的先前增量压缩成一个 12bit 的历史签名。ST 跟踪最近访问的 256 个页面，并存储每个页面中最后访问的块，以计算当前内存访问的增量签名，然后使用该签名访问和更新模式表(PT)。每当有 L2 缓存访问时，其物理地址会传递给 ST，以找到相应物理页面的匹配条目。

② 与其他基于历史的算法不同，当地址模式在物理页面之间转换时，这些算法会错失许多预取机会，而 SPP 能**跨越物理页面边界**跟踪复杂模式，并在地址模式移动到新页面后继续预取。
当有一个超出当前页面 A 的预取请求时，传统的流预取器必须停止预取，因为预取器无法预测下一个物理页面号。但当首次访问下一个页面并发现该页面的第一个访问的页面偏移量与 SPP 先前预测的超出页面边界的偏移量匹配时，这种跨越边界的预测依然有用。为了跟踪这种行为，跨越边界的预测存储在一个小型的 8 项全局历史寄存器(GHR)中。GHR 存储当前签名、路径置信度、最后偏移量和用于跨页预取请求的增量。

③ SPP 根据对其预测的**置信度**，自适应地在每个预取流的基础上调节其预取行为。如果整体预取准确率较高，α将非常缓慢地降低路径置信度P_d，从而允许更深度的前瞻预取；如果全局预取准确率较低，α将迅速降低前瞻预取的强度。当预取器观察到路径置信度P_d或 L2 读取队列资源不足时，SPP 将停止预取。

实验结果：SPP 比无预取 baseline 提高了 27.2%的性能，比最先进的最佳偏移预取器性能高出 6.4%。SPP 以最小的开销实现了这一点，严格在物理地址空间中操作，而且未使用程序计数器或其他核心寄存器。

5.4 异构 Hadoop 集群的数据预取

Hadoop 集群：Hadoop 集群是一个由多台计算机组成的分布式计算环境，用于处理大规模数据。它基于 Apache Hadoop，是一个开源的、可扩展的框架，用于存储和处理大规模数据集。Hadoop 集群的工作原理是将大规模数据划分为多个块，并将每个块分配给集群中的不同节点进行处理。每个节点独立地处理分配给它的数据块，并将结果返回给主节点或其他节点进行进一步的处理或合并。这种分布式处理方式能够显著加快数据处理速度，并提供高可靠性和可伸缩性。

相比于传统 Hadoop 按顺序处理 Map 任务以实现数据传输和数据处理，异构 Hadoop 采用的数据预取使得数据传输和数据处理并行工作。在这种情况下，如果要进行数据跨节点的流动，则数据传输阶段会被隐藏在数据处理阶段中。如图节点 1 分配了一个 Map 任务 M，而由 Map 任务 M 处理的数据位于节点 2，因此它从节点 2 向节点 1 提取数据。

它创建一个预取线程来请求输入数据，并使用缓冲区来临时存储已获取的数据。线程从远程节点获取数据并临时存储在缓冲区中，Map 任务处理存储在缓冲区中的输入数据。通过这种形式，实现了数据传输和数据处理的并行操作

实验结果与分析：
论文作者采用 8 台虚拟机用于创建异构集群，采用词计数和 CCV Evaluator 等多种方式来评估数据预取性能，其中，单词计数计算单词出现的频率，CCV 找到分类算法中使用的类中心向量。同时，考虑到在理论计算中，预取机制的性能将受到数据处理和数据传输速度的影响，故在实验中人为引入了延迟。

对比上面的数据可以得出，modified Hadoop 确实对数据预取和处理有改进作用，并且这种作用的大小与某些参数大小有着密切关系，为了找出具体的关系，作者又做了两次实验

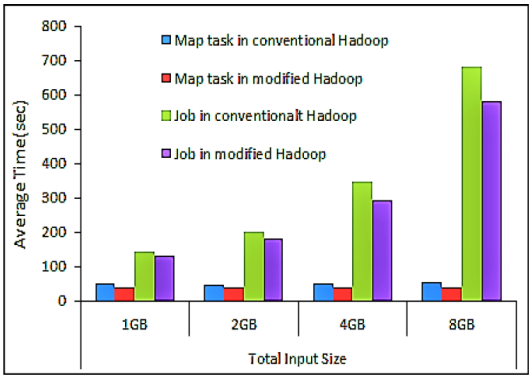


Fig 4: Performance of Hadoop for different input size

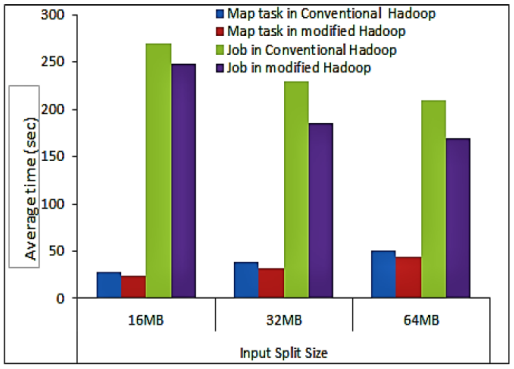


Fig 5: Performance of Hadoop for varied Input Split Size

从第一个实验可以看出，对于大于或等于 2 GB 的输入数据大小，作业执行时间平均减少了 15%。对于 1 GB 的输入数据，作业执行时间的改进较少。第二个实验说明显示了随着块大小的增加，性能提升也随之增加。对于 64 MB 的块大小，性能提升了 25%。这和我们前面的计算结果基本相符。

总结：相比于传统 Hadoop 按顺序处理 Map 任务以实现数据传输和数据处理，改进后的 Hadoop 采用了并行结构，数据被从远程节点预先获取到预取缓冲区，然后计算节点通过访问预取缓冲区中的数据来启动任务，将数据传输与数据处理阶段重叠。使得对于大于或等于 2GB 的输入数据大小，数据传输时间减少，作业执行时间减少 15%。

5.5 Berti-基于准确“local delta”的预取器

①概念解析：

IP（instruction pointer）：指令指针，和 PC 差不多，但区别在于它指向下一条待执行指令
Local delta v.s. stride：同一 IP 发出的两次请求访问之间的缓存行地址差 v.s.具有相同 IP 的连续加载访问的地址之间的差
Timely local delta：可以有效地提前预取的 delta
MSHR：相当于一个大小固定的数组，用于存放所请求数据还没返回到 L1 缓存中的 miss 请求。 Cache miss 的时候，首先会检查 MSHR 中是否存在相同的缓存请求，如果存在，则请求在 MSHR 中合并(succ hit)，不发送新的数据请求到下一级；如果不存在相同请求，则在

MSHR 和 cache 中为这条数据请求申请新的 entry。

Prediction：Generating requests

1.Delta 覆盖率+L1D 的 MSHR 占用率用以确定使用哪些增量+预取到哪个缓存级别
use four watermarks to decide where to issue the prefetch requests
1）Coverage>a high-coverage watermark
And MDHR<occupancy watermark
预取到的缓存级别：prefetch requests using that delta get filled at all the cache levels till L1D
2）coverage>medium-coverage watermark
预取到的缓存级别：Till L2
3)Coverage>low-coverage watermark
预取到的缓存级别：In LLC
2.生成预取请求
当前 addr+delta（都是虚拟地址），插入 PQ（预取队列，按 FIFO 执行），从 SLTB（ L2 转换先行缓冲区）中获取物理地址
如果翻译不了就丢弃；翻译了，request 会检测目标是否已在高速缓存中，若 miss，预取并将 request 插入 MSHR 中

③硬件实现

1.测量获取延迟
MSHR 和 PQ 扩展了 16 位的字段存放 miss/添加新 request 的 timestamp

2.学习 timely delta
历史条目表：
write（预取高速缓存行）Hit_p 或 L1D Miss
Search MSHR 的 fill 或 Hit_p：
MSHR 情况：使用来自 MSHR 信息进行搜索
Hit 情况：使用 stored latency 进行搜索

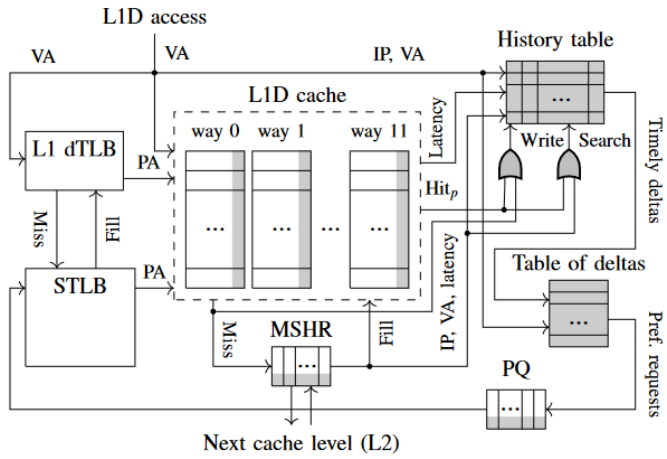


Figure 5. Berti design overview. Hardware extensions are shown in gray.

ppt 中疑问：dTLB：数据翻译缓冲器（Data Translation Lookaside Buffer，简称 DTLB），负责管理虚拟内存和物理内存之间的映射关系，用于存储虚拟地址到物理地址的映射表项。当程序访问虚拟地址时，CPU 首先查询 DTLB 以获取对应的物理地址，如果在 DTLB 中找到了对应的映射关系，那么就可以立即转换为物理地址并访问内存。如果在 DTLB 中没有找到对应的映射关系，就需要通过其他方式来查找映射关系，这将会引入额外的延迟。

3.计算 delta 覆盖率
Delta table：counter=16（溢出）时计算 coverage 大于高覆盖率 watermark 的增量将其状态设置为 L1D pre f 。
高覆盖率 wm 和中覆盖率 wm 之间的增量（65%和 35%之间）将状态设置为 L2 pre f 。
剩余增量的状态设置为 No pre f（即，不对该增量发出预取请求）。
一旦设置了状态，计数器和置信度数组就会重置，新的学习阶段开始。

3.计算 delta 覆盖率
在预热状态字段时，如果收集了至少 8 个增量，也会发出预取请求，从而将高覆盖率水印增加到 80%，因为只有 4 个增量，预取器需要更多的置信度。实证研究表明，使用高于 65%的水印可以带来较高的准确度。尽管 Berti 只为低覆盖率增量提供了预取到 LLC 的可能性，但评估表明选择此选项时没有性能改进。因此，将低覆盖率水印设置为 35%（等于中覆盖率水印），以仅禁用对 LLC 的预取。为了不断获取新的 delta，要抛掉老的 delta。当新 delta 获取时，前一阶段覆盖率低于 50% 的增量将被考虑在当前阶段驱逐。为此，如果选择 L2 pre f 状态时的覆盖率低于 50%，则将状态设置为 L2 pre f repl。驱逐策略选择覆盖率较低且状态为 L2 pre f repl 或 No pre f 的 delta。如果不存在这样的 delta，则丢弃新的 delta。

4.发出预取请求
在每次 L1D 访问中，都会使用匹配的 IP 来搜索 delta 表（IP、VA 箭头指向图 5 中的增量表）。
1）将状态为 L1D pre f 或 L2 pre f 的增量添加到当前 VA 以形成插入到 PQ 中的预取请求（图 5 中的首选请求箭头）。
2）当状态为 L1D pre f 并且 MSHR 占用率低于 70%（occupancy wm）时，这些预取请求将填充到 L1D 之前的所有缓存级别。
3）其余情况，预取请求将被填满直至 L2。

六、分工

杜宇航 22307130436 除 3.1 节外全文撰写
冯俊杰 22307130076 3.1 节
冯昭栋 22307130003 第五章部分
张一凡 22307130068 第五章部分
田慧楠 20307140073 第五章部分
王一鸣 22302010013 第五章部分
张乐然 22307130472 第五章部分

七、引用文献

[1] P. Michaud, "Best-offset hardware prefetching," 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), Barcelona, Spain, 2016, pp. 469-480, doi: 10.1109/HPCA.2016.7446087.

[2] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran and H. Sarbazi-Azad, "Bingo Spatial Data Prefetcher," 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, DC, USA, 2019, pp. 399-411, doi: 10.1109/HPCA.2019.00053.

[3] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. 2021. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21). Association for Computing Machinery, New York, NY, USA, 1121–1137. https://doi.org/10.1145/3466752.3480114

[4] Mittal S (2016) A survey of recent prefetching techniques for processor caches. ACM Comput Surv CSUR 49(2):1–35. https://doi.org/10.1145/2907071

[5] V. Jiménez, R. Gioiosa, F. J. Cazorla, A. Buyuktosunoglu, P. Bose and F. P. O'Connell, "Making data prefetch smarter: Adaptive prefetching on POWER7," 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), Minneapolis, MN, USA, 2012, pp. 137-146.

[6] D. Guttman, M. T. Kandemir, Meenakshi Arunachalam, and Vlad Calina. 2015. Performance and energy evaluation of data prefetching on intel Xeon Phi.(ISPASS)

[7]C. Chen et al., "Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension : Industrial Product," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)

[8] B. Falsafi and T. F. Wenisch. 2014. Primer on Hardware Prefetching. Morgan 8 Claypool.