小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems
DOI:10.20009/j.cnki.21-1106/TP.2025-0117
2026 年 4 月 第 4 期
Vol.47 No.4 2026

面向大规模推荐模型推理的 HBM-DRAM 嵌入向量存储系统

楼博涵,敖旭扬,王永福,李 京

(中国科学技术大学 计算机科学与技术学院,合肥 230022)
E-mail:lbh2001@mail.ustc.edu.cn

摘 要: 存储大规模推荐模型的嵌入向量特征需要大量的内存(DRAM),嵌入向量的高频查询和跨计算层的传输已成为推荐模型推理的性能瓶颈.GPU 的 HBM(High Bandwidth Memory)具备 TB/s 级高带宽,采用 HBM 来存储、访问嵌入向量可以显著提高推荐模型推理的性能,但 HBM 昂贵且容量有限,无法存放所有的嵌入向量.本文结合推荐场景中存在数据倾斜的特性,设计了一种混合存储系统,使用 HBM 作为一级存储存放热点嵌入向量加速推理,使用 DRAM 作为二级存储降低推理成本,实现了面向大规模推荐模型推理的嵌入向量存储系统.实验结果表明,相较于常见的纯 DRAM 方案,嵌入向量部分的吞吐率提升了 14 倍;相较于其他使用 GPU 来存储嵌入向量的实现,采用本系统实现的嵌入向量表,嵌入向量部分的吞吐率有 3.8 倍的提升.

关 键 词: 推荐系统;高性能计算;AI 大模型;GPU 加速
中图分类号: TP391 **文献标识码:** A **文 章 编 号:** 1000-1220(2026)04-0769-07

HBM-DRAM Embedding Storage System for Large-scale Recommendation Model Inference

LOU Bohan, AO Xuyang, WANG Yongfu, LI Jing
(School of Computer Science and Technology, University of Science and Technology, Hefei 230022, China)

Abstract: Storing the feature embedding vectors required by large recommendation models consumes considerable memory resources, and the frequent querying and cross-layer transmission of these vectors can become a bottleneck during inference. While GPU offers TB/s-level bandwidth, leveraging GPU memory for storing and accessing embedding vectors can improve inference performance, GPU memory is costly and has limited capacity, preventing it from holding all embedding vectors. This paper addresses the characteristic data skewness observed in recommendation scenarios to develop a hybrid storage system tailored for large-scale recommendation model inference. This system optimizes inference by harnessing the high bandwidth of GPU while reducing costs through the use of DRAM as secondary storage. The experimental results demonstrate that, compared to conventional implementations utilizing memory for storing embedding vectors, our system achieves a 14-fold increase in throughput for the embedding vector component. Furthermore, when compared to other implementations that employ GPU for embedding vector storage, our system's approach to managing the embedding table yields a 3.8x enhancement in throughput for the embedding vector component.

Keywords: recommendation system; High-Performance Computing (HPC); large-scale AI model; GPU acceleration

0 引 言

推荐系统提供了对目标用户的个性化推荐功能,在各大互联网服务中有着广泛的应用,例如商品推荐、广告点击率预测、短视频推荐等.传统推荐系统主要依赖于协同过滤[1]、基于内容的推荐[2]和混合方法[3].随着深度学习技术的发展,其在推荐系统中的应用[4-8]也逐渐成为研究热点.近年来,随着数据量的增加和特征维度的扩展,推荐系统中嵌入层参数规模庞大的问题愈发凸显.其中,大规模推荐模型的参数存储,推理过程中的 QPS(Queries Per Second)和延迟要求也越来越受到人们的关注.

传统的实现大规模推荐系统推理的方式是建立分布式 CPU 集群[9-12],使用 DRAM(Dynamic Random Access Memory)来存储推荐模型的参数.分布式集群带来了额外的网络开销和模型同步问题,使得模型更新较为困难;同时,使用 CPU 进行深度学习的推理计算速度较慢,降低了推理的吞吐率,使得企业需要更多的机器来满足用户请求,增加了成本.

HBM(High Bandwidth Memory)是 GPU 中使用的一种高带宽内存技术[13-15],它的特点是通过 3D 堆叠结构显著提升内存的带宽,降低延迟,特别适合需要大量数据传输速度的应用场景.使用 GPU 的 HBM 来存储推荐模型的参数,可以加快参数的访问;另外,利用 GPU 的并行计算的特性,可以更快地处理推荐模型中神经网络部分的计算.HBM 的容量不足以容纳全部的参数,考虑到推荐模型参数具有高度倾斜性,使用 HBM 作为一级存储,DRAM 作为二级存储,可以降低推理成本,提高推理吞吐率,减少推理延迟.

基于上述背景,本文提出并在开源项目 DeepRec[16] 的基础上实现了一个面向大规模推荐模型推理的混合存储系统 HS-Rec,具体贡献如下:

1)HBM-DRAM 混合存储结构的系统设计:设计 HBM-

收稿日期:2025-03-14 收修改稿日期:2025-04-14 基金项目:中国科学院战略性先导科技专项(A 类)项目(CCS54WHZ1123116926096472)资助. 作者简介:**楼博涵**,男,2001 年生,硕士研究生,研究方向为高性能计算;**敖旭扬**,男,2000 年生,硕士研究生,研究方向为高性能计算;**王永福**,男,1999 年生,博士研究生,研究方向为区块链、高性能计算;**李 京**,男,1966 年生,博士,教授,博士生导师,CCF 高级会员,研究方向为分布式系统、区块链、高性能计算.

DRAM 的多级存储方案以利用有限的 HBM 空间. 利用 HBM 的高带宽特性,加速频繁访问的数据读取;利用 DRAM 的高容量特性,存储大量不频繁访问的数据.

2)面向嵌入向量的 HBM 存储结构和高性能查询、替换算法:通过充分发挥 GPU 的并行性,利用其访存特性,设计嵌入向量查询算法,加速推理过程中嵌入向量的查询和拷贝. 观察到推荐系统的访问具有倾斜性、时序性和长尾效应,基于这些观察,设计适用于 GPU 的嵌入向量替换策略.

3)DRAM-HBM 数据传输和显存分配优化:结合 GPU 的访存特性,使用隐式访问优化 CPU 到 GPU 之间的数据传输,降低通信开销. 通过为每个线程建立显存分配池,采用合适的显存分配算法,减少多线程场景下的内存分配开销.

1 相关工作

1.1 基于深度学习的推荐模型

基于深度学习的推荐模型[4-8](Deep Learning Recommendation Models,DLRM)通过预测用户与特定内容进行互动的概率,即点击率(Click Through Rate,CTR),来解决推荐问题并提供个性化推荐. 为了生成这些预测,DLRM 接受代表用户和内容的两种类型特征作为输入:稠密特征和稀疏特征. 稠密特征代表连续的输入信息,例如用户的年龄、商品的价格,其由底层 MLP(Multilayer Perceptron)层处理;而稀疏特征则代表离散的、分类的数据,例如用户的居住地点、商品的种类,其作为输入传递给嵌入层. 嵌入层的功能是将这些离散的输入转换为稠密的、连续的嵌入向量(Embedding),以编码特征潜在的表示. 整体结构如图 1 所示.

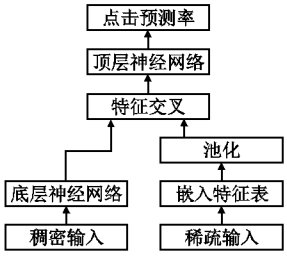

图 1 基于深度学习的推荐模型的结构图

Fig. 1 Structure of DLRM

推荐模型的参数一般具有以下几个特征:

1)参数量大:近年来存储推荐模型所需的参数量逐年增加,调研表明,存储嵌入向量所需内存占总推荐模型总内存比例可以高达 97%[9],推理时访存耗时可以占到总推理耗时的 79%[9].

2)高度数据倾斜性:嵌入数据的访问出现高度倾斜现象[17]. 统计显示,90% 的访问集中在仅占 1% 的热点数据上. 以在线购物系统为例,热门商品的访问频率远远超过其他商品.

3)时序性:嵌入热点数据的访问并不是一成不变的. 可能部分嵌入数据随着时间推移不再是热点,部分嵌入数据成为了新的热点. 这与现实生活中热点商品随着社会需求,广告舆论相变换也是对应的.

4)长尾效应:在推荐系统中,长尾效应[18-20]表明大部分商品或项目则只有少量的交互. 这些长尾商品通常是那些不太热门、较为小众或特定兴趣领域的商品. 这种现象对于推荐系统算法具有重要意义,因为它提示了系统不仅应该关注热门商品的推荐,也需要考虑到长尾商品的推荐. 处理长尾部分的访问对推荐系统的性能同样至关重要.

1.2 基于 HBM-DRAM 混合存储的推荐系统

近年来,为了提高推荐系统的推理性能,降低推理成本,越来越多的工作使用混合存储的方案来实现推荐模型的推理系统.

AIBox[21]被认为是混合存储的里程碑式成果. AIBox 采用了 SSD、DRAM、HBM 三级存储,将所有嵌入向量存储于 DRAM 和 SSD 中,以实现单机大规模推荐模型的存储. 同时,其会将查询到的嵌入向量传输至 GPU,利用 GPU 加速 MLP 层计算结果. AIBox 的不足之处在于它没有关注推荐模型数据的倾斜性,每次会全量的在 DRAM 到 HBM 间搬运嵌入数据,而嵌入数据的维度和数量都较大,造成了严重的数据传输开销. BagPipe[22]注意到了推荐模型数据存在的倾斜性质,并通过预读的方式来判断需要淘汰哪些数据,通过预取的方式来掩盖数据传输的开销. 上述两个推荐系统框架的嵌入表放在 DRAM 上,GPU 仅缓存嵌入向量的值,查询效率较低且数据传输开销较大.

HugeCTR[23]采用将查询嵌入表放入 GPU 的方法,在嵌入向量命中 HBM 中缓存的情况下,整个推理链路都在 GPU 上,完全消除了数据传输,增加了推理性能. FLECHE[24]在其基础上,考虑了更加细节的问题. 它注意到存在缓存命中率不足和内核调用开销较大的现象,采用将多个嵌入表合并的方法来动态调整各个特征列的嵌入表大小,并且通过核函数融合的方法来减少内核调用开销. 但是动态调整在推理场景下不适用,会带来频繁的显存分配和迁移开销. 两者的缺点是在缓存有少量不命中时,均采用返回默认值的方法以减少阻塞,影响了模型的推理精度. 而在推荐系统中,精度的微小损失会极大地影响企业效益.

1.3 冷热特征识别

推荐系统中的冷热特征识别问题涉及到如何识别和处理用户行为数据中的热门和冷门特征,以便提供更加个性化和高效的推荐服务. 这里,“热特征”通常指的是用户频繁交互的特征(如经常点击的商品、热门搜索关键词等),而“冷特征”则是用户很少交互的特征(如很少点击的商品、较少被搜索的关键词等). 通过识别热特征,推荐系统可以更准确地捕捉用户的兴趣和偏好,提供更相关的推荐内容. 对冷特征进行识别和处理,可以避免在推荐过程中浪费计算和存储资源.

常见的冷热特征识别方法包括统计分析、动态分析和特征重要性评估. 统计分析是冷热特征识别中最常见的方法,通过静态的分析访问序列的频率来划分冷热. 许多推荐模型[25]在训练中会采用这种方法进行预处理. 动态分析通过记录数据特征的访问频率或访问时间来区分数据的热度层级. 依据所采用的具体算法,将访问频率较低或最长时间未被访问的数据定义为冷数据,并据此对其进行相应的处理措施. CAFE[26]中使用了特征梯度的 L2 范数作为衡量特征重要性的标准. 在 Katharopoulos 等人[27]这篇文献中,作者通过实验和理论分析,证明了使用梯度的 L2 范数作为特征重要性衡量

标准的有效性.

在大规模推荐模型推理场景下,统计分析具有滞后性,无法反映热度的实时变化;推理时不计算梯度,无法使用其 L2 范数来评估特征重要性;本文最终采用了动态分析的方法来进行冷热特征识别.

2　实现原理

2.1　系统结构

在推荐系统中,访问嵌入向量是访存密集型的操作,并且这种访问模式通常表现出高度的数据倾斜性. 基于此,本文采用 HBM-DRAM 混合存储来存储嵌入向量,实现高性能的 GPU 查询、替换嵌入向量算法以加速访问嵌入向量的过程.

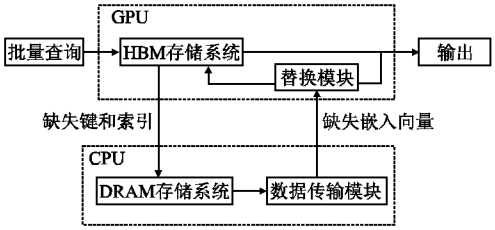

图 2　系统结构图

Fig. 2　System architecture

如图 2 所示,系统主要由 4 个部分组成:

1)HBM 存储系统:包含一个 GPU 嵌入向量表作为存放热嵌入向量的主要数据结构. 此表利用高效的并行查询算法,能够快速响应查询请求并返回相应的嵌入向量.

2)DRAM 存储系统:包含一个多线程无锁的 CPU 嵌入向量表,用于存储完整的嵌入向量. 当遇到 HBM 中未找到的数据时,会通过这个表进行多线程查询以获取所需数据.

3)数据传输模块:负责合并缺失数据的请求以及将这些数据从 DRAM 拷贝到 HBM 中. 这一过程优化了数据移动效率,减少了延迟.

4)替换模块:用于决定是否需要替换 HBM 中的嵌入数据,并选择淘汰访问频率较低的数据. 由于 DRAM 中保存着全部的嵌入数据,因此被替换的数据无需写回,直接覆盖即可.

系统的整体工作流程如下:当一个批量查询嵌入向量的请求到达混合存储系统时,首先会访问 GPU 中的 HBM 存储系统,在 GPU 嵌入向量表中进行查询. 如果找到了所需的嵌入向量,则直接将其值输出给推荐模型的下一层;如果没有找到,则转向 DRAM 嵌入表进行进一步的查询. 查询结果经过数据传输模块整合后,通过隐式传输的方式异步拷贝至 GPU. 最后,由 GPU 中的替换模块判断是否需要更新嵌入向量表中的数据,以及决定对应的更新位置.

以上架构旨在确保高效的数据访问、低延迟的响应以及数据的一致性,以满足大规模推荐模型推理任务的要求.

2.2　GPU 嵌入向量表的数据结构

本文参照常见的 GPU 哈希表[28-31],针对推荐系统嵌入向量的查询和拷贝设计了一个 GPU 嵌入向量表.

GPU 的嵌入向量表由 3 层结构组成:槽、组和组集,如图 3 所示.

槽:槽是一个逻辑上的概念,其代表了存储 GPU 嵌入向量基本单元. 每个槽包含一个嵌入向量和对应的键,以及一个访问计数器. 三者在静态表中共享相同的索引,通过这一机制实现彼此之间的关联与对应.

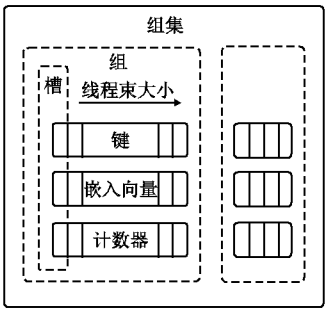

图 3　GPU 嵌入向量表的数据结构

Fig. 3　Data structure of GPU embedding table

组:现代 GPU 架构在以线程束(warp)为最小调度单位管理和执行代码. 编写支持线程束的程序可以实现最佳性能. 线程束的一般大小为 32,即一个线程束中包含 32 个 GPU 线程. 因此,本文将 32 个槽组合成一个组,以便每个线程束中的线程处理连续的显存中存储的内容. 同时,以线程束为单位,使用协作组(Cooperative Groups)可以实现最细粒度的 GPU 同步操作.

组集:为了充分利用 GPU 的大规模并行计算能力,减少哈希冲突的发生,类似于操作系统中的组相联策略,本文将若干个组并成组集. 每个待查询的键首先被映射到特定的组集,但后续可以占据该组集中的任意槽. 这样,查询时的线性探测被限制在单个组集内,不会与其他的的组集产生冲突. 较小的组集大小可以减少搜索延迟,但也会增加组集内部冲突的可能. 找到合适的组集大小来平衡这两个因素对查询性能来说至关重要. 本文将组集内的组数设置为 2.

2.3　GPU 嵌入向量表的查询和替换算法

GPU 嵌入表查询的基本思想是一个线程束的线程负责查询一个键对应的组集中的所有可能位置以保证访存连续性. 算法的具体过程如算法 1 所示. 嵌入向量以槽为基本单位存储于组集 S 中. 组集 S 可以看作是一个二维数组,第 1 个维度是组的个数,第 2 个维度是一个组中包含的槽的个数. 算法首先计算查询键 K 的哈希值以获得该键在表中存储组的组号. 然后一个线程束中的元素和该组中的元素一一对应进行查询. 查询结果通过线程束级别的同步操作进行传输,然后由该线程束内所有线程拷贝嵌入向量到输出.

以组集中包含 2 个组,组中包含 32 个槽为例子:查询时一个线程束中的 32 个线程访问对应的 32 个槽,如果其中某个键与查询的键相等,说明查询到了对应的嵌入向量,该线程会进行线程束内广播,然后这个线程束中的线程任务转变为拷贝该嵌入向量至输出,线程束中的线程会连续的访问嵌入向量进行拷贝;如果未找到则访问下一个组直至访问完所有对应的组. 如果在整个组集中均未找到对应的嵌入向量,则标记这个查询,等待整个批次的查询完成后访问 DRAM 中的 CPU 嵌入表进行查询.

算法 1. GPU 嵌入向量表的查询算法

输入:查询键 K; 嵌入向量表中的组集 S;GPU 线程号 ThreadID

输出:特征值 EV;缺失特征的索引 M

```
GroupID = -Hash(K) mod |S|
锁定 S[GroupID]
if K == S[GroupID][ThreadID].key
    广播至该线程组内的所有线程
End if
线程束级同步
if 收到广播
    拷贝嵌入向量 EV 到输出
else
    记录缺失的索引 M
End if
解锁 S[GroupID]
```

该查询算法主要有两大优点:1)无论是对键的查询还是对嵌入向量的拷贝,每个线程束的显存访问都是连续的,可以最大化利用 GPU 内部的缓存;2)线程束是 GPU 线程调度和同步的最小单位,在线程束内部进行广播、投票等同步操作开销较低,可以尽量减少同步开销.

GPU 嵌入表替换算法与查询的不同之处主要有概率准入和归约查找替换两点. 查找键对应的组集的过程与查询算法一致. 基于对数据集的观察和对推荐系统长尾效应的理解,本文制定了概率准入策略. 其中设定了一个概率准入阈值,当随机数小于这个阈值就不进行替换. 这一机制的目的是减少那些只访问一次的嵌入向量进入 HBM 缓存的概率,优化缓存的使用效率. 通过减少低频数据的准入,可以确保高频数据在缓存中的命中率更高,从而提升系统的整体性能. 当随机数超过阈值后,便会由该线程束在组集内进行并行归约查找该组集中计数器最小的槽. 然后由该线程束对槽内的嵌入向量进行替换更新.

算法 2. GPU 嵌入向量表的替换算法

输入:待替换键 K; 嵌入向量表中的组集 S;新的特征值 EV; 准入阈值 threshold

```
GroupID = -Hash(K) mod |S|
产生随机数 r
if r < threshold
    不替换,跳过该流程
else
    锁定 S[GroupID]
    并行归约查找该组集中计数器最小的槽 MinSlot
    拷贝 EV 至 S[GroupID][MinSlot]
    解锁 S[GroupID]
End if
```

2.4 DRAM 嵌入向量表的数据结构

DRAM 嵌入向量表存储全量的嵌入向量,用于查询在 HBM 中缺失的嵌入向量. 为了提高内存利用率并减少查找时间,本文采用了密集哈希表来存储全量的嵌入向量,采用多线程并行的方式进行查询加速.

密集哈希表由于采用了连续存储的方式,在推荐模型这种以读操作居多的场景下性能优秀. 如图 4 所示,密集哈希表通过一个状态位来标记占用和删除,简化了插入逻辑,提高了查询效率. 采用 K-V 分离的存储方式,使得该嵌入向量表可以支持不同大小不同类型的嵌入向量,提高了系统的通用性.

在多线程查询更新时,密集哈希表会有一些冲突情况需要处理. 本文通过使用读-复制-更新(RCU)的方式原子地更新表中元素来解决冲突. 以插入为例,当两个线程同时插入一个特征时,会通过 CAS 操作先对其状态进行原子地修改,再修改其值. 第 2 个线程发现其状态已经不为空后,会转而寻找下一个合适的位置进行插入.

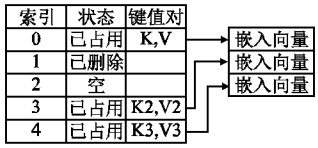

图 4 DRAM 嵌入向量表的数据结构

Fig. 4 Data structure of DRAM embedding table

2.5 DRAM-HBM 数据传输优化

为了快速地从 DRAM 到 HBM 之间迁移数据,本文采用了一系列诸如数据打包,使用页锁定内存(Pinned Memory),隐式访问等方式加快不同存储介质之间的数据迁移.

所有数据迁移操作都会打包以高效地将 CPU 上的数据发送到 GPU,减少接口调用的次数. 以更新缺失的嵌入向量为例,查找到的元素会将其值拷贝到一个连续的缓冲区中,然后 GPU 访问之.

页锁定内存指在物理内存中固定的、不会被移动或交换的内存. 在 CPU 和 GPU 之间的数据传输过程如图 5 所示,数据需要先从可分页内存拷贝到页锁定内存中,随后才能被发送给 GPU. 本文通过直接将嵌入向量存储在页锁定内存中来避免触发额外的复制操作,从而减少了数据传输的延迟. 同时,页锁定内存可以与异步数据传输结合使用,允许 GPU 在数据传输过程中进行其他计算操作,提高了整体的效率.

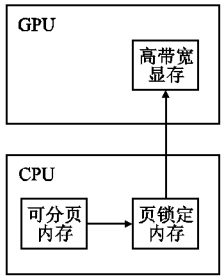

图 5 CPU-GPU 数据传输示意图

Fig. 5 CPU-GPU data transfer diagram

从 DRAM 到 HBM 的每次显式拷贝都需要发起一次 GPU 内核请求. 由于页锁定内存直接映射到 PCIe 总线上,GPU 可以直接访问并将其内容复制到输出缓冲区. 本文通过这种直接访问页锁定内存这种隐式访问的方式,减少了接口调用的开销.

3 实 验

3.1 实验环境

实验在基于 tensorflow 的 DeepRec 框架中进行搭建. 实验中使用的 GPU 为 Tesla T4,拥有 16G 显存. CUDA 的运行时环境为 11.6,PCIe 的最大传输速率为 8GT/s. 开发环境为 python3.7,tensorflow1.14,在 Linux 操作系统下进行对比实验.

3.2 数据集

为了评估本文提出的推理框架 HS-Rec 的性能,在 3 个数据集上进行实验:

MovieLens[32]:MovieLens 数据集是由 GroupLens 研究团队提供的电影推荐系统数据集,包含了大量用户对电影的评分、标签及时间戳等信息,广泛用于推荐系统算法的研究与开发.

Taobao[33]:发布在开源数据集网站 Kaggle 上的淘宝广告点击预测率数据集,其中包含了从淘宝网站随机抽取的百万名用户,他们 8 天的广告展示/点击日志和相关特征信息.

Ele. me[34]:由天池网站发布的大型推荐数据集,其中包含了 7 天饿了么软件的点餐数据,数据量大,特征列数目多.

表 1　实验数据集

Table 1　Datasets for evaluation

Dataset	T	N	S
MovieLens	4	100K	20M
CTR on Taobao	6	27M	2.4G
Ele. me	38	146M	80G

关于数据集的详细信息如表 1 所示,其中,T 代表数据集中嵌入向量表的个数,也是其中特征列的数量;N 代表数据条目数量;S 代表嵌入向量表的总大小,以字节为单位.

3.3 对比系统

为了验证本文提出的系统性能,选取了 3 个具有代表性的系统进行对比,反映了不同的嵌入向量查询和存储思路.

DeepRec[16]:这代表大部分工业界使用的推荐系统框架,嵌入向量的查询和存储都在 DRAM 上进行.

AIBox[21]:该系统嵌入向量查询和存储在 DRAM 上进行,拷贝在 HBM 上进行.

EV Store[35]:该系统嵌入向量查询在 DRAM 上进行,缓存和拷贝在 HBM 上进行.

HS-Rec:这是本文提出的系统,嵌入向量查询、存储拷贝均在 HBM 上进行,仅当缺失时访问 DRAM 进行拉取缺失的嵌入向量.

3.4 实验评估和参数设置

在实验中,如无特殊说明,本文将嵌入向量的维度设定为 128,这一选择与推荐系统领域内常用的嵌入向量维度标准相一致. 本文将 batch size 设置为 2048 以充分发挥 GPU 并行能力,嵌入向量的元素类型为 32 位浮点数. 对于本系统设计的 GPU 嵌入向量表,组集内组数设置为 2 以平衡查询延迟和吞吐率;组内槽数设置为 32,与一个线程束内线程数量保持一致. 当启用缓存淘汰概率准入策略时,准入概率设置为 50%.

本文与 Xie 等人[24]相同,采用推理时吞吐率来衡量系统性能,分为运行整个模型端到端的吞吐率和仅运行本文所优化的嵌入向量部分的吞吐率. 端到端模型采用 DLRM[3]. 针对本文提出的 GPU 嵌入向量表和概率准入替换策略,采用缓存命中率指标来衡量不同策略的优劣.

3.4 实验数据分析

3.4.1 系统吞吐率测试

本实验旨在评估不同系统在处理嵌入向量时的性能表现,具体涵盖嵌入向量部分的吞吐率及端到端的整体吞吐率. 通过在 3 个不同数据集上进行测试,该实验比较了各系统在多种应用场景和数据规模下的吞吐率,确保实验结果具备广泛的适用性和可靠性.

从表 2 的数据可以看出,本文提出的针对嵌入向量部分的优化取得了显著成效. 与 DeepRec 采用 CPU 查询并在 DRAM 上进行数据拷贝的方法相比,HS-Rec 通过利用 GPU 加速查询并使用 HBM 进行数据拷贝,使得嵌入向量部分的吞吐率提升了 14 倍,这与 HBM 和 DRAM 之间的访问速度差异相符. 相较于 AIBox,HS-Rec 在嵌入向量部分的吞吐率提高了 5 倍,主要原因是 AIBox 需要频繁地将嵌入向量数据传输至 GPU,而 HS-Rec 仅需传输缺失的数据,减少了不必要的数据移动. 相较于 EV Store,HS-Rec 的嵌入向量部分吞吐率提升了 3.8 倍,这是由于 HS-Rec 将嵌入向量查询任务也置于 GPU 上执行,从而充分利用了 GPU 的并行处理能力,同时在完全命中的情况下消除了 DRAM 到 HBM 之间的数据传输.

表 2　嵌入向量部分吞吐率(10^6 inference/sec)

Table 2　Embedding only throughput(10^6 inference/sec)

System	MovieLens	Taobao	Ele. me
DeepRec	1.86	1.77	1.63
AIBox	5.04	4.48	4.29
EV Store	5.27	5.53	5.11
HS-Rec	**34.36**	**39.47**	**24.82**

对比不同的数据集,可以看到本系统在中型数据集 Taobao 中表现最好. 这是因为在小型数据集 MovieLens 中无法充分发挥 GPU 的并行优势,启动内核带来的额外开销占比较大;在大型数据集 Ele. me 中,GPU 无法存放下所有的嵌入向量,使得需要从 DRAM 中拷贝缺失的嵌入向量数据至 HBM,降低了系统的吞吐率. 在中型数据集 Taobao 中,嵌入向量能全部缓存在 HBM 中,同时参数规模和访问规模也较大,能较好地利用 GPU 的计算和访存的性能优势加速.

表 3　端到端吞吐率(10^6 inference/sec)

Table 3　End-to-end throughput(10^6 inference/sec)

System	MovieLens	Taobao	Ele. me
DeepRec	1.31	1.27	1.26
AIBox	2.36	2.25	2.20
EV Store	2.41	2.65	2.39
HS-Rec	**3.92**	**3.91**	**3.82**

表 3 给出了运行端到端的 DLRM 模型的推理时吞吐率. 在 MovieLens 数据集上,HS-Rec 较 DeepRec 提升了 199%,较 AIBox 提升了 66%,较 EV Store 提升了 63%;在 Taobao 数据集上,HS-Rec 较 DeepRec 提升了 207%,较 AIBox 提升了 74%,较 EV Store 提升了 48%;在 Ele. me 数据集上,HS-Rec 较 DeepRec 提升了 203%,较 AIBox 提升了 74%,较 EV Store 提升了 60%. 相较于 DeepRec,模型吞吐率提升最为明显,是因为 DeepRec 完全使用 DRAM 来存储和处理嵌入向量,在模型中时间占比大,优化后效果显著;相较于 EV Store 吞吐率提升较为一般,是因为其已使用 HBM 来加速访存,从而减少了嵌入向量处理这一瓶颈,两者的加速来源于 GPU 嵌入向量表和 DRAM 嵌入向量表查询的性能差距. 对于本文优化后的推荐系统而言,嵌入向量的访问和拷贝已经不再是瓶颈所在,

对其进行进一步的优化收益在整体系统的吞吐率上表现不明显.

表4 不同批处理大小下吞吐率(10^6 inference/sec)

Table 4 Throughput under different batch size (10^6 inference/sec)

Batch size	MovieLens	Taobao	Ele. me
64	1.93	1.95	1.89
128	3.16	3.42	3.26
256	6.22	6.78	6.15
512	11.89	13.08	10.92
1024	22.04	24.64	17.97
2048	34.36	39.47	24.82
4096	**37.92**	**44.37**	**31.06**

为了展示本系统在处理大批次数据时的性能,本实验给出了不同批处理大小下本系统在不同数据集下的吞吐率. 如表4所示,在批处理大小较小时,由于GPU使用率不高,且此时系统固有开销较大,性能较差. 随着批处理大小的增加,本系统在所有数据集上的吞吐率都有提升. 如图6所示,随着批处理大小的增加,延迟有着较大提升,吞吐率的提升一般. 综合考虑在实际系统中请求到来的速度和延迟的需求,本文最终设定2048为使用的批处理大小.

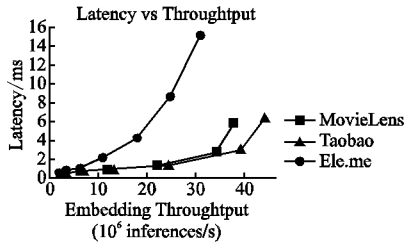

图6 吞吐率与延迟的权衡分析

Fig. 6 Throughtput-latency trade-off analysis

3.4.2 不同策略对命中率的影响

本实验使用Taobao数据集,限制其GPU能使用的大小,探究不同的GPU淘汰策略对命中率的影响. 本文在GPU上实现了常见的LRU(Least Recently Use)和LFU(Least Frequently Use)淘汰策略,以及对应的增加概率准入模块的版本LRU_PA和LFU_PA. 实验中的Cache Size代表GPU嵌入向量表中能存储的嵌入向量占总嵌入向量表的比例.

表5 不同策略的命中率

Table 5 Hit rate with different strategies

Cache Size	LRU/%	LFU/%	LRU_PA/%	LFU_PA/%
10	65.09	70.90	67.12	**71.95**
20	81.03	84.57	82.44	**84.77**
40	93.11	**94.25**	93.68	94.22
60	97.31	**97.57**	97.50	97.55
80	99.10	**99.13**	99.12	99.13
100	99.99	**99.99**	99.99	99.99

表5给出了不同嵌入向量表的替换策略在不同Cache Size下的命中率情况. 对于LRU算法而言,增加概率准入模块后,命中率都有所提升,在Cache Size较小的时候提升比较明显,有2.03%. 在Cache Size较大时命中率都较高,提升不明显. 对于LFU算法而言,在Cache Size较小时,命中率有1.05%的提升. 在Cache Size较大时,命中率略有下降. 概率准入策略在缓存大小较小时有效的核心原因在于他可以减少不必要的冷数据的缓存填充,尽量避免缓存被低效利用. 在缓存大小较大时,几乎所有常用的嵌入特征向量都可以被存储在缓存中,此时应用概率准入策略会导致数据访问的时间局部性没有得到充分利用. 考虑到实际应用场景中GPU嵌入向量表能缓存约10%到20%的嵌入向量,本系统最终应用了概率准入策略的LFU以提升命中率.

3.4.3 准入概率的单变量敏感性实验

本实验使用Taobao数据集,淘汰算法采取LFU,嵌入表大小设置为总嵌入大小的10%,对概率准入参数进行实验,实验结果如图7所示.

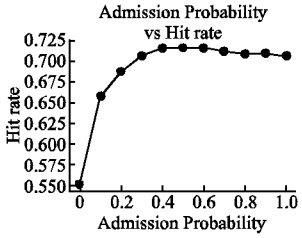

图7 概率准入参数对命中率的影响

Fig. 7 Effect of admission probability on hit rate

当准入概率较低时,该策略的效果等同于未实施任何淘汰机制,从而导致较低的命中率;而当准入概率较高时,对于仅访问一次的请求,其过滤效率则显得不足. 综合考虑实验结果,40% ~60%之间的准入概率是较为合理的参数区间. 基于此,本文最终确定将50%作为实际应用的准入概率值.

3.4.4 数据传输的性能分析

本文使用NVIDIA Nsight Compute工具观察在嵌入表中查询得到嵌入向量后,并行拷贝至MLP层的数据传输速率. 结果显示,此时HBM内传输带宽为281.9GB/s,Tesla T4的HBM理论带宽约为320GB/s,此时传输带宽达到了理论峰值的88%,表明本系统在显存带宽利用方面表现较为高效.

4 总 结

本文提出了一个面向大规模推荐模型推理的嵌入向量存储系统HS-Rec. 该系统使用GPU嵌入向量表来加速嵌入向量查询,HBM来加速嵌入向量拷贝,DRAM来存储全量的嵌入向量数据,提高了推荐系统嵌入向量部分的吞吐率,降低了单机单卡场景下的推理成本. 实验在3个数据集上进行,表明了多种场景和数据大小下本系统的有效性和优越性. 本文还通过探索不同的嵌入向量淘汰策略,通过应用概率准入策略提高了实际场景中的命中率. 未来的研究方向之一是将本系统应用到更广泛的硬件场景中,例如在多机多卡环境下的系统部署,以更好地满足工业界的需求. 此外,大规模语言模型中的键值缓存也呈现出类似的存储结构和访问模式. 因此,本系统的设计理念和优化策略未来可以扩展应用于大语言模型,提升此类模型的性能.

References:

[1] Sarwar B, Karypis G, Konstan J, et al. Item-based collaborative filtering recommendation algorithms[C]//Proceedings of the 10th International Conference on World Wide Web, 2001: 285-295.

[2] Lops P, Jannach D, Musto C, et al. Trends in content-based recommendation: preface to the special issue on recommender systems based on rich item descriptions[J]. User Modeling and User-Adapted Interaction, 2019, 29: 239-249.

[3] Burke R. Hybrid recommender systems: survey and experiments[J]. User Modeling and User-Adapted Interaction, 2002, 12(4): 331-370.

[4] Cheng H T, Koc L, Harmsen J, et al. Wide & deep learning for recommender systems[C]//Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, 2016: 7-10.

[5] Guo H, Tang R, Ye Y, et al. DeepFM: a factorization-machine based neural network for CTR prediction[J]. arXiv preprint arXiv: 1703.04247, 2017.

[6] Naumov M, Mudigere D, Shi H J M, et al. Deep learning recommendation model for personalization and recommendation systems[J]. arXiv preprint arXiv: 1906.00091, 2019.

[7] Zhou G, Zhu X, Song C, et al. Deep interest network for click-through rate prediction[C]//Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018: 1059-1068.

[8] Zhou G, Mou N, Fan Y, et al. Deep interest evolution network for click-through rate prediction[C]//Proceedings of the AAAI Conference on Artificial Intelligence, 2019: 5941-5948.

[9] Lui M, Yetim Y, Özkan Ö, et al. Understanding capacity-driven scale-out neural recommendation inference[C]//IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2021: 162-171.

[10] Sethi G, Bhattacharya P, Choudhary D, et al. FlexShard: flexible sharding for industry-scale sequence recommendation models[J]. arXiv preprint arXiv: 2301.02959, 2023.

[11] Hsia S, Gupta U, Acun B, et al. Mp-rec: hardware-software co-design to enable multi-path recommendation[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2023: 449-465.

[12] Zha D, Feng L, Bhushanam B, et al. Autoshard: automated embedding table sharding for recommender systems[C]//Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2022: 4461-4471.

[13] Kim J, Kim Y. HBM: memory solution for bandwidth-hungry processors[C]//IEEE Hot Chips 26 Symposium (HCS), 2014: 1-24.

[14] Jun H, Cho J, Lee K, et al. HBM (High Bandwidth Memory) DRAM technology and architecture [C]//IEEE International Memory Workshop (IMW), 2017: 1-4.

[15] Kim K, Park M. Present and future, challenges of High Bandwith Memory (HBM) [C]//IEEE International Memory Workshop (IMW), 2024: 1-4.

[16] Liu Tongxuan, Peng Tao, Chen Bangduo, et al. DeepRec [EB/OL]. https://github.com/DeepRec-AI/DeepRec, 2022-04-08.

[17] Lai F, Zhang W, Liu R, et al. AdaEmbed: adaptive embedding for Large-Scale recommendation models[C]//17th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2023: 817-831.

[18] Nagrecha K. Systems for parallel and distributed large-model deep learning training[J]. arXiv preprint arXiv: 2301.02691, 2023.

[19] Micikevicius P, Narang S, Alben J, et al. Mixed precision training [J]. arXiv preprint arXiv: 1710.03740, 2017.

[20] Cui H, Zhang H, Ganger G R, et al. Geeps: scalable deep learning on distributed gpus with a gpu-specialized parameter server[C]//Proceedings of the 11th European Conference on Computer Systems, 2016: 1-16.

[21] Zhao W, Zhang J, Xie D, et al. AIBox: CTR prediction model training on a single node[C]//Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019: 319-328.

[22] Agarwal S, Yan C, Zhang Z, et al. Bagpipe: accelerating deep recommendation model training[C]//Proceedings of the 29th Symposium on Operating Systems Principles, 2023: 348-363.

[23] Wei Y, Langer M, Yu F, et al. A GPU-specialized inference parameter server for large-scale deep recommendation models[C]//Proceedings of the 16th ACM Conference on Recommender Systems, 2022: 408-419.

[24] Xie M, Lu Y, Lin J, et al. Fleche: an efficient GPU embedding cache for personalized recommendations[C]//Proceedings of the 17th European Conference on Computer Systems, 2022: 402-416.

[25] Adnan M, Maboud Y E, Mahajan D, et al. Accelerating recommendation system training by leveraging popular choices[J]. arXiv preprint arXiv: 2103.00686, 2021.

[26] Zhang H, Liu Z, Chen B, et al. CAFE: towards compact, adaptive, and fast embedding for large-scale recommendation models[J]. Proceedings of the ACM on Management of Data, 2024, 2(1): 1-28.

[27] Katharopoulos A, Fleuret F. Not all samples are created equal: deep learning with importance sampling[C]//International Conference on Machine Learning, PMLR, 2018: 2525-2534.

[28] Alcantara D A, Sharf A, Abbasinejad F, et al. Real-time parallel hashing on the GPU[M]. New York: United States ACM SIGGRAPH, 2009: 1-9.

[29] Awad M A, Ashkiani S, Porumbescu S D, et al. Better GPU hash tables[J]. arXiv preprint arXiv: 2108.07232, 2021.

[30] Li Y, Zhu Q, Lyu Z, et al. Dycuckoo: dynamic hash tables on gpus [C]//IEEE 37th International Conference on Data Engineering (ICDE), 2021: 744-755.

[31] Ashkiani S, Farach Colton M, Owens J D. A dynamic hash table for the GPU[C]//IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2018: 419-429.

[32] Harper F M, Konstan J A. The MovieLens datasets[J/OL]. ACM Transactions on Interactive Intelligent Systems, 2016: 1-19, http://www.grouplens.org/datasets/movielens/, 2024-05-26.

[33] Pavan Sanagapati. Ad display/click data on taobao.com[EB/OL]. https://www.kaggle.com/datasets/pavansanagapati/ad-display-click-data-on-taobaocom, 2020-04-29.

[34] Zhi Yu. Recommendation data from ele.me[EB/OL]. https://tianchi.aliyun.com/dataset/131047, 2022-05-27.

[35] Kurniawan D H, Wang R, Zulkifli K S, et al. EVStore: storage and caching capabilities for scaling embedding tables in deep recommendation systems [C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2023: 281-294.