一种基于向量索引的内存 OLAP 星型连接加速新技术

张延松[1),2),3)] 张 宇[4)] 王 珊[1),2)]

[1)](中国人民大学数据工程与知识工程教育部重点实验室 北京 100872)

[2)](中国人民大学信息学院 北京 100872)

[3)](中国人民大学中国调查与数据中心 北京 100872)

[4)](国家卫星气象中心 北京 100081)

摘 要 星型连接是 OLAP 中重要的操作,事实表与维表基于星型连接执行多维分析处理. 星型连接的性能取决于连接性能. 当前研究主要集中在如何在不同的处理器平台上优化哈希连接性能,然而如何获得最优的哈希连接参数或实现是一个复杂的问题. 哈希连接不依赖于模式的语义信息,然而却可以在事实表与维表之间通过维映射特征进一步优化连接性能. 该文提出了一种新颖的面向 OLAP 负载的向量索引以提高事实表与维表之间的连接性能. 从模式的角度来看,维表可以映射为向量索引,每一个事实表记录可以直接映射到向量索引上的相应位置,无须执行基于值匹配的哈希连接操作. 从实现技术的角度看,向量索引是一种位图索引、字典表压缩、主外键参照完整性约束和连接索引相结合的技术. 系统化的设计使向量索引可以扮演多种角色:(1)向量索引与位图索引类似起到过滤作用;(2)向量索引相对于只存储 0 或 1 的位图索引使用更多的位来表示更多的信息;(3)映射或创建自动增长的主键作为向量索引地址并且更新相应的外键,将主外键参照完整性约束转换为向量参照约束;(4)外键连接操作简化为通过外键值引用向量单元. 基于向量索引,OLAP 中代价大的星形连接可以抽象为向量索引计算,OLAP 查询可以简化为基于向量索引的单表扫描处理. 向量索引简化的设计不仅可以提升性能,而且降低了在 GPU 平台实现的复杂度. 本文首先讨论了向量索引机制和如何在数据库中应用向量索引;然后设计向量索引更新机制,以保证在更新时向量参照约束;最后提出基于向量索引的 OLAP 框架来提高内存数据库 OLAP 性能. 基于向量索引的星型连接可以用作 GPU 上的 OLAP 加速器,使 CPU 可以将计算密集型负载转移到高性能 GPU 平台来加速 OLAP 处理. 实验结果表明向量索引更新代价较低,而向量引用性能收益较大. 更重要的是,向量索引支持 OLAP 中的星形连接操作在内存数据库引擎之外进行加速,降低了内存数据库的 CPU 负载,或者将星形连接负载通过硬件级加速器,如 GPU 进行加速. 基于向量索引的星型连接可以显著提升 CPU 和 GPU 平台上的星型连接性能,相对于内存数据库 Vector,在 SSB Q4.1 查询可以获得最大 3 倍的性能提升,平均性能提升了 1.2 倍.

关键词 内存 OLAP;外键连接;向量索引;向量引用;星型连接加速

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2019.01686

A Novel In-Memory OLAP Star Join Acceleration Technique with Vector Index

ZHANG Yan-Song[1),2),3)] ZHANG Yu[4)] WANG Shan[1),2)]

[1)](*Key Laboratory of Data Engineering and Knowledge Engineering* (*Renmin University*), *Ministry of Education*, *Beijing* 100872)

[2)](*School of Information*, *Renmin University of China*, *Beijing* 100872)

[3)](*National Survey Research Center at Renmin University of China*, *Beijing* 100872)

[4)](*National Satellite Meteorological Centre*, *Beijing* 100081)

Abstract Star-join is an important operator in OLAP, in which the big fact table needs to join with multiple dimension tables to perform a multidimensional analytical processing. The star-join performance is dominated by join performance. State-of-the-art researches majorly focused on how to optimize hash join performance for different hardware platforms, while achieving the

收稿日期:2017-08-27;在线出版日期:2018-05-10. 本课题得到国家自然科学基金项目(61772533,61732014)和北京市自然科学基金资助项目(4192066)资助.**张延松**,博士,副教授,主要研究方向为数据仓库、内存数据库. E-mail: zhangys_ruc@hotmail.com. **张 宇**(通信作者),博士,高级工程师,主要研究方向为数据仓库、GPU 数据库. E-mail: yuzhang@cma.gov.cn. **王 珊**,教授,博士生导师,主要研究领域为数据库、信息检索、数据挖掘.

maximal hash join parameters or implementation is a complex issue. Hash join doesn't rely on semantic information of schema, while we can further optimize join between fact table and dimension table with dimension mapping feature. This paper introduces a novel vector index for OLAP workloads to accelerate join performance between fact table and dimension tables. From schema perspective, dimension table can be mapped to vector index, each fact tuple can be directly mapped to corresponding positions of vector indexes instead of key matching based hash join. From implementation perspective, vector index is designed as combination of bitmap index, dictionary compression, PK-FK referencing constraint, and join index, the systematic design enables vector index playing multiple roles in query processing: (1) vector index acts as filter like bitmap index; (2) vector index has more bits to present more information than only 0 or 1 of bitmap; (3) mapping or creating incremental PK as vector address and updating corresponding FK column to enable PK-FK referencing constraint as vector referencing constraint; (4) the foreign key join is simplified as referencing vector cell with FK value. With vector index, the costly star-join of OLAP can be extracted as vector index computing, and the OLAP query can be simplified as vector index oriented single table processing. The simplified design of vector index not only improves performance but also reduces the complexity of implementation on GPU platform. We first discuss the vector index mechanism and how to implement vector index inside database, and then design the update mechanism of vector index to guarantee vector referencing constraint during updates, and finally propose a vector index oriented OLAP framework to accelerate OLAP workloads of main-memory databases. The vector index based star-join is employed as an OLAP accelerator on GPU, so that CPU can offload the computing intensive workload of OLAP to high performance GPU platform to accelerate OLAP. The experimental results show that the maintenance overhead of vector index during updates is very low, while the performance gain is huge. Moreover, the vector index enables star-join of OLAP to be accelerated out of in-memory database engine, which can offload the CPU workload of in-memory database or offload the workload to hardware accelerator such as GPU. The vector index based star-join can remarkably improve star-join performance for both CPU and GPU platforms, comparing with the leading in-memory database Vector, the maximal performance gain is achieved by SSB Q4.1 as 3X, the average performance gain achieves to 1.2X.

Keywords　in-memory OLAP; foreign key join; vector index; vector referencing; star-join acceleration

1　引　言

大数据实时分析处理需求带来巨大的性能压力，传统的数据库查询处理技术主要依赖 CPU 的性能，伴随着摩尔定律接近极限而产生的 CPU 在主频，核心数量以及能耗效率上的缓慢增长极大地限制了通用 CPU 上数据库性能的提升. 随着新型处理器技术逐渐成为高性能计算的主流平台，传统数据库系统面临着新硬件技术发展所带来的机遇与挑战.

GPGPU，Xeon Phi，FPGA 等新型处理器技术突破了通用 CPU 技术所面临的硬件扩展性瓶颈，为加速数据库查询处理性能提供了新硬件平台机遇，面向新兴处理器平台的数据库查询处理技术成为学术界和工业界的热点问题. 但数据库软件滞后于硬件发展水平，数据库系统与软件复杂度的提升等问题导致面向硬件特征重写数据库底层实现技术的成本高昂，硬件技术发展的不确定性也增加了新硬件数据库实现技术的风险[1]. 从新硬件数据库研究的技术路线来看，通用化与专用化是两类代表性的技术路线. 通用化强调数据库系统向新硬件平台的迁移，系统开发成本高，但性能优化效果好；专用化技术定位于新硬件性能加速器实现技术，对数据

库系统中计算性能代价较大的少量核心操作进行面向新硬件的性能加速技术研究,实现成本低,对数据库系统影响较小.专用化的数据库系统加速器技术研究的关键是找到数据库查询处理中代价最大的计算型负载,将其转移到新型众核处理器平台进行加速,同时需要该负载能够与数据库系统较好地分离与整合,即实现与数据库系统的松耦合设计.

在关系数据库系统中,连接是关系操作中复杂度较高和执行代价较大的操作,在数据仓库的OLAP操作中,星形连接性能是OLAP整体性能中最具有决定性影响的因素之一.随着内存与处理器技术的发展,内存计算成为新兴的高性能平台,内存数据库也逐渐成为高性能数据库的主流技术.近年来,对内存连接算法进行性能优化成为热点问题[2-7],学术界和产业界在不断探索将算法设计与硬件特性相结合来提升内存连接性能,随着新型众核处理器,如Xeon Phi,GPU以及FPGA在数据处理领域的广泛应用,基于新型处理器的内存连接算法也成为新的研究方向[8-13].

在当前内存连接算法研究中,哈希连接被证明是当前性能最好的算法[2].哈希连接算法性能的决定因素是哈希表访问性能,即在连接内表上创建哈希表并在连接外表上进行哈希访问的性能.在x86处理器平台上cache缓存是最重要的优化技术,针对cache容量和访问特点,哈希连接算法主要分为两类:基于*hardware-conscious*设计的分区哈希连接算法和基于*hardware-oblivious*算法设计的无分区哈希连接算法.基本思想是以cache为中心的优化技术,当cache容量充足及在cache的辅助下内存访问性能较高时采用简单的、不考虑cache等硬件特性的无分区哈希连接算法;当cache容量不足及内存访问延迟较高时将连接表按cache和TLB大小进行分区,从而实现较小分区上cache内的哈希连接性能.当前研究的主要结论是分区哈希连接算法性能优于无分区哈希连接算法性能.然而随着硬件技术的发展,以下硬件条件的影响推动对原有结论进行新的评估:

(1)Xeon Phi众核处理器支持更高的线程(每核心4线程),GPU的SIMT(单指令多线程)机制也有较好的并发内存访问性能;

(2)新一代KNL Phi处理器支持16GB板载高带宽内存HBM配置为自动cache,提供大容量cache缓存;

(3)Phi,GPU和FPGA板载HBM上有限的容量要求算法有较高的内存利用率.

新硬件技术进一步扩展了无分区哈希连接算法的性能优势区间,同时,新硬件对内存利用率要求的提高限制了分区哈希连接算法的应用范围.在OLAP多维查询处理中,星形连接操作性能,即事实表与多个维表之间的多表连接性能,对于整体查询性能具有关键性影响.而当前研究中性能最优的分区哈希连接算法需要将每个阶段的连接结果物化,然后与下一个表共同哈希分区后再执行连接操作,难以有效利用现代内存数据库基于向量处理的流水线处理技术来优化中间结果的存储访问代价,也增加了多表连接时的内存消耗,对于新兴的众核处理器而言,其有限板载内存容量的限制导致众核处理器的数据处理能力大幅降低.

无分区哈希连接算法与数据库模式及负载的特征相结合可以进一步简化算法设计.数据仓库的维表通常较小且采用连续主键,在列存储内存数据库中支持将主键直接映射为记录的偏移地址,这个特点可以实现对哈希连接算法的优化,而且优化后的连接算法可以简化或不使用哈希表结构而采用更加简单的数组、向量等数据结构,在算法中消除指针跳转、哈希值计算、哈希桶定位等复杂的操作,从而更加适合核心功能简化但核心数量较大的众核处理器架构.但这些算法主要应用场景设置为*read-only*模式或专用的数组存储引擎,在移植到已有的内存数据库中时需要解决以何种形式融入现有的查询处理引擎以及在通用的数据管理场景下如何解决更新操作对主键地址映射约束的影响.

本文从硬件技术发展趋势和OLAP模式特点的视角出发,提出内存数据库向量索引技术,通过向量索引实现简单高效的向量连接操作,将传统复杂的哈希连接简化为向量地址映射访问,提高连接操作的代码效率并通过数据结构与算法的简化使其易于移植到新兴的异构处理器平台;针对向量索引要求保证键值-地址映射的强制约束条件,提出了向量索引更新技术,通过逻辑地址映射和批量映射更新机制支持数据更新时的键值-地址映射,并在实验部分评估更新操作对连接性能的影响及代价;进一步地,本文还实现了基于向量索引的星形向量连接技术,以及在GPU处理器上的星形向量连接实现技术,探索了通过硬件加速器卸载CPU计算负载,加速内存OLAP查询处理性能的方法.

本文的贡献体现在三个方面:通过向量索引机制将哈希连接简化为基于地址映射的向量连接操

作，使连接操作更好地适应未来处理器大规模简单核心并行处理的发展趋势，简化算法设计，提高连接算法内存利用率；通过向量更新机制研究解决基于地址映射连接技术对应用场景依赖性过强的问题，使优化的向量引用连接技术适用于通用的 OLAP 应用场景，从而使向量连接技术能够灵活地移植到现有内存数据库查询处理引擎中；通过星形连接技术和 GPU 连接优化技术的研究，探索如何基于向量索引技术加速现有的内存数据库系统以及通过 GPU 加速内存 OLAP 性能，探索面向未来异构处理器平台的内存 OLAP 查询处理实现技术.

本文第 2 节对向量索引相关技术进行对比分析；第 3 节描述向量索引实现技术、向量索引更新实现技术和基于向量索引的 OLAP 星形向量连接加速技术；第 4 节通过实验给出基于向量索引的连接算法的连接操作性能、向量索引更新性能、基于向量索引的内存数据库 OLAP 性能以及基于 GPU 的内存数据库 OLAP 性能；最后对本文进行总结.

2　相关工作

内存连接操作的性能是内存数据库性能的重要影响因素，其中内存哈希连接是应用最为广泛的连接技术. 在以 cache 为中心的 x86 处理器平台，哈希连接操作的性能主要取决于创建哈希表和哈希探测的性能. 创建哈希表的主要代价集中在并行线程创建共享哈希表时的并发访问代价，文献[3]通过优化哈希表存储结构将 *latch* 与哈希表的 *header* 数据结构合并，减少并发访问产生的 cache miss. 当对输入表进行分区时，每个分区由线程独立创建哈希表，消除了哈希表的并发访问冲突，但同时带来分区操作较大的内存空间和处理时间代价；文献[4]采用逻辑分区方法创建物理共享哈希表，通过线程逻辑分区消除创建哈希表阶段的同步代价；而分区哈希连接则通过物理分区操作将连接表划分为适合 cache 大小的分区并在分区上独立创建哈希表，*radix* 分区采用多趟分区技术来优化 TLB 缓存效率，而 *radix* 分区趟数参数的优化配置则需要综合考虑 TLB 缓存优化与分区内存访问代价[5-6]，与页面大小、cache 大小、数据集大小、*radix* 位数等因素相结合使 *radix* 分区操作的优化变得复杂并难以对其性能进行准确评估. 在哈希探测阶段，性能主要取决于哈希表相对于 cache 的大小. 无分区哈希连接使用共享哈希表，当哈希表超过 cache 大小时主要通过现代处理器的超线程、自动预取等机制优化哈希表的内存访问性能，从硬件技术发展的角度来看，新兴的 Xeon Phi 处理器更强大的超线程处理能力（每核心 4 线程）、下一代 KNL Phi 处理器最大 16GB 的 cache 容量以及 GPU 强大的 SIMT 并发内存访问性能为提高哈希探测性能提供了硬件技术支持；从软件优化技术角度来看，优化哈希表存储结构、数据压缩、提高哈希表填充因子[5]等技术通过提高哈希表存储效率来提高哈希探测操作在 cache 的数据局部性.

当前内存哈希连接优化技术研究的应用背景是数据仓库和 OLAP 查询处理领域以外键连接操作为对象. OLAP 应用中的连接操作除了等值连接特点之外，维表主键通常采用无语义的连续整数形式，数据仓库模式特点和数据特点支持了键值与记录位置的一一映射关系，而这种键值-地址映射关系被应用于连接优化技术之中. *invisible join* 算法[5,7]中将事实表外键与维表记录的连接操作简化为 *position extraction* 操作，Blink3 中使用 CAT join[5,8]算法实现基于 dense key 主键特性消除键值与哈希桶映射冲突并消除键值存储空间代价；array join[6]针对 OLAP 模式中所使用的主键 AUTOINCREMENT 约束和 dense primary key distributions 场景通过数组地址访问优化连接操作性能；AIR[9]算法则基于数组 OLAP 存储模型实现数组下标作为维表主键的强制约束条件，将 OLAP 中的事实表-维表连接操作统一为数组地址访问操作，除 AIR 算法外的连接技术采用传统的哈希连接操作处理不满足 dense key 主键特性的连接操作. 本文提出的向量索引则面向 OLAP 应用场景中的主外键连接操作，为数据库增加一种新的向量索引结构，通过 AUTOINCREMENT 约束为主键表增加逻辑向量索引，在外键表中增加新的外键连接列，实现外键列与逻辑向量的一一地址映射，将传统的主外键参照完整性约束升级为向量地址参照约束关系. 通过一系列更新机制保证主键表更新时的向量地址映射，并通过优化的批量向量更新机制提高向量索引的更新性能.

文献[6]指出连接在复杂 OLAP 查询中执行时间占比可能只有 10%～15%，但其结论来自于两表连接查询，并且采用 *ad-hoc* 记录地址访问方式，没有将谓词操作下推，因此放大了连接操作中的谓词处理代价. 文献[9]中采用谓词向量技术优化连接操作时，在 SSB 的 OLAP 查询中星形连接操作的代价占总 OLAP 查询处理代价的 80%以上，连接与星形连接操作是 OLAP 查询性能最重要的影响因素. 在

连接优化技术研究中主要关注两表连接性能,因此需要根据不同记录宽度测试连接性能.在 OLAP 查询处理的星形连接操作中通常采用后物化策略,即首先通过较小的记录 OID 完成多表连接操作,然后再根据多表连接结果抽取相应的记录[7].文献[9]采用动态字典表压缩技术,将较小的维表分组属性压缩为紧凑的维向量,在多表连接时通过将维向量映射为多维数组地址减少输出记录宽度.与 C-store 的后物化星形连接技术[7]相比减少了连接后对维表记录的二次访问次数并用维属性压缩编码替代原始属性进行分组聚集计算.

随着新兴的众核处理器技术逐渐成为高性能计算平台的主流配置,面向众核处理器的内存连接优化技术成为近年学术界研究的热点问题.文献[10]在文献[3]的基础上通过 Xeon Phi 512 位的 SIMD 技术优化哈希连接性能,通过 Phi 处理器更高的超线程处理能力和更宽的 SIMD 并行计算能力提高了哈希表访问性能.文献[11]在 FPGA 上实现了哈希连接算法,主要通过大量并发的硬件级线程掩盖内存访问延迟,从而提高哈希连接性能.GPU[12-15]相对于依赖多级 cache 的 CPU 主要通过其众多的核心和大量硬件级并发访问线程减少内存访问延迟.第二代 Xeon Phi KNL 处理器[16]支持将 16 GB 的 MCDRAM(多通道 DRAM)配置为 cache,从而打破多核处理器每核心 2.5 MB L3 cache slice 的容量限制,从超线程处理能力和提高 cache 局部性两个方面来提高内存哈希访问性能,从而使简单的无分区哈希连接算法具有更好的适应性.随着不同硬件特征的新型处理器逐渐成为新兴的数据库硬件平台,不同连接算法的特征与性能在面对不同类型处理器时也呈现出不同的性能表现,连接实现算法与处理器特性的优化匹配仍然是一个需要不断深入研究的课题.

内存哈希连接算法的实现有多种技术选择[5],优化的复杂度较高.在异构处理器平台结合不同的硬件特性使得哈希连接优化技术更加复杂,如内存哈希表链接桶结构、分支判断处理、复杂数据类型处理、并发控制机制等在 GPU 及 FPGA 等适合大规模并行计算而不适合复杂数据管理及控制的处理器上的效率相对较低,因此从算法实现和优化技术角度来看,传统基于哈希表的连接算法在异构处理器上的实现及优化难度高于新兴的基于键值-地址映射的 CAT join、array join、AIR 等算法,但这些新算法需要解决如何使基于键值-地址映射的连接技术适应通用的 OLAP 应用领域,从而使异构处理器上的连接操作更加易于实现并且高效的问题.

当前基于键值-地址映射的连接技术仅应用于特定应用场景或案例,还未成为 OLAP 中通用的优化技术.数据库更新操作破坏键值-地址映射而导致连接优化失效,为保持键值-地址映射关系的键值重置代价也是一个需要考虑的因素.因此,如何在数据库系统中支持键值-地址映射优化技术,如何支持数据更新,以及如何将以键值-地址映射为基础的连接优化技术应用于内存数据库系统是本文的主要研究内容.从系统架构的角度来看,新型算法直接替代现有内存数据库系统算法的难度较大,要使学术研究成果应用于实现系统不仅需要更高的性能,还需要一种相对灵活的机制来实现"可插件化"的性能加速技术.

3 向量索引

索引是传统数据库中重要的查询加速技术,但索引对于数据库系统而言是一柄"双刃剑",在提供高性能的同时需要付出较高的维护代价.索引采用键值存储访问技术,需要较大的索引存储和索引更新开销,新一代内存数据库由于内存存储成本高和内存查询处理性能相对较好,因此很少使用索引提高查询处理性能.索引是一种"开/关"型加速技术,即索引可以在保持原有查询处理框架的基础上提高特定操作的性能.在当前传统 x86 架构处理器与新兴的众核处理器未来发展趋势尚不明朗的情况下,硬件级索引加速器是一种既能够充分利用新型处理器强大的计算性能,又能够减少数据库系统重新设计开销的可行技术路线.

传统索引结构具有存储空间开销大、计算代价低的特点,与新兴的众核处理器板载内存容量小、计算性能强大的特点相反,因此本文提出一种硬件级的计算型向量索引结构,优化索引存储空间,简化索引维护与更新,强化索引的计算特点,将数据库 OLAP 查询处理中的计算型负载从 CPU 平台转移到众核协处理器平台,减轻数据库的查询处理负担,优化查询处理性能.向量索引简化了连接操作,在存储空间利用率和性能方面有较好的表现,基于向量和向量地址访问的简单存储与计算模型对处理器硬件特性依赖度降低,易于迁移到不同架构的处理器平台,降低了数据库查询优化技术的硬件迁移成本.

3.1 向量索引结构

向量索引是一种保持键值-地址映射关系的索引机制. 向量索引形式上为数组结构的向量，向量宽度由向量索引中存储非空值元素个数决定，即 $W_{vec}=\log_2|vector|$，向量长度由键值决定，实现将键值直接映射为向量单元的偏移地址，即 $L_{vec}=\mathrm{Max}(value)$.

向量索引可以定义如下：

$Vector\ Index=\{v_i\mid 0<=i<=n\}$，向量索引是一个有序的数值序列，其中 i 为默认的向量单元下标，n 代表向量元组的数量，v_i 表示向量单元 i 中的元组值，可以为空值或非空整数，表示该向量单元不满足向量索引过滤条件或者向量索引在该单元的取值.

从结构上看，传统的位图索引相当于宽度为 1 bit 的向量索引，向量索引相当于存储多位的位图索引，但二者在索引机制上有较大的区别. 我们以图 1 为例说明向量索引与位图索引之间的区别：

(1) 向量长度. 位图索引与表的行数相等，每一位对应表中的一条记录，即位图索引取决于表的物理记录长度，如图 1 中 customer 表中记录有 4 条，位图长度为 4；向量索引的长度可以超过表记录行数，当主键值为连续整数时，向量与表等长，当记录中主键有缺失值时，向量长度超过表记录数量，即向量索引取决于逻辑记录长度，如图 1 中记录主键最大值为 5，向量索引长度为 5，缺失值 4 对应向量索引中的一个空值单元；

(2) 向量值. 位图索引只能表示单一的状态 0 或者 1，用于为表属性成员创建位图集合，如图 1 所示，当查询包含多个属性值时需要进行多个位图之间的位运算才能得到相应的结果；向量索引可以表示属性集合信息，属性中的不同取值存储在字典表中，如图 1 中所示字典表，向量索引中存储字典表编码，即向量索引是一个压缩的投影列；

(3) 向量映射. 位图索引实现的是记录 RID 与位图单元的映射，位图索引与物理记录形成一一映射关系；向量索引实现的是表主键值与向量索引单元的映射，也就是说，向量索引与表的主键值形成一一映射关系；

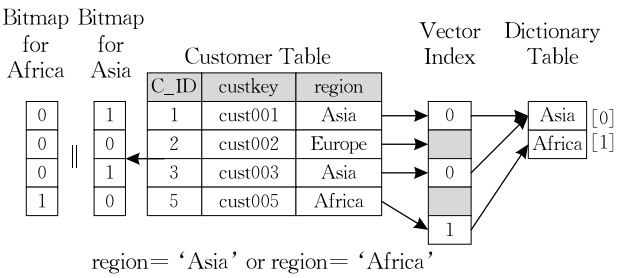


图 1　向量索引与位图索引示例

(4) 使用方式. 位图索引需要预先创建，是一种静态索引，需要根据查询访问相应的位图，位图索引的大小取决于创建位图索引的属性中不重复值的数量；而向量索引是动态更新的索引，通过表上的选择和投影操作实时更新向量索引内容，是一种通过计算生成的索引；

(5) 向量依赖. 位图索引对记录键值没有依赖关系，主要用于过滤索引所在表的记录，而向量索引需要表的主键列满足连续增长自然序列的条件，实现主键值与向量索引单元地址的一一映射. 向量索引将表过滤后投影的结果存储为向量，用于过滤与该表具有主外键参照完整性约束条件的连接表，即用于连接过滤.

向量索引的结构非常简单，但向量索引在使用中需要结合数据压缩技术、主键约束与主外键参照完整性约束机制，用于加速外键连接操作，是一种面向 OLAP 领域特征而定制的索引技术.

3.2 向量索引实现技术

在内存列存储数据库中，最简单的向量索引实现技术是使用列偏移地址作为主键，实现向量索引单元与记录的一一映射. 文献[14] 提出的 A-store 采用数组存储引擎，将数组下标用作表的主键，从而实现主键值与向量索引的一一映射. 为保持键值与向量索引地址的映射关系，数组存储引擎需要使用原位更新策略和删除记录位置复用技术以保证更新时的键值与向量索引地址的物理映射关系. 数组存储引擎简化了存储模型与 OLAP 查询处理模型设计，是一种面向 OLAP 多维存储模型的定制化存储引擎.

进一步地，我们讨论如何利用关系数据库既有的实现技术与机制实现向量索引. 与 array join[7] 类似，我们同样可以利用数据库对属性的 AUTO_INCREMENT 约束机制在数据库引擎中实现向量索引技术. AUTO_INCREMENT 约束创建一个自动增长序列，我们将 AUTO_INCREMENT 约束的属性作为向量索引地址映射主键. 为实现向量索引技术，我们设计了如下策略：

(1) 增加地址映射主键

为 OLAP 数据库中的维表增加 AUTO_INCRE-

MENT 约束属性列，作为维表新的地址映射主键.

(2) 外键更新策略

在相应的主外键参照完整性约束条件的外键列所在表增加新的地址映射外键列或者将原外键列更新为新的地址映射外键列. 通过基于原主外键约束关系的连接操作将原主键对应的地址映射主键值更新为新的外键值.

(3) 向量索引动态更新机制

AUTO_INCREMENT 约束主键对应一个向量索引，即与主键值一一映射的数组向量. 数组长度由主键最大值确定，数组宽度由维表上选择投影操作结果集的字典压缩编码宽度确定. 向量索引是一种动态创建或更新的索引结构，将维表上选择投影操作的结果以数据压缩形式存储为向量，并保持向量地址与主键值的一一映射.

简而言之，向量索引是维表的一个附加列，其存储空间相对固定，向量索引不预存储索引值，在查询时动态生成向量索引值，与普通索引，如 B+树索引、位图索引等不同，在插入记录、删除记录、修改记录时不具有同步更新索引中相应的数据的同步代价.

3.3 向量索引更新技术

AUTO_INCREMENT 约束主键在更新操作时可能造成键值不连续，如 *delete* 操作导致键值序列产生“空洞”，*update* 操作转换为 *delete* 和 *insert* 操作后发生位置变化，破坏键值与向量地址映射关系，为解决更新操作对向量索引的影响，我们采用如下策略：

(1) 逻辑键值-向量索引地址映射

在数据库中使用 AUTO_INCREMENT 约束主键后，由于数据库采用不同的更新技术，无法保证键值与向量索引单元偏移地址的物理一一映射关系，即 AUTO_INCREMENT 约束只能保证键值的递增属性而不能保证键值的物理连续性. 我们将 AUTO_INCREMENT 约束主键作为向量索引的逻辑地址，在生成动态向量索引时将记录的压缩编码存储到主键值对应的向量索引单元中. 如图 2 所示，在生成向量索引时产生向量索引上的随机内存地址访问，相对物理地址映射产生了一定的 cache miss 代价，但解除了主键与向量索引地址的物理映射关系，使向量索引可以借助于数据库固有的 AUTO_INCREMENT 约束机制实现向量索引，而不需要像 A-Store 一样需要定制数组存储引擎来支持向量索引，从而保证向量索引技术易于集成到不同的数据库引擎中.

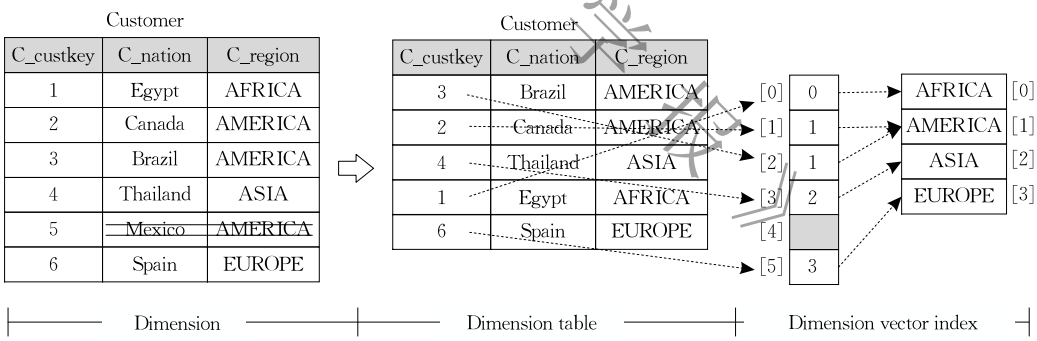


图 2 逻辑键值-地址映射向量索引

(2) 批量更新

对于 *delete* 操作，我们采用延迟批量更新策略，当删除记录数量达到一定阈值(如 20%)或者小于 LLC 大小的向量因删除操作产生大量“空洞”而导致向量大小超过 LLC 时，执行批量更新操作.

采用延迟更新策略时，删除记录的位置(主键值)被末尾记录复用，以减少更新操作影响记录的数量. 如图 3 中主键值为 2 和 4 的删除记录的主键分别分配给末尾主键值为 5 和 6 的记录，填补向量索引中的“空洞”. 更新后的主键值需要同步更新到外键中，我们通过向量索引完成外键更新操作. 如图 3 所示，按原始维表创建向量索引，将未更新记录对应的向量索引单元置为空，更新主键记录对应的向量索引单元中存储更新后的键值，如向量索引下标 4 对应 4，5 对应 2，然后外键列将外键值作为向量索引中连接相应单元的地址对向量索引进行地址探测(address probing，相对于传统哈希连接中的 hash probing 实现基于地址的直接探测，消除了哈希映射及键值匹配等处理过程)，当向量单元非空时，用向量单元值更新外键值，完成外键列的对应更新. 外键更新操作代价为基于向量引用技术的连接操作，外键更新操作执行时间由向量长度和更新率决定.

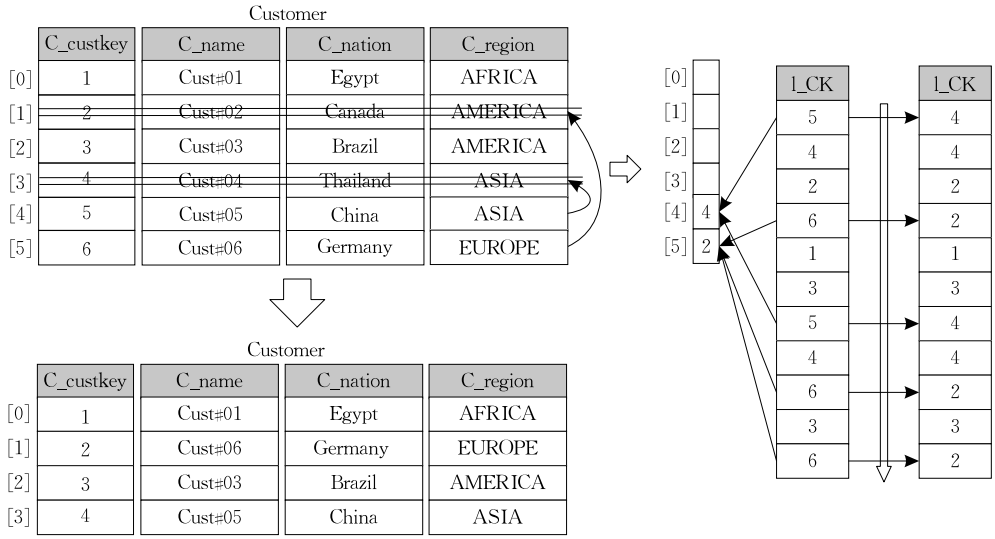


图 3　批量向量索引更新

3.4　基于向量索引的内存 OLAP 实现技术

向量索引机制将数据库中普遍使用的主外键参照完整性约束转化为向量参照完整性约束，传统的外键连接操作简化为外键值在向量索引中的引用访问，称为向量连接操作. 向量连接操作算法描述如下.

算法 1.　向量连接 Vector Join.

输入：向量索引 *VectorInx*，外键列 *FKCol*

输出：连接结果 *JoinResult*

```
BEGIN
  FOR EACH fkValue IN FKCol DO
    JoinItem←getVectorInxValue(VectorInx,fkValue);
    IF JoinItem IS NOT NULL
    THEN JoinResult←JoinItem;
//根据外键值fkValue访问向量索引VectorInx[fkValue]中的值,当向量单元值非空时,生成连接结果,加入连接结果集
  END FOR
Return JoinResult;
END
```

如图 4 所示，FK_1 列与 Dim Vector $Index_1$ 向量索引执行向量连接操作，FK_1 列中的属性值映射到向量索引单元地址，当向量索引单元为空时，不满足连接条件，当向量索引单元非空时，将向量索引单元值作为连接结果输出. 向量索引简化了连接算法设计，优化了连接性能.

数据仓库多维数据模型决定了事实表需要与多个维表执行星形连接操作来实现多维分析处理，在 OLAP 操作中，星形连接性能是影响 OLAP 查询处理性能的重要因素. 相对于两表连接操作，星形连接既需要较高性能的连接算法，又需要优化多表连接过程连接中间结果的物化代价，减少内存开销与额外的计算代价. 同时，星形连接操作衔接了连接与分组聚集操作，我们进一步使用向量索引加速分组聚集操作性能.

我们在事实表上创建向量索引. 维表向量索引的长度取决于主键值，而事实表向量索引则与事实表等长，与记录物理顺序一一对应，用于记录星形向量连接操作的结果集. 事实表向量索引是在星形向量连接操作中迭代生成的，事实表向量索引既记录了每一次向量连接操作的结果，也起到位图索引的作用，向量索引中非空位置用于过滤下一个外键列.

图 4 给出了基于事实表与三个维表的星形向量连接的处理过程. 每个维表生成一个向量索引和一个相应的字典表，三个字典表构成一个 3 维数组，用于表示 3 个字典表编码构成的地址空间. 维表向量索引中深色阴影单元为空值单元，可以用向量宽度对应的最大值表示，向量能够表示的最大分组数为 2^n-1，n 表示向量位数，$0\sim2^{n-1}$ 表示向量编码，2^n-1 表示空值. 设 3 个维向量字典表中元素数量分别为 I,J,K，则完成外键列与 3 个维表向量索引的星形向量连接时满足连接条件记录的全局向量压缩编码可以表示为 3 维数组下标形式，即 $i\times J\times K+j\times K+k$. 在图 4 所示的星形向量连接示例中，每一个向量连接的结果用于更新事实表向量索引，然后执行基于事实表向量索引的外键列过滤和向量连接操作，迭代更新事实表向量索引，在迭代更新过程中计算前缀多维数组下标值，最终计算出全局的、

基于多维数组形式的压缩编码.星形向量连接操作的结果为向量索引,用于在事实表上执行向量聚集计算,即根据事实表向量索引非空值过滤事实表度量属性列,然后以向量索引值为分组执行分组聚集计算.从整体看,不同的 OLAP 查询的星形向量连接操作中输入是较小的维表向量索引,输出通常是比较稀疏的事实表向量索引,事实表外键列作为固定的计算数据集可以驻留在内存或众核处理器的高带宽板载内存中以提高星形向量连接操作性能.

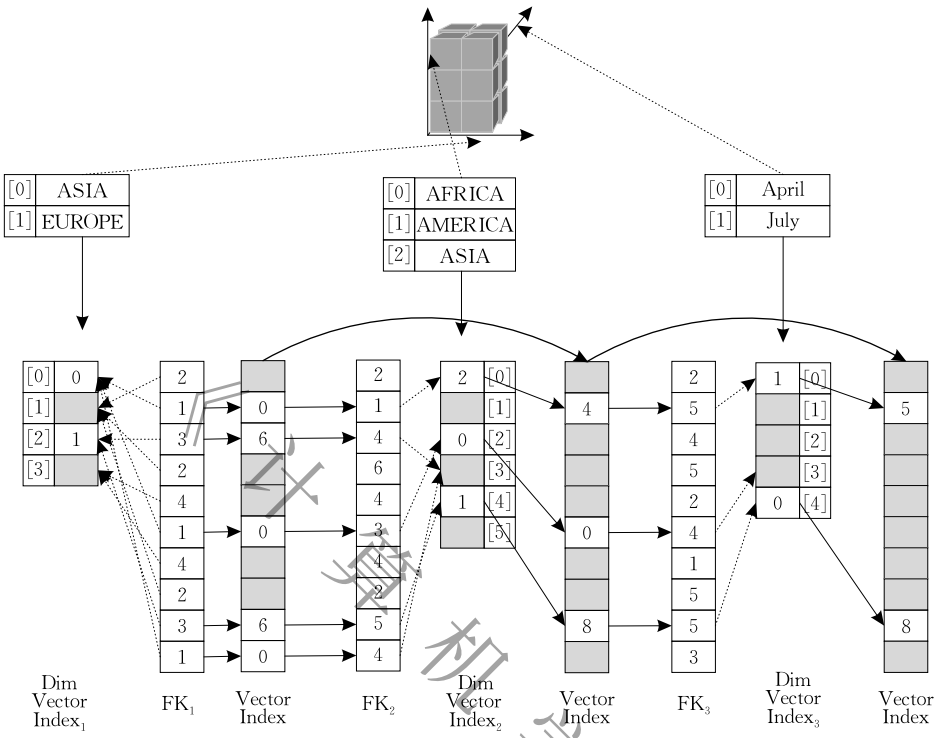


图 4 星形向量连接操作示例

星形向量连接算法描述如下.

算法 2. 星形向量连接 Star Vector Join.

输入:向量索引集合 *VectorInx*,外键列集合 *FKCol*,维表向量索引字典表元组数量集合 *DicCard*,事实表记录数量 *F_num*

输出:向量索引 *VectorInx*

```
BEGIN
  InitVec(Fact_VecInx,RowNumber(FKCol[0]));
  //初始化事实表向量索引
  FOR i=0 TO NumOfFKCol (FKCol) DO
    IF (i==0) THEN
      BEGIN
        FOR j=0 TO NumOfFKTuples(FKCol[0]) DO
          IF (VectorInx[i][FKCol[i][j]]) IS NULL
          THEN Fact_VecInx[j]=NULL;
          ELSE Fact_VecInx[j]=AggInx(DicCard);
        END FOR
    //第一个向量连接结果填充到向量索引中,根据外键值映射的向量索引单元将事实表向量置空或迭代计算的多维数据下标
    ELSE
      BEGIN
        FOR j=0 TO NumOf Fact_VecInx(Fact_VecInx)
        DO
          IF (Fact_VecIn[j]) IS NOT NULL
          THEN
            IF (VectorInx[i][FKCol[i][j]]) IS NULL
            THEN Fact_VecInx[j]=NULL;
            ELSE Fact_VecInx[j]=AggInx(DicCard);
        END IF
      END FOR
    //以事实表向量索引为外键列过滤索引,根据非空位置访问外键列,执行向量连接操作,更新事实表向量索引
    END IF
  Return VectorInx;
  END
```

在多趟多维索引计算中,外键列通过键值地址映射到维表向量索引相应单元地址,当向量单元为空值时,将对应的事实表向量索引单元置为空值;当

维向量索引单元非空时，根据维向量压缩编码和 OLAP 查询对应的相关维字典表进行多维编码计算，并将当前阶段的计算结果存储在事实表向量索引对应的单元中. 生成的事实表向量索引，在下一个外键连接阶段作为过滤器只访问非空单元对应的外键值，采用同样的维表向量索引单元访问和多维编码计算，并更新事实表向量索引. 当完成全部外键的向量索引计算时，得到事实表向量索引，非空单元对应满足 OLAP 查询选择和连接条件的记录位置，向量索引单元值表示对应 OLAP 分组属性的多维压缩编码，基于事实表向量索引可以完成 OLAP 其后的分组聚集计算，并通过将多维压缩编码映射到相应的字典表输出分组属性.

在向量索引计算中使用了两种类型的索引，一种是主键表(维表)上的向量索引，用于实现主键值与向量单元地址的映射，以及外键值向向量索引单元的地址映射访问，相当于外键连接过滤器；另一种是外键表(事实表)上的向量索引，用于过滤星形向量连接的外键列，并生成最终的外键表(事实表)向量索引. 通过计算生成的向量索引将 OLAP 的星形连接操作转换为简单的基于向量索引的聚集计算.

向量索引的意义在于将 OLAP 查询处理中最为复杂的星形连接操作转换为星形向量连接操作，并生成向量索引，建立了一种计算型索引机制.

3.5　基于向量索引的 GPU OLAP 星形连接加速器

在 OLAP 查询中，选择、投影、分组、聚集都是较为简单的操作，低选择率的向量索引对应的随机内存访问也适合 CPU 处理，而多维向量索引计算是 OLAP 查询处理的主要代价，可以通过现代多核处理器，如 GPU、Phi、FPGA 等并行计算性能强大硬件加速器设备的进行加速，实现硬件级的向量索引，分离 CPU 上的 OLAP 星形连接处理负载.

图 5 为基于 GPU 的星形向量连接加速技术. CPU 负责关系数据管理，创建维表向量索引，基于向量索引的分组聚集计算等数据访问密集型任务，而计算密集型的星形向量连接计算则由 GPU 承担，从而达到将少量数据(外键表)上的高计算负载通过众核处理器进行硬件加速的目标.

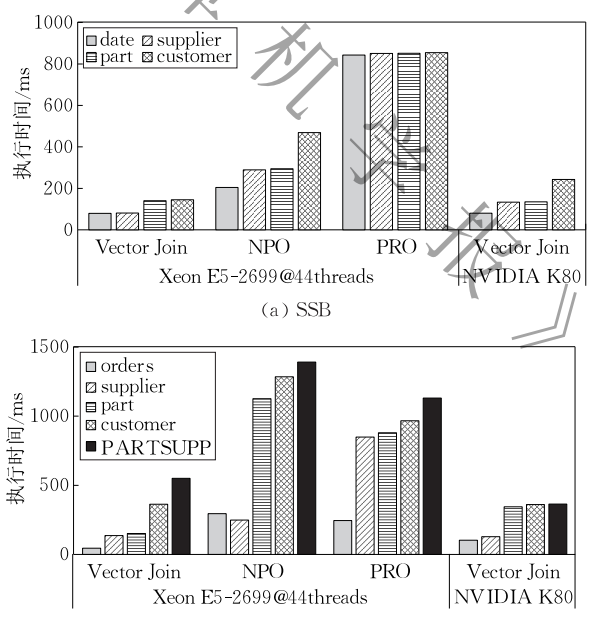


图 5　基于 GPU 的星形向量连接加速技术

在具体实现中，考虑到 OLAP 查询选择率相对较高，面向多维分析的上卷，下钻等操作涉及不同选择率的查询的问题，以及最大化利用 GPU 高性能设备内存的目标，我们将事实表外键列和事实表向量索引列驻留在 GPU 设备内存，较小的维表向量索引和事实表向量索引在查询中动态生成并在在 CPU 与 GPU 之间传输.

GPU 向量索引模块采用 cuda 编程方式，为事实表外键列、事实表向量索引列、维表向量索引列预分配 pinned memory 空间，提高内存访问性能. GPU 向量索引实现使用数组、数组地址访问等简单操作，减少传统哈希表的指针跳转、分支判断等不利于 GPU 上大规模 SIMT(单指令多线程)并行处理的操作，事实表外键列与事实表向量索引数组地址

一一对应，实现向量索引计算过程中的数据级并行，减少对共享资源的争用. 与 CPU 平台不同，GPU 向量索引计算时需要通过优化较小向量索引、计数器等高频访问数据在 shared memory (L1 cache)中的缓存来减少对设备内存访问的延迟，需要显式设计 shared memory 中存储的数据结构及替换方法. 在并行向量索引计算时，还需要根据 GPU 的硬件特征优化配置 BLOCK 和 THREAD_NUM 等重要的并行计算参数以获得 GPU 上最优的并行计算性能，在算法实现及代码优化方面与 CPU 平台有所不同.

每个 OLAP 查询动态产生在查询相关维表上的向量索引，并通过 PCI-E 通道更新到 GPU 设备内存的维向量索引数组中，然后调用 GPU 向量索引模块执行 GPU 上的向量索引计算，并将向量索引计算的结果更新到事实表向量索引中，向量索引通过 PCI-E 通道传输给 CPU，由 CPU 根据向量索引完成其后的分组聚集计算.

在 GPU 向量索引计算模块中，数据库模式特征决定了向量索引的存储空间，采用与事实表等长的向量索引支持 OLAP 查询的选择率变化，当查询的总选择率较低时可以在向量索引传输时进一步采用数据压缩技术减少向量索引传输代价. GPU 向量索引只存储事实表外键列的特性使较小的 GPU 设备内存能够支持较大数据集的向量索引计算加速，这种由数据库模式决定的向量索引设计可以根据 GPU 设备内存大小计算出对指定数据库支持的最大数据量；当数据库较大时，可以采用多块 GPU 并行向量索引计算模式，将事实表外键列及事实表向量索引列水平分片分布在不同的 GPU 上，查询的维表向量索引同时广播到各 GPU 卡上进行并行计算.

综上所述，向量索引是一种面向 OLAP 查询处理特征而设计的、优化外键连接操作的新型索引技术，通过向量索引将复杂的连接操作简化为外键对向量索引单元的地址映射访问，更重要的是，向量索引机制将 OLAP 查询处理中最复杂和代价最大的星形连接操作封装为星形向量连接计算，并易于在众核处理器平台实现.

4 实验结果与分析

本文实验的主要目标是验证三个基本问题：(1)测试向量索引更新操作代价，评估向量索引更新代价对连接性能的影响程度，确定向量索引的实用性；(2)测试基于向量索引的向量连接操作性能，评估引入向量索引机制后相对于传统连接技术的连接性能提升空间，测试在 x86 多核处理器平台和 GPU 平台上的向量连接性能，分析通过 GPU 加速向量连接操作性能的可行性；(3)测试基于向量索引的 OLAP 星形连接加速性能，通过不同算法、数据库及 GPU 平台的星形连接性能对比分析基于向量索引的 OLAP 星形连接加速技术的可行性，并通过模拟实验评估在内存数据库 Actian Vector 中引入基于向量索引的 OLAP 星形连接加速技术的实现方案及性能提升空间. 通过三个层次的实验，我们分别从可行性、性能、数据库系统集成性等方面综合评估向量索引技术在异构处理器平台内存数据库技术的可行方案和性能.

实验平台为一台 Supermicro SuperWorkstation 7047GR-TPRF 工作站，配置有 1 块 Intel Xeon E5-2699 v4@2.2 GHz 22 核心 CPU，44 个物理线程，55 MB L3 cache，256 GB DDR4 内存，操作系统为 CentOS 7，Linux 版本为 3.10.0-514.16.1.el7.x86_64，gcc 版本为 4.8.5. 此外还配置了 1 块 NVIDIA® Tesla™ K80 GPU，显存容量为 24 GB，其中集成了 2 个 Kepler GK210 核心，每个 GK210 核心集成了 2496 个流处理器，共计 4992 个流处理器，shared memory 为 128 KB. 实验中使用 SSB，TPC-H 和 TPC-DS 数据集($SF=100$)验证向量索引的更新性能和向量连接性能；我们使用文献[3]所提供的开源算法 NPO 和 PRO 作为两种代表性的内存哈希连接算法，分别表示基于共享哈希表的无分区哈希连接算法和基于 *Radix* 分区的内存哈希连接算法，并将向量连接算法集成到开源连接算法中，以保证向量连接与哈希连接具有相同的代码效率和数据结构；我们使用在 OLAP 星形连接查询加速性能实验中，使用 TPC-H 测试中性能最好的 Actian Vector 5.0 内存数据库作为 OLAP 性能对比内存数据库系统，模拟在 Actian Vector 中通过 SQL 完成向量映射和向量聚集计算，通过 C++和 cuda 开发的向量索引计算作为星形连接加速器，对比原始 Actian Vector 与基于向量索引计算加速后的 Actian Vector 性能，评估引入基于向量索引的 OLAP 星形连接加速技术后对内存数据库系统的性能提升作用.

4.1 向量索引更新性能

我们测试了基于图 3 的批量向量索引更新性

能. 如表 1 所示，选择率增长步长为 10%，0 时为基于向量索引的向量连接性能. 在 $SF=100$ 的 SSB 数据集中，4 个维向量索引最大仅为 3MB，更新操作对应的向量连接完全是 in-cache 操作，因此不同维表上向量索引的更新性能差异不大，最大向量索引更新代价仅为 1.2 cycle/tuple，低于 NPO（最大 1.65 cycle/tuple）与 PRO（最大 3.14 cycle/tuple）算法的连接执行代价[14]. 向量索引的更新操作相当于一次额外的向量连接操作，由于向量连接性能较高，因此向量索引更新代价控制在较低的范围.

表 1　SSB 数据集向量索引更新性能

Update ratio/%	date	supplier	part	customer
0	0.61	0.61	0.71	0.72
10	0.88	0.89	0.92	0.93
20	1.03	1.08	1.06	1.06
30	1.12	1.14	1.13	1.13
40	1.16	1.17	1.14	1.12
50	1.18	1.16	1.14	1.12
60	1.18	1.17	1.15	1.15
70	1.18	1.18	1.17	1.18
80	1.17	1.17	1.19	1.18
90	1.17	1.17	1.20	1.19

对于 $SF=100$ 的 TPC-H 数据集，其中包含较大的事实表 PARTSUPP 和 orders 表，且两表满足主外键参照完整性约束条件，两个事实表上的向量索引更新则产生内存访问代价.

表 2 中 customer、supplier 和 part 表上的更新代价最大仅为 1.28 cycle/tuple，而两个较大的参照事实表向量索引更新代价最大达到 3.64 cycle/tuple，仍然低于 NPO（最大 4.96 cycle/tuple）与 PRO（最大 4.15 cycle/tuple）算法的连接执行代价[14]. 当向量索引大小超过 cache 容量时，CPU 的分支预测技术对低选择率或高选择率的更新操作有较好的分支预测命中率，因此在表 2 中的更新性能呈现中间高，两端低的特点.

表 2　TPC-H 数据集向量索引更新性能

Update ratio/%	customer	supplier	part	PARTSUPP	orders
0	0.81	0.70	0.80	2.56	3.08
10	1.01	0.92	0.98	2.71	3.21
20	1.14	1.06	1.08	2.83	3.37
30	1.21	1.13	1.16	2.94	3.58
40	1.22	1.14	1.17	2.96	3.64
50	1.22	1.13	1.18	3.17	3.54
60	1.23	1.15	1.16	3.01	3.61
70	1.25	1.17	1.20	3.02	3.63
80	1.26	1.19	1.17	2.93	3.55
90	1.27	1.20	1.18	2.91	3.38

在 OLAP 负载中，由于维表较小且增长缓慢，维表向量索引通常小于现代主流多核处理器的 LLC 容量，一方面我们可以采取延迟更新策略，对删除的向量索引元组暂不处理，牺牲部分存储效率达到简化更新操作的目标，当向量索引小于 LLC 容量时仍然能够保持较高的向量连接性能；另一方面我们结合 OLAP 负载中维表更新较少的特点采用批量向量索引更新策略，当向量索引中删除的向量索引元组达到一定阈值时启动批量向量索引更新操作，并同步更新相应较大的事实表向量索引.

表 3　逻辑向量索引更新性能

TPC-DS Cycles Increment/%	AIR Execution Time/%			BUILD/%
	BUILD	PROBE	TOTAL	
reason	36.69	−0.57	−0.17	1.06
store	42.48	−0.12	−0.07	0.13
promotion	37.83	0.41	0.46	0.13
household_demographics	16.99	−0.03	−0.00003	0.18
date_dim	286.70	−0.13	0.08	0.07
time_dim	298.47	−0.04	0.20	0.08
item	112.67	0.06	0.29	0.20
customer_address	190.71	0.29	1.08	0.41
customer_demographics	171.94	−0.04	1.35	0.81
customer	207.83	0.08	1.64	0.75
store_returns	246.21	−5.28	7.76	5.19

逻辑向量索引在创建时产生额外的向量映射代价，表 3 列出了 TPC-DS 数据集（$SF=100$）上基于逻辑向量索引的向量连接操作相对于基于物理向量索引的向量连接操作在向量映射、向量地址访问和总的向量连接时间上的对比. 由于逻辑向量索引的键值与地址失去了物理对应关系，在向量映射过程中产生内存随机访问代价. 如表 3 所示，较小维表向量索引上的逻辑地址映射发生在 L1 或 L2 cache，因此逻辑映射代价较低，而随着向量索引的增长，逻辑向量映射逐渐产生 L1 cache miss、L2 cache miss、L3 cache miss 等代价，向量映射代价升高. 在向量地址访问（向量单元引用）阶段性能影响较小，在总查询处理时间上只有较大的逻辑向量索引产生超过 1%的额外代价，较大的参照事实表 store_return 在向量映射阶段执行时间增长了 246%，向量连接性能受到较大的影响，总的向量连接时间增长超过 7%.

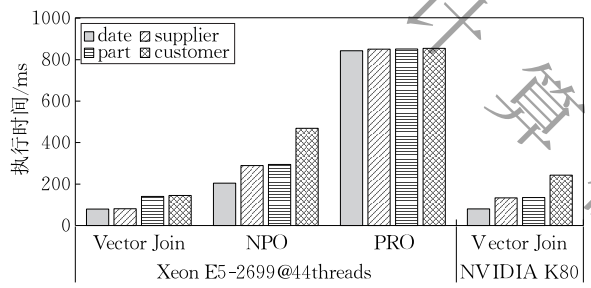
总体来说，向量索引更新代价较低，批量更新和逻辑向量索引机制适合 OLAP 查询处理中维表小、更新频度低等数据仓库固有的特点.

4.2　向量连接操作性能测试

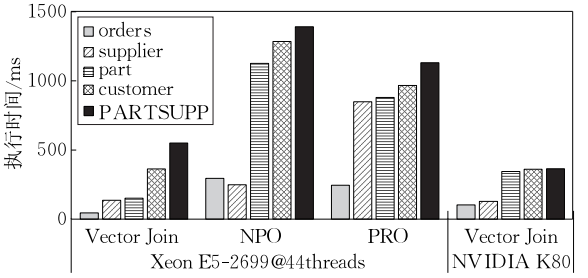
我们对比了基于向量索引的向量连接算法与近年学术界所使用的开源哈希连接算法 NPO（无

分区哈希连接)与 PRO(*Radix* 分区哈希连接)[3]的性能.

如图 6(a)所示,SSB 数据集中较小的维表 date,supplier 的连接性能比较接近,而 GPU 上的向量连接性能略低于 CPU,主要原因是 CPU 的 cache 机制对于 in-cache 连接操作具有较高的性能;本文实验平台 E5-2699 v4 的 LLC 为 55 MB,相对于 SSB 较小的四个维表来说,NPO 算法所生成的共享哈希表有较好的 cache 数据局部性,因此 NPO 算法性能优于 PRO 算法性能;同样地,由于较大的 LLC,SSB 数据集对应的四个向量连接操作在 CPU 上的性能略高于 GPU. 对于向量连接算法而言,当向量索引小于 LLC 时,CPU 上的向量连接性能通常优于 GPU. 例如,当使用宽度为 int_8 的向量索引时,E5-2699 v4 CPU 上向量连接性能超过 GPU 的向量索引阈值为 55 M 行(55 MB/1B).



(a) SSB



(b) TPC-H

图 6 向量索引与连接性能比较

图 6(b)中显示了 TPC-H 各表连接性能,TPC-H 相对于 SSB 数据集各外键表相对较大,在 CPU 平台除较小的 supplier 表外,PRO 算法显著优于 NPO 算法;向量连接性能显著优于 NPO 与 PRO 算法,向量连接在大表连接时同样产生内存访问延迟,但相对于 NPO 与 PRO 算法仍然有较好的性能;GPU 向量连接性能在较小的表连接中(如 customer,supplier,part)低于 CPU,但在大表连接中(PARTSUPP,orders)较大的向量索引访问通过 SIMT 机制有效地通过多线程并发掩盖内存访问延迟,相对于 CPU 平台向量连接性能受向量索引大小影响的特点而言,GPU 平台上的向量索引性能对向量索引大小敏感度较低,始终保持较为稳定的性能.

从向量连接在 CPU 和 GPU 平台的性能测试结果来看,当向量索引小于 CPU 的 LLC 时,向量连接在 CPU 上的性能优于 GPU,当向量索引大于 LLC 时,向量连接在 GPU 上的性能优于 CPU;向量索引大小取决于数据仓库模式特点和维表数据增长特征,随着多核 CPU LLC 容量的增长,CPU 平台能够支持更大范围的高性能向量连接操作;GPU 平台适合处理较大向量索引对应的向量连接操作,除具有较优的性能以外,GPU 平台上的向量连接具有较好的稳定性,简化了 GPU 上的代价估算模型设计.

4.3 基于向量索引 OLAP 星形连接性能

星形连接是 OLAP 中的重要操作符,是多维分析处理的基础操作. 与独立的连接操作性能特征不同,星形连接性能一方面取决于连接算法性能,另一方面也受多表连接时中间结果的物化策略影响. 现代数据库普遍采用流水线处理模式来消除多表连接操作中的中间结果物化代价,达到提高星形连接整体性能的目标. 相对于 NPO 算法,PRO 算法通过分区提高连接操作的数据局部性,提高连接性能,但在多表连接中需要将连接结果物化后再进行下一阶段的分区操作,从而产生较大的物化代价,影响星形连接的整体性能. 为进一步探索星形连接性能,我们基于 SSB 数据集与四个维表的星形连接测试了基于不同连接算法和代表性内存数据库的星形连接性能.

如图 7 所示,我们实现了基于向量连接、NPO 和 PRO 算法的星形连接算法,其中 NPO 算法采用流水线处理模式,而 PRO 算法则需要将当前连接结

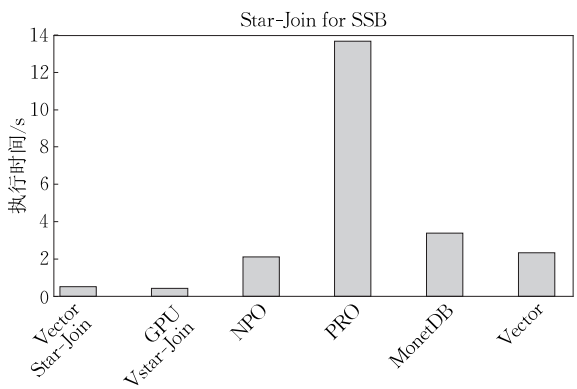


图 7 向量索引与连接性能比较

果物化后再执行与下一个表之间的 *Radix* 分区哈希连接操作，向量索引星形连接采用如图 4 所示的向量索引迭代连接方法，在多表连接中复用物化的向量索引，通过向量索引依次过滤各连接外键列，星形向量连接采用以 CPU 平台上 C++和 GPU 平台上 cuda 开发的程序实现. 相对于基于流水线的星形连接算法，向量索引迭代星形连接算法仅物化并复用固定大小的向量索引，以外键列为单位的迭代向量连接既提高和连接操作的代码效率，又避免了列处理模型较大的中间结果物化开销，当查询选择率较低时，向量索引的过滤作用能较好地提高外键列的访问效率，提高星形连接的整体性能. 向量连接算法分别实现了 CPU 与 GPU 平台的星形连接算法，在实验中我们设置每个维表上的选择率为 100%，消除选择率对星形连接操作性能的影响. 我们还测试了代表性的内存数据库 Vector 5.0 和 MonetDB v11.25.15 上的星形连接性能，测试 SQL 命令为

```
SELECT COUNT(*)
FROM lineorder,date,part,supplier,customer
WHERE lo_orderdate=d_datekey
AND lo_suppkey=s_suppkey
AND lo_partkey=p_partkey
AND lo_custkey=c_custkey;
```

从图 7 所示的星形连接测试结果来看，PRO 星形连接算法性能较差. 一方面是 SSB 数据集维表较小，在 LLC 较大的情况下 NPO 的 in-cache 操作更具优势，相对地，PRO 性能较差；另一方面是 PRO 算法在星形连接中所产生的中间结果的物化代价较高，而 NPO 算法采用类似的流水线处理模式，优化了连接中间结果物化代价. 星形向量连接在 CPU 和 GPU 上的性能接近，GPU 略优于 CPU，当维表上谓词操作产生较低的选择率时，CPU 相对于 GPU 更好的分支判断和自动预取等硬件级优化技术使其优于 GPU 星形向量连接算法. MonetDB 与 Vector 数据库的星形连接性能略低于 NPO 星形连接算法性能，其星形连接实现技术与 NPO 比较相近.

从三种连接算法的内存利用率来看，NPO 算法需要为每个查询实时创建相应的哈希表，哈希表大小取决于连接表上的选择率、哈希表结构、哈希填充因子、哈希记录大小等因素的影响，在低选择率时哈希表较小，在高选择率时哈希表较大；PRO 算法的主要开销为 *Radix* 分区代价，采用多趟分区策略（为优化性能通常采用 2-pass 分区）需要将两个连接表按相同的 *Radix* 值进行分区，加倍了算法的内存开销，降低了内存数据库在大数据处理时的有效内存利用率，尤其在设备内存容量较小的 GPU、Phi、FPGA 等硬件加速器设备上能够处理的连接数据量通常不超过设备内存容量的 50%，增加了数据处理时的 PCI-E 数据传输代价；向量索引算法则采用定长向量存储方式，即数据库中查询共享与维表等长（或与逻辑键值等长）的向量索引，不同的查询及不同的选择率仅决定向量索引中存储值的内容及分布，不产生动态的内存需求，简化了内存数据库及硬件加速器上的内存分配策略，在星形连接执行时，不同连接阶段的中间连接结果更新共享的事实表向量索引，事实表向量索引也作为最终查询结果输出集，从而能够实现对星形连接中间结果开销的最小化，提高硬件加速器有限板载高带宽内存的利用率. 我们在 Phi 协处理器连接技术研究中[17]也得到了类似的结论，PRO 算法的空间开销使其仅能支持低于 50%板载内存容量的数据集，NPO 次之，向量连接支持的数据集最大.

我们以代表性的内存数据库 Vector 5.0 为对象，通过 SQL 命令模拟维表投影属性压缩和向量映射操作（通过 SQL 命令将分组属性 distinct 值存储为字典表，通过连接操作生成模拟向量索引列）；星形向量连接操作采用我们开发的星形向量连接算法测试该查询处理阶段的执行时间，模拟数据库中的星形连接性能；我们在数据库中模拟向量聚集操作，通过增加一个名称为 *vector* 的列模拟向量索引，并通过 update 命令模拟生成按查询选择率和分组值创建的向量索引列，然后通过 SQL 命令模拟基于向量索引的分组聚集计算过程.

我们以 SSB 查询 Q3.1 为例说明 SQL 模拟的向量映射和向量聚集处理过程.

```
SELECT c_nation, s_nation, d_year, sum(lo_revenue)
AS revenue
FROM customer, lineorder1, supplier, date
WHERE lo_custkey=c_custkey
AND lo_suppkey=s_suppkey
AND lo_orderdate=d_datekey
AND c_region='ASIA' and s_region='ASIA'
AND d_year>=1992 and d_year<=1997
GROUP BY c_nation, s_nation, d_year
ORDER BY d_year asc, revenue desc;
```

以 customer 维表上的向量映射操作为例，模拟

向量映射过程如下：

```
CREATE TABLE Dvect(groups char(30), id integer
auto_increment);
CREATE TABLE dimvec(vec integer, id integer);
INSERT INTO vect(groups)
    SELECT distinct c_nation
    FROM customer
    WHERE c_region='ASIA';
INSERT INTO dimvec
    SELECT c_custkey, id
    FROM vect,customer
    WHERE c_region='ASIA' AND groups=c_nation;
```

其中，Dvect 表为维表向量索引字典表，采用自动增长 id 值；vect 表模拟向量索引，使用自动增长 id 属性作为向量索引的逻辑键值. 首先将 customer 表按查询相关谓词对分组列进行选择，并通过 distinct 命令对投影出的分组属性去重复值，存入字典表 Dvec 作为维表 customer 上的向量索引字典表；然后将 customer 表与向量索引字典表 Dvect 进行连接，将 customer 表主键和相应的字典表 id 插入到维向量索引表 dimvec 中，模拟按 customer 表主键逻辑映射到向量索引的过程. 向量索引的字典表压缩和创建过程可以通过一系列 SQL 命令模拟其处理过程，当开发独立的向量映射模块时，向量索引的创建与字典压缩可以通过哈希表一次性完成，能够进一步提高向量映射操作的性能.

在向量聚集处理阶段，我们使用事实表 lineorder 中增加的列 *vector* 模拟向量索引，通过 update 命令以 lo_partkey 列和值 *value* 来调节查询指定的选择率，并按查询分组大小以 lo_partkey 列为种子随机分配分组压缩编码. 基于向量索引，OLAP 查询转化为以向量索引列 *vector* 为过滤和分组属性的聚集操作.

```
UPDATE lineorder SET vector=(CASE WHEN lo_partkey<=
value THEN MOD(lo_partkey,150) ELSE -1 END);
SELECT vector, SUM(lo_revenue) AS revenue FROM lo
WHERE vector>=0 GROUP BY vector;
```

图 8 为 SSB 13 个查询的执行时间，Vector 5.0 为 TPC-H 测试性能最好的内存数据库①，其向量处理模型被内存数据库系统广泛采用，其查询性能在当前主流内存数据库产品中具有良好的代表性.

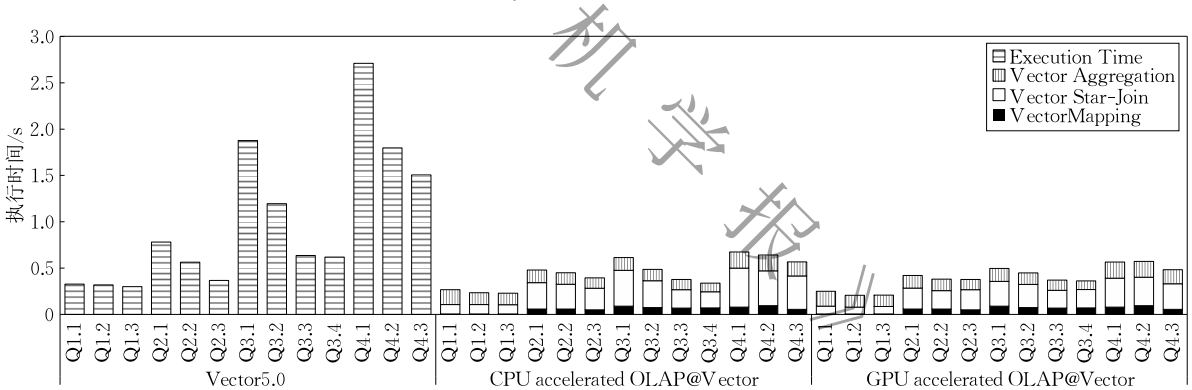


图 8 基于向量索引的 OLAP 性能

在 SSB 星形向量连接加速性能测试中，OLAP 查询执行时间由三部分模拟查询处理过程时间叠加而成. 从实验结果来看，Vector 5.0 在 SSB 的 13 个基准查询中平均执行时间为 1 s，而基于向量索引加速技术的 OLAP 查询平均执行时间为 0.44 s，性能提升了 1.2 倍. 从查询性能提升情况来看，Q4.1 性能提升最大达到 3.1 倍，性能提升最小的是 Q2.3，提升－7%. 总体来看，随着连接表数量的增加，星形向量连接操作的加速效果愈加显著；在连接表数量相同时，选择率高对应加速效果更好；当连接表数量较少且选择率较低时加速效果较差，如 Q1.x、Q2.2、Q2.3 平均性能提升仅 20%左右. 从基于向量索引的 OLAP 查询处理三个阶段性能来看，13 个查询中星形连接占查询时间比例的平均值为 55.3%，向量聚集占 33.4%，向量映射占 11.2%. 向量映射和向量聚集采用 SQL 模拟方式，查询执行时间相对较长，当开发独立的功能模块时会进一步提高这两个处理阶段的性能. OLAP 查询中星形连接是执行代价最高的操作，本文基于向量索引的星形连接加速技术能够有效地提高星形连接性能，从而提高 OLAP 查询处理的整体性能.

实验结果显示基于向量索引的 OLAP 查询处理性能显著优于 Vector 5.0，其主要原因是基于向

① TPC-H-Top Ten Performance Results-Non-Clustered Version 2 Results. http://www.tpc.org/tpch/results/tpch_perf_results.asp?resulttype=noncluster, 2017.8.3

量索引的向量连接性能优于基于传统哈希表的连接性能. 在基于向量索引的模拟 OLAP 查询处理过程中，生成向量索引的时间与生成哈希表相近，基于向量索引的分组聚集计算使用简单的压缩编码代替原始查询中的长字符串，哈希分组性能更高，而且基于向量索引的事实表索引访问相对于流水线记录迭代访问具有更高的性能，因此总体 OLAP 性能取得了较大的提升.

在 SSB 模式中，维表相对较小，因此星形向量连接操作通常为 in-cache 向量访问，多核处理器较大的 cache 相对于 GPU 的 SIMT 内存访问仍然具有较大的优势. OLAP 查询通常连接表数量较多但总选择率较低，较小的表首先执行连接操作并不断更新向量索引，较大表基于低选择率的向量索引执行连接操作，CPU 的分支预测及自动预取机制相对于 GPU 的 SIMT 机制能够更好地发挥其性能优势. SSB 13 个测试查询星形向量连接操作在 Intel Xeon E5-2699 v4 22 核 CPU 上的平均执行时间为 0.25 s，在 NVIDIA Tesla K80 GPU 上的平均执行时间为 0.37 s，体现了 CPU 在分支预测、索引访问等方面较好的性能. NVIDIA Tesla K80 GPU 集成了两个 G210 核心，当使用两个 GPU 核心执行星形向量操作时，外键列平均分配在两个 GPU 核心上，平均执行时间缩短为 0.21 s. 从价格上看，Intel Xeon E5-2699 v4 与 NVIDIA Tesla K80 GPU 价格基本相当，即 K80 GPU 上的星形向量连接操作性能整体略优于 Intel Xeon E5-2699 v4 处理器.

在高性能计算平台，配置 GPU 加速卡已成为主流趋势，本文所提出的星表向量连接加速技术将 OLAP 查询处理中计算代价最大的星形连接操作抽象为基于向量索引的星形向量连接操作，其较小的输入/输出向量索引和外键存储适合 GPU 内存计算特点，可以有效地将 OLAP 查询中的计算型负载从 CPU 分离，达到通过硬件设备可插件化的 OLAP 查询加速作用.

5　结束语

以众核处理器为代表的硬件加速器成为高性能计算平台的主流配置，也成为大数据时代实时 OLAP 分析处理新的硬件级加速引擎. 当前数据库实现与优化技术的研究重心开始从传统的 CPU 平台向新型众核处理器平台转移，其中关键问题既包括传统数据库查询处理如何在不同硬件特征的新型众核处理器上实现与优化，又包括在继承数据库完善的理论与系统框架的前提下如何融入新型处理器硬件加速技术.

本文以硬件级索引技术的视角探索了在既有的内存数据库系统框架下灵活地扩展硬件加速引擎的方法. 本文所提出的向量索引结合了数据仓库模式特点、数据分布特点、数据更新特点和多维数据模型特点，以简单的向量结构实现向量索引，并将数据仓库维表-事实表之间的主-外键参照完整性约束条件转换为维表、事实表向量索引之间的地址访问，通过索引技术简化了传统连接算法设计和代码执行效率，尤其重要的是，基于向量索引的向量连接算法使用简单的数组数据结构和数组地址访问操作，易于在新型众核处理器上实现. 向量索引可以看作是基于新型处理器硬件平台上的星形向量连接加速器，实现了将 OLAP 查询处理中计算代价最大的星形连接操作分离出数据库引擎，并且通过向量索引实现数据库引擎与硬件加速引擎的协同计算. 我们在实验中验证了基于向量索引的向量连接操作相对于传统连接操作的性能优化，并进一步通过模拟实验验证了基于向量索引和众核处理器硬件加速技术对现有内存数据库 OLAP 查询处理性能的加速作用，实验表明，通过简单的算法实现和基于已有的 SQL 处理模式能够实现内存数据库 Vector 5.0 上 56% 的性能提升.

我们在未来的工作中将进一步探索如何将向量索引技术引入数据库查询处理引擎，优化传统的查询处理技术，并结合 Xeon Phi、FPGA 等众核处理器技术加速 OLAP 性能，探索硬件自适应的向量连接实现技术，实现面向未来异构处理器计算平台开放的高性能内存 OLAP 计算框架.

参　考　文　献

[1] He B. Data management systems on future hardware: Challenges and opportunities//Proceedings of the International Conference on Data Engineering (ICDE). San Diego, USA, 2017: 1609

[2] Balkesen C, Alonso G, Teubner J, Özsu M T. Multi-core, main-memory joins: Sort vs. Hash revisited. Proceedings of the VLDB Endowment, 2013, 7(1): 85-96

[3] Balkesen C, Teubner J, Alonso G, Ozsu T. Main-memory hash joins on multi-core CPUs: Tuning to the underlying

hardware//Proceedings of the International Conference on Data Engineering (ICDE). Brisbane, Australia, 2013: 362-373

[4] Blanas S, Li Y, Patel J M. Design and evaluation of main memory hash join algorithms for multi-core CPUs//Proceedings of the SIGMOD Conference. New York, USA, 2011: 37-48

[5] Richter S, Alvarez V, Dittrich J. A seven-dimensional analysis of hashing methods and its implications on query processing. Proceedings of the VLDB Endowment, 2015, 9(3): 96-107

[6] Schuh S, Chen X, Dittrich J. An experimental comparison of thirteen relational equi-joins in main memory//Proceedings of the SIGMOD Conference. San Francisco, USA, 2016: 1961-1976

[7] Abadi D J, Madden S, Hachem N. Column-stores vs. row-stores: How different are they really?//Proceedings of the ACM SIGMOD International Conference on Management of Data. Vancouver, Canada, 2008: 967-980

[8] Barber R, Lohman G M, Pandis I. Memory-efficient hash joins. Proceedings of the VLDB Endowment, 2015, 8(4): 353-364

[9] Zhang Y, Zhou X, Zhang Y, et al. Virtual denormalization via array index reference for main memory OLAP. IEEE Transactions on Knowledge and Data Engineering, 2016, 28(4): 1061-1074

[10] Jha S, He B, Lu M, et al. Improving main memory hash joins on Intel Xeon Phi processors: An experimental approach. Proceedings of the VLDB Endowment, 2015, 8(6): 642-653

[11] Halstead R J, Absalyamov I, Najjar W A, Tsotras V J. FPGA-based multithreading for in-memory hash joins//Proceedings of the Conference on Innovative Data Systems Research (CIDR). Asilomar, USA, 2015

[12] Kaldewey T, Lohman G, Mueller R, Volk P. GPU join processing revisited//Proceedings of the International Workshop on Data Management on New Hardware (DaMoN). Scottsdale, USA, 2012: 55-62

[13] Pirk H, Manegold S, Kersten M. Accelerating foreign-key joins using asymmetric memory channels//Proceedings of the International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS@VLDB). Seattle, USA, 2011: 27-35

[14] Yuan Y, Lee R, Zhang X. The Yin and Yang of processing data warehousing queries on GPU devices. Proceedings of the VLDB Endowment, 2013, 6(10): 817-828

[15] He J, Lu M, He B. Revisiting co-processing for hash joins on the coupled CPU-GPU architecture. Proceedings of the VLDB Endowment, 2013, 6(10): 889-900

[16] Sodani A, Gramunt R, Corbal J, et al. Knights landing: Second-generation Intel Xeon Phi product. IEEE Micro. 2016, 36(2): 34-46

[17] Zhang Yu, Zhang Yan-Song, Chen Hong, Wang Shan. An MIC Phi oriented in-memory OLAP foreign key join algorithm. Journal of Software, 2017, 28(3): 490-501(in Chinese)
(张宇, 张延松, 陈红, 王珊. 一种基于众核架构 Phi 协处理器的内存 OLAP 外键连接算法. 软件学报, 2017, 28(3): 490-501)

ZHANG Yan-Song, Ph. D., associate professor. His current research interests include data warehouse and main-memory database.

ZHANG Yu, Ph. D., Senior Engineer. Her research interests include data warehouse, GPU database mining.

WANG Shan, professor, Ph. D. supervisor. Her research interests include database, data and information retrieval.

Background

As new hardware techniques such as multicore processor and many core processor come to be main-stream high performance computing platforms, the traditional query optimization techniques are no longer adaptive to the new hardware with new features. In recent years, the academic and industry focused on how to accelerate database query processing with advanced multicore and many core architecture processors to meet the requirements of big data real-time analytical processing. As join is the most important and complex operation in relational databases and dominates the whole query processing performance, how to accelerate join performance with hardware-conscious designs remains hot topic which involved rich researches as more and more hardware are introduced for accelerate query processing performance. Most of the researches limited in join performance perspective to prove whether hardware-conscious join or hardware-oblivious join to be better, while star join operation is more important for data warehousing workloads with multidimensional model. The existed conclusions of join performance cannot be directly used for star join research for

the different materialization strategies of different join implementations dominate the star join performance. In this paper, we focus on star join algorithm design and performance evaluation, the implementation techniques for heterogeneous platforms with x86 architecture CPU and many core architecture NVIDIA GPU. Moreover, we further research on how to apply hardware accelerated star join operation into nowadays in-memory databases. We also design experiments with widely accepted benchmarks to evaluate star join performance with different implementations and platforms, the simulation experiment shows how to integrate hardware accelerated star join implementation into databases.

This work were supported by the National Natural Science Foundation of China (61732014, 61772533) and the Beijing Natural Science Foundation (4192066). The target of research is to exploit how new hardware devices influence the database architectures, system designs and query processing optimizations. We have published more than ten related papers and patents in hardware accelerated in-memory analytics domain, and this work concerns the database system integration perspective to exploit how to design hardware accelerated module and how to combine the traditional database architecture and new query processing accelerator together.