软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(3):883−895 [doi: 10.13328/j.cnki.jos.005446]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于向量引用 Platform-Oblivious 内存连接优化技术*

张延松[1,2,3], 张 宇[4], 王 珊[1,2]

[1](数据工程与知识工程教育部重点实验室(中国人民大学),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)

通讯作者: 张宇, E-mail: yuzhang@cma.gov.cn

摘 要: 以 MapD 为代表的图分析数据库系统通过 GPU、Phi 等新型众核处理器来支持高性能分析处理,在面向复杂数据模式时,连接操作仍然是重要的性能瓶颈.近年来,异构处理器逐渐成为高性能计算的主流平台,内存连接性能的研究从多核 CPU 平台扩展到新兴的众核处理器,但众多的研究成果并未系统地揭示连接算法性能、连接数据集大小、硬件架构之间的内在联系,难以为未来异构处理器平台的数据库提供连接平台优化选择策略.以面向多核 CPU、Xeon Phi、GPU 处理器平台的内存连接优化技术为目标,通过优化内存哈希表设计,实现以向量映射替代哈希映射操作,消除哈希代价对内存连接算法的影响,从而更加准确地测量内存连接算法在多核 CPU 的 cache 大小、Xeon Phi 的 cache 大小、Xeon Phi 的并发多线程、GPU 的 SIMT(单指令多线程)机制等硬件相关因素影响下的性能特征.实验结果表明,缓存与并发多线程机制是提高内存连接算法性能的重要影响因素.缓存机制对于满足 cache 大小的连接操作具有性能优势,而 GPU 的并发多线程机制则在较大表的连接操作中具有较高的性能,Xeon Phi 则在满足其 L2 cache 大小的连接操作中具有最高性能.实验结果揭示了内存连接操作性能与异构处理器硬件特性的联系,为未来异构处理器平台内存数据库查询优化器提供了优化策略.

关键词: 内存连接操作;哈希连接;向量映射;异构处理器平台

中图法分类号: TP311

中文引用格式: 张延松,张宇,王珊.基于向量引用 Platform-Oblivious 内存连接优化技术.软件学报,2018,29(3):883−895. http://www.jos.org.cn/1000-9825/5446.htm

英文引用格式: Zhang YS, Zhang Y, Wang S. Vector referencing oriented platform-oblivious in-memory join optimization technique. Ruan Jian Xue Bao/Journal of Software, 2018,29(3):883−895 (in Chinese). http://www.jos.org.cn/1000- 9825/5446.htm

Vector Referencing Oriented Platform-Oblivious In-Memory Join Optimization Technique

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

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

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

* 基金项目: 国家自然科学基金(61732014, 61772533); 国家高技术研究发展计划(863)(2015AA015307); 中央高校基本科研业务费专项资金(16XNLQ02)

Foundation item: National Natural Science Foundation of China (61732014, 61772533); National High Technology Research and Development Program of China (863) (2015AA015307); the Basic Research Funds in Renmin University of China from the Central Government (16XNLQ02)

本文由基于图结构的大数据分析与管理技术专刊特约编辑林学民教授、杜小勇教授、李翠平教授推荐.

收稿时间: 2017-07-31; 修改时间: 2017-09-05; 采用时间: 2017-11-07; jos 在线出版时间: 2017-12-05

CNKI 网络优先出版: 2017-12-06 15:23:24, http://kns.cnki.net/kcms/detail/11.2560.TP.20171206.1522.010.html

[3](National Survey Research Center at Renmin University of China, Beijing 100872, China)
[4](National Satellite Meteorological Centre, Beijing 100081, China)

Abstract: Graph analysis database such as MapD employs the emerging manycore architecture GPU and Phi processors to support high performance analytical processing, where the join operation is still the performance bottleneck when facing complex data schemas. In recent years, as heterogeneous processors come to be main-stream high performance computing platforms, the researches of in-memory join performance extend the focuses from multicore to the emerging manycore platforms. However those efforts have not uncover the inner relationships among join algorithm performance, join dataset size and hardware architectures, and cannot provide sufficient join selection strategies for databases under the future heterogeneous processor platforms. This paper targets in-memory join optimization techniques on multicore, Xeon Phi and GPU processor platforms. By optimizing hash table design, this work uses vector mapping instead of hash mapping to eliminate the hashing overhead effects for performance, so that the in-memory join performance characteristics influenced can be measured by hardware factors such as multicore cache size, Xeon Phi cache size, Xeon Phi simultaneous multi-threading mechanism, and GPU SIMT (single instruction multiple threads) mechanism. The experimental results show that caching and simultaneous massive-threading mechanism are key factors to improve in-memory join algorithm performance. Caching mechanism performs well for cache fit join operations, the simultaneous massive-threading mechanism of GPU does well for big table joins, and Xeon Phi achieves the highest performance for L2 cache fit joins. The experimental results also exploit the relationship between in-memory join performance and heterogeneous processor hardware features, and provide optimization policy for in-memory database query optimizer on future heterogeneous processor platforms.

Key words: in-memory join operation; Hash join; vector mapping; heterogeneous processor platform

社交网络分析和大数据可视化,是图结构大数据分析的重要组成部分.MapD[1]是一种面向大规模社交网络分析性能压力而开发的GPU数据库系统,主要用于Twitter社交网络数据实时分析处理,其研发的初衷是传统数据库技术难以支持大规模图数据实时分析处理需求,其独特的 GPU 数据库查询处理引擎相对于传统基于 x86 处理器架构的数据库查询处理引擎有显著的性能优势,成为当前大数据实时分析处理领域的代表性技术.从图数据分析处理需求来看,社交网络分析具有典型的多维分析处理特征,在数据可视化[2]中,需要以 GIS 数据、时间数据、类别数据等信息构建多维分析处理模型,支持从不同维度、不同层次、不同视角的社交网络分析处理.而在面对复杂模式下的分析处理时,连接是其中执行代价较高的操作,连接性能也是数据库综合性能的重要指标之一.从 MapD 数据库技术特点来看,其发展揭示了新兴数据库技术充分利用新硬件技术发展来克服传统数据库性能瓶颈的设计思想,为数据库查询优化技术研究提供了宝贵的参考.

从硬件技术发展趋势来看,处理器技术经历了从单核到多核再到众核架构的革新.在标志着高性能计算的TOP 500[3]和绿色计算的GREEN500[4]榜单中,包含众核协处理器 Xeon Phi 和 GPU 的异构处理器平台已经成为主流配置,在高性能计算 HPC 领域得到了广泛应用,数据库也面临从 x86 多核 CPU 平台向支持新一代众核处理器的异构处理器平台的技术升级.但硬件结构变化在带来新硬件性能红利的同时,也增加了数据库查询优化的技术障碍和系统结构的复杂性,数据库在异构处理器平台上的应用仍然进展缓慢.

连接是数据库执行代价最大的操作之一,连接优化技术一直是数据库性能优化研究的一个热点问题.hardware-conscious 还是 hardware-oblivious,哪种算法设计有更高的性能?研究中的 hardware 泛指 x86 多核处理器,hardware-conscious 算法设计思想的目标是追求最高的性能,其指导思想是充分考虑相关硬件的特性对连接算法进行深度优化设计与调优;而 hardware-oblivious 算法设计思想的目标是追求简单通用的实现框架和较高的硬件自适应性能,其指导思想是利用硬件具有的共性设计连接算法,利用硬件性能的支持自动达到相对较高的性能.内存数据库核心的连接算法广泛采用 hardware-conscious 优化设计思想.传统的 x86 架构处理器技术采用以 cache 为中心的设计思想,40%以上的晶体管用于制作 cache 单元,处理器结构复杂,核心数量增长缓慢,价格昂贵,系统能耗较高.其上的连接操作主要以中、低度并行算法设计和以 cache 为中心的数据访问优化技术为基础,性能受制于 x86 平台有限的 LLC(last level cache,最后一级 cache)容量和较低的核心数量.GPU 采用与 x86 处理器不同的架构,数以千计的处理核心支持更多的硬件级线程,提供强大的并行处理能力,GPU 主要通过 SIMT(single instruction multiple threads,单指令多线程)机制,以大量并发线程访问内存,并通过硬件级的线程切

换掩盖内存访问延迟,对连接操作数据集大小的约束较小.Xeon Phi 是一种兼容 x86 指令集的众核处理器架构,它扩展 AVX 指令集到 512 位,增强了向量处理能力.当前,新型的 KNL Phi 至强融核处理器集成了 72 个核心,每核心支持 4 线程,以并发多线程机制提高并行处理性能.Phi 采用共享 L2 cache 结构,其板载的 16GB 高带宽内存可以配置为 cache 使用,以增强的 cache 机制提高缓存性能,解决了传统多核 CPU 在 cache 优化中因 LLC 容量不足所导致的数据集大小约束问题.从处理器硬件总体特征来看,缓存与并发多线程仍然是优化内存访问性能的关键技术,因此在面对异构处理器平台时,我们需要解决以下几个关键问题:何种连接算法能够更好地适应异构处理器平台?连接算法在不同的处理器平台的性能特征是什么?数据库如何根据负载特征优化选择处理器平台?

内存哈希连接是学术界近年来证明的最优内存连接算法[5],但对于新型 Xeon Phi 处理器及 GPU 处理器,其哈希表结构、哈希映射函数等方面的复杂性[6]增加了算法在新硬件平台的实现难度,也导致哈希连接算法的性能有很大不确定性.

本文的贡献主要体现在 3 个方面.

- 通过对当前主流内存连接算法的分析构建极限性能哈希连接模型,减少哈希连接算法性能的影响因素,构建适应异构处理器平台的 platform-oblivious 连接算法;
- 通过多核 CPU、Xeon Phi、GPU 处理器平台的内存连接算法性能测试,揭示不同处理器平台上的内存连接算法性能特征,连接数据集大小、LLC 大小与连接性能之间的关系,以及不同 Benchmark 与处理器连接性能之间的适应性;
- 通过异构处理器连接性能实验结果,分析提出了面向异构处理器平台的内存连接算法选择策略,通过连接性能曲线界定算法性能优势与劣势区间,为连接算法选择提供依据.

本文第 1 节对异构处理器平台内存连接相关技术进行对比分析.第 2 节、第 3 节在对当前主流内存连接算法分析的基础上构建 platform-oblivious 连接算法.第 4 节给出相关内存连接算法在多核 CPU、Phi、GPU 协处理器上的实验性能.最后对本文加以总结.

1 相关工作

连接操作一直是数据库查询优化的核心.连接操作与数据库的模式优化技术、数据分布、负载特征、索引技术等数据库软特征密切相关,具有联结其他关系操作的枢纽作用.因而,连接操作优化不是单一的优化技术,它与整个数据库的查询优化技术紧密结合.硬件方面的相关因素包括 cache 结构、TLB entry 数量、NUMA 结构、SIMD 宽度、核心数量、超线程数量等,软件方面的相关因素包括哈希表结构、并发访问控制技术、数据压缩技术、哈希探测算法、哈希函数等,软件优化策略根据硬件参数进行优化设计,以减少 cache miss 为目标.

近年来,内存数据库系统成为主流,学术界对内存连接算法的优化工作不断深化,对内存连接算法性能的比较成为热点问题,但结论仍难以确定.随着异构处理器平台的普及,连接操作在异构处理器平台上的性能如何,以及在哪种处理器平台上连接操作性能最优,仍是需要解答的问题.

内存数据库连接优化的 hardware-conscious 还是 hardware-oblivious 两种技术路线之争持续数年,从基础 CPU 平台[7,8]扩展到 NUMA 平台[9,10]以及新型处理器平台,其结论也在不断地刷新.这种现象背后深层次的原因是数据库在核心数据结构及算法设计上的复杂性与多样性所导致的优化技术难以定量化,例如在哈希连接中,哈希结构包括 Chained hashing[11]、Linear probing、Robin Hood Hashing[12]、Cuckoo Hashing[13]、Quadratic Probing 等多种实现技术,哈希函数也有多种不同的实现技术,以及结合位图索引技术的新型哈希表[14]等不同的实现方法[6],因此,对于什么是最好的内存哈希连接算法,现阶段仍然难以获得准确的答案[5].

从综合结论来看,学术界倾向于 hardware-conscious 类型 Radix 分区哈希连接算法,其基础硬件假设是 LLC 较小而连接表较大,并且 Radix 分区阶段所需要的内存临时存储空间不受限制.随着处理器技术的发展,这些基础硬件假设可能面临着巨大变化,例如新一代至强融核处理器 KNL Phi 将 16GB 的板载内存用作新的 LLC,解决了 LLC 容量问题、众核处理器板载 HBM 高带宽内存容量相对系统内存有限等,这些硬件特性的变化改变了

hardware-conscious 和 hardware-oblivious 算法设计的平衡点,也导致未来计算平台上的连接性能结论可能再次反转.

从异构处理器发展趋势来看,将面向 CPU 类型的 hardware-oblivious 算法扩展为面向异构处理器的 platform-oblivious 内存连接算法具有现实的意义.面向异构处理器,内存哈希表的指针结构、锁机制、哈希表结构、哈希探测方法等方面的设计除性能因素外,还需要考虑数据结构与算法跨平台的兼容性与适应性.相对于复杂的内存哈希表结构,文献[15]提出的 AIR 数组地址引用技术和文献[5]中使用的 array join 算法使用数组代替哈希表结构,使用键值-地址映射代替哈希连接操作,不仅简化了内存连接算法设计,而且所使用的数组数据结构和数组地址访问技术还具有良好的平台适应性,具备了 platform-oblivious 的基本特征,我们在本文中将进一步探讨哈希连接与基于数组地址映射的连接算法之间的融合技术.

早期的 GPU 数据库实现技术研究中,哈希连接操作采用内存 Radix 分区,然后加载到 GPU 内存,通过 nested loop 或二分查找方法进行连接[16].由于分区操作需要较高的内存消耗,GPU 上的哈希连接通常采用简单的无分区哈希连接算法[17,18].融核 APU 架构将 CPU 与 GPU 集成到相同的芯片上,从而消除 CPU 与 GPU 之间的 PCIe 传输代价,并能够支持更细粒度的 CPU 与 GPU 之间的协同查询处理[19],提高 CPU 与 GPU 的利用率,提升总体性能.

至强融核 Phi 协处理器采用与 CPU 兼容的 x86 指令集和类似的体系结构,但 Phi 采用 AVX512 SIMD 指令集与当前 CPU 平台不兼容.文献[20]将文献[8]的开源哈希连接算法改写为 Phi 协处理器版本,通过 AVX512 指令优化了哈希函数计算和哈希桶加载.文献[20]的研究扩展了 Phi 协处理器 512 位 SIMD 指令的应用范围,通过 SIMD 实现所有的基础操作符,从而更加充分地利用了 Phi 协处理器 512 位 SIMD 指令的高性能,提升了 Radix 分区哈希连接性能.进一步地,内存哈希连接技术研究扩展到 FPGA 平台[22],通过新兴的硬件加速传统的连接操作性能成为主流技术趋势.

综上所述,从连接算法实现层面来看,无分区哈希连接和 Radix 分区哈希连接仍是主流技术,并逐渐扩展到 Phi、GPU、FPGA 等新兴处理器平台,面向异构处理器不同硬件特性而进行的调优和深度优化设计是连接优化技术研究的一个重要趋势.在连接算法设计层面,以数据库模式特点和负载特点为基础的 AIR 和 array join 算法简化了复杂的哈希机制,具有更好的异构处理器平台算法迁移能力.从异构处理器平台应用层面看,探索连接算法在不同类型处理器平台上的极限性能以及连接算法在不同负载下的性能特征,是有效构建面向异构处理器平台连接操作代价模型的基础.

2 基于向量引用的哈希连接优化技术

向量处理是内存数据库中典型的优化技术,如图 1 所示,当前代表性的向量处理技术主要分为 3 种类型.

- SIMD 向量计算.以 CPU 寄存器或 Phi 向量处理器宽度为单位执行以向量为粒度的计算,提高指令执行效率.SIMD 计算可应用于哈希函数计算、哈希桶定位等计算过程[20];
- 向量引用.AIR[15]与 array join[5]算法使用向量代替哈希表,使用向量引用(vector referencing)代替哈希探测,通过简单的向量映射代替复杂的哈希映射机制;
- 向量化处理.以适合 L1 cache 大小的向量[23]为查询处理粒度,优化查询中间结果存储访问代价,用于优化流水线查询处理性能.

从向量处理技术的硬件依赖性来看,SIMD 依赖于向量寄存器宽度,向量化处理依赖于 L1 cache 大小,向量引用依赖向量相对于 LLC 的大小.

向量引用技术可以看做是位图索引、主外键参照完整性约束与哈希技术的结合,极大地简化了哈希表结构,我们将进一步讨论向量引用技术的适用范围及性能约束条件.

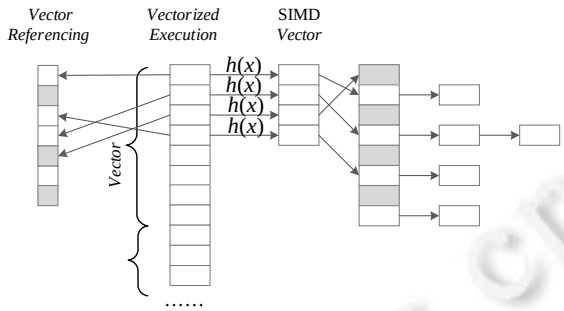

Fig.1 Vectorized processing oriented join optimization techniques

图 1 向量处理连接优化技术

2.1 向量引用技术适用范围

向量引用技术使用向量作为连接数据结构,向量单元下标作为向量的隐式主键,在数据库主外键参照完整性约束条件的支持下实现外键值与向量单元地址的映射,将传统数据库基于值匹配的连接操作简化为基于值-地址映射的内存访问.

- 可行性

向量引用技术在数据库主键约束的基础上增加了以向量单元下标作为主键值的约束条件,强化了对数据库模式的要求.在数据仓库中广泛使用的代理键(surrogate key)采用连续自然数序列作为维表主键,满足了向量引用技术的要求.在 array join 算法[5]的设计中,使用 PRIMARY KEY AUTOINCREMENT 数据库自动增长序列满足主键的地址映射约束;在 AIR 算法[15]设计中,则采用数组存储模型和原位更新机制来保证维表主键的地址映射约束.

当主键表使用不能直接映射为记录地址的主键时,可以为主键表创建新的代理键或 AUTOINCREMENT 类型主键,并在外键表创建新的外键,替代原有的主键与外键列完成基于向量引用的连接操作.

- 局限性

向量引用技术有效性的前提是保证连接表主键的地址映射关系,尤其是在更新操作中需要保持连接表主键的连续性.从应用场景来看,数据仓库历史数据只读性和维表与事实表之间的主外键参照完整性约束保证了数据 insert-only 的应用特点,简化了更新操作,保证了维表主键的地址映射特征.从主键使用模式来看,主键采用连续数列增加了地址语义,同时消除了逻辑语义,因此主键表上的 update 操作并不更新主键取值,不影响主键的地址映射关系.为避免 MVCC 等机制造成的异位更新破坏主键表记录原始位置信息,Vectorwise 数据库采用 Positional Delta Trees 技术[23]存储更新数据,通过内存位置访问更新数据提高更新数据访问效率.文献[24]使用逻辑代理键索引机制放松了主键值与位置的强约束关系,更好地适应 MVCC 更新机制下的数据库管理机制.对于主键表记录被删除时导致被参照的向量中产生的相应空洞,当删除记录数量达到一定阈值时,通过重新组织主键表主键值分配以及相应的外键表外键更新重建向量引用约束.

向量引用技术在主键上增加了额外的约束条件,不适合存在频繁记录删除操作的应用场景,但数据仓库的 OLAP 查询处理应用场景能够显著减少更新操作的代价.

- 性能损失

向量引用技术是在查询中采用定长向量作为连接数据集,向量下标用作主键值,相对于哈希表节省了键值存储空间.在 OLAP 应用中,维表向量用于表示查询分组属性,通常只包含数量较少的分组值,通过字典表压缩能够显著减少向量宽度.文献[25]所采用的基于数据压缩技术的连接优化主要面向静态数据压缩,文献[15]则对维表上的选择和投影操作结果进行动态字典表数据压缩,根据对数据仓库 Benchmark SSB 和 TPC-H 的分析,1 字节(支持 2^8–1 个分组属性压缩编码)和 2 字节(2^{16}–1 个分组属性压缩编码)宽度的向量能够支持绝大部分 OLAP 查询处理需求,从而降低了向量整体空间消耗.但从选择率角度来看,哈希表只存储满足选择条件的记录,而向

量则需要存储固定大小的向量,在选择率较高的查询中,向量的存储空间低于哈希表;而在低选择率时,哈希表存储空间少于向量存储.因此在低选择率查询中,向量引用技术的存储效率会有一定损失.

从相关研究[24]的实验测试结果来看,向量引用技术(AIR 算法)在向量大小低于 LLC 容量时表现出稳定的性能,即,查询选择率对向量引用性能的影响存在一个不敏感区间,我们将在后面的实验中深入分析选择率对连接算法的影响.

综上所述,向量引用技术相对于哈希连接是一种定制化的连接实现技术,主要应用场景为:具有主外键完整性约束条件,主键表主要为 insert-only 类型的更新,delete 和 update 类型更新操作较少,连接表相对较小.与哈希连接算法相比,向量引用技术不是通用的等值连接操作,但能够较好地适应 OLAP 应用特点,并极大地简化了连接算法设计.

2.2 向量引用技术性能分析

图 2 显示了哈希连接与向量引用算法示意图.影响哈希连接算法的性能因素相对较多:在哈希表构建阶段,哈希表结构、哈希映射方式、哈希桶并发访问控制有不同的实现技术,对哈希连接算法的性能产生了不同的影响.在哈希探测阶段,哈希函数计算、定位哈希桶、键值比较、溢出桶查找等操作产生较多的 CPU cycle,如图 2 所示,SIMD 技术可以用于加速哈希计算过程.向量引用技术可以看做是一种特殊的哈希连接操作,与连接表 R 等长的向量替代了哈希表,主键映射机制消除键值存储,连接表记录与向量单元的值-地址映射消除哈希函数计算代价和并发访问控制代价.同样,在哈希探测阶段也消除了哈希函数计算、定位哈希桶、键值比较以及溢出桶查找等计算代价,消除了不同哈希表结构、不同哈希映射函数、不同哈希桶并发访问机制、SIMD 优化技术等因素对连接性能的影响,使连接操作性能的影响因素缩减为唯一的向量访问代价,简化了连接操作的代价模型.在 CPU 和 Xeon Phi 处理器上,主要取决于向量相对于 LLC 的大小;在 GPU 处理器上,则取决于 SIMT 的并发内存访问性能.

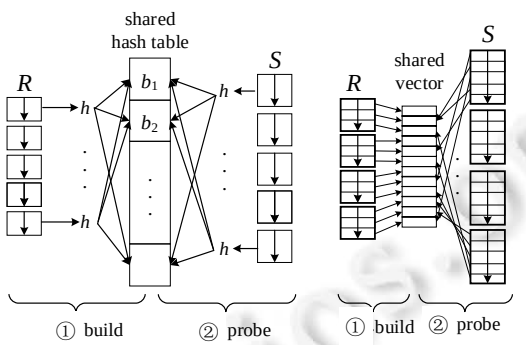

Fig.2 Hash join and vector referencing

图 2 哈希连接与向量引用

在数据仓库的 OLAP 应用中,向量大小取决于维表大小.表 1 列出当前代表性的 OLAP 应用 Benchmark TPC-H,SSB 和 TPC-DS 中不同数据集大小对应的各维表向量大小(向量宽度为 1 字节).在 300GB 以下数据集中,各维表向量均不超过当前最新 CPU 的 LLC 大小(E7-8890 v4,60MB LLC);在 10TB 数据集中,TPC-H 最大维表向量大小接近 2GB,仍远小于最新 Xeon Phi 7290 16GB 的板载内存(可配置为 cache)容量;在更具有代表性的 TPC-DS 中,维表数量较多但记录数量较少,且随数据量快速增长而缓慢增长,因此,维表向量大小维持在较小的水平.向量大小低于 LLC 时,向量引用技术具有较好的性能[24];而且当数据分布倾斜时,即使较大的向量也能产生较好的数据局部性.在现代处理器大容量 LLC 的支持下,算法可以达到连接算法的最优性能.

Table 1 Vector sizes in database Benchmarks (MB)

表 1 数据库 Benchmark 中的向量大小 (MB)

Benchmark	Table	100GB	300GB	1TB	3TB	10TB
TPC-H	CUSTOMER	14.31	42.92	143.05	429.15	1430.51
	PART	19.07	57.22	190.73	572.20	1907.35
	SUPPLIER	0.95	2.86	9.54	28.61	95.37
	NATION	0.000 02	0.000 02	0.000 02	0.000 02	0.000 02
	REGION	0.000 005	0.000 005	0.000 005	0.000 005	0.000 005
SSB	CUSTOMER	2.86	8.58	28.61	85.83	286.10
	PART	1.46	1.76	2.09	2.39	2.73
	SUPPLIER	0.19	0.57	1.91	5.72	19.07
	DATE	0.002	0.002	0.002	0.002	0.002
TPC-DS	call_center	0.000 03	0.000 03	0.000 04	0.000 05	0.000 05
	catalog_page	0.02	0.02	0.03	0.03	0.04
	customer	1.91	4.77	11.44	28.61	61.99
	customer_address	0.95	2.38	5.72	14.31	30.99
	customer_demographics	1.83	1.83	1.83	1.83	1.83
	date_dim	0.07	0.07	0.07	0.07	0.07
	household_demographics	0.01	0.01	0.01	0.01	0.01
	income_band	0.000 02	0.000 02	0.000 02	0.000 02	0.000 02
	item	0.19	0.25	0.29	0.34	0.38
	promotion	0.001	0.001	0.001	0.002	0.002
	reason	0.000 05	0.000 06	0.000 06	0.000 06	0.000 07
	ship_mode	0.000 02	0.000 02	0.000 02	0.000 02	0.000 02
	store	0.000 4	0.000 8	0.001 0	0.001 3	0.001 4
	time_dim	0.08	0.08	0.08	0.08	0.08
	warehouse	0.000 01	0.000 02	0.000 02	0.000 02	0.000 02
	web_page	0.002	0.002	0.003	0.003	0.004
	web_site	0.000 03	0.000 03	0.000 04	0.000 05	0.000 05

OLAP 应用中:典型的上卷、下钻操作对应不同选择率的连接查询处理任务,哈希连接算法需要为每个查询创建不同大小的哈希表,并在查询结束后释放空间,产生内存碎片;而向量引用技术则可以在上卷、下钻操作中使用相同的维表向量,不同查询只需要更新维表向量内容,提高了内存空间的使用效率.

3 向量引用技术的 Platform-Oblivious 特征

内存哈希连接算法向新型处理器平台迁移,是当前学术界重要的研究方向之一.将内存哈希连接算法[8]迁移到 GPU[19]、Xeon Phi[20]和 FPGA[21]处理器平台,并结合处理器平台硬件特性实现 hardware-conscious 算法设计,以充分发挥新型处理器的性能.随着异构处理器成为高性能计算的主流平台,platform-oblivious 算法设计无疑更加具有吸引力.但在 CPU 平台上,hardware-oblivious 的无分区哈希连接算法在迁移到异构处理器平台时仍然需要面向处理器硬件特性进行优化,如链接哈希表存储访问优化(GPU,FPGA)、哈希探测中计算过程的 SIMD 优化(Xeon Phi)、键值比较分支判断及共享内存访问优化(GPU)等,因而成为具有 platform-conscious 特性的算法设计.硬件结构的变化及优化技术的研究,使异构处理器平台上的连接操作性能影响因素增多,难以准确地评价连接性能.

Platform-Oblivious 连接算法的主要特征是数据结构与算法设计对不同结构的处理器平台具有自动适应性,即,不需要面向不同处理器硬件特征进行过多的优化设计.相对于哈希连接算法,向量引用技术使用最简单的向量(数组)数据结构,消除指针和动态地址分配,在 GPU 和 FPGA 平台更加易于实现;在连接算法上,以键值-地址映射访问,消除连接时哈希映射、键值比较、溢出桶查找等额外的复杂计算代价,将连接性能影响因素缩小为向量访问性能,自动利用 CPU、Phi 的多级 cache 机制和 GPU 的 SIMT 机制,达到自动优化的目标.对于 CPU 与 Phi、较大的 cache 和相对较少的线程在多线程并行处理时,通过线程同步机制实现线程间连接结果集的合并;对于 GPU,线程间可以通过 shared memory 进行数据共享访问.在多表连接操作中,我们采用与连接表等长的向量存储连接结果,以预分配 GPU 内存策略适应 OLAP 上卷、下钻等操作对应不同选择率连接操作结果集的存储空间复用问题.

结论 1. 总体上,向量引用技术简单的向量结构和向量地址映射访问保证了其在异构处理器平台上的实现技术最小化硬件结构差异,通过处理器常规的技术实现 platform-oblivious 连接算法.

4 实验结果与分析

本文实验平台为一台 DELL PowerEdge R730 服务器,配置有 2 块 Intel Xeon E5-2650 v3@2.30GHz 10 核 CPU,共 20 个核心,40 个线程,LLC 大小为 25MB,512 GB DDR3 内存.操作系统为 CentOS,Linux 版本为 2.6.32-431. el6.x86_64,gcc 版本为 4.4.7.服务器配置一块 Intel Xeon Phi 5110P@1.053 GHz 协处理器,其中集成了 60 个核心,每核心支持 4 线程,共计 240 线程,Phi 5110P 协处理配置有 8GB 内存,协处理器内置操作系统的 Linux 版本为 2.6.38.8+mpss3.3,gcc 版本为 4.7.0;服务器还配置一块 NVIDIA K80 GPU,集成了 2 个 GK210 核心,每个 GK210 核心集成了 2 496 个流处理器,共计 4 992 个流处理器,shared memory 为 128KB.

实验中使用文献[8]提供的开源内存哈希连接算法,NPO 和 PRO 分别表示 hardware-oblivious 的无分区哈希连接算法和 hardware-conscious 的 Radix 分区哈希连接算法,我们基于文献[24]的方法增加了 AIR 连接算法,在 Phi 平台使用文献[20]的开源哈希连接算法,并且实现了基于 CUDA 的 GPUAIR 连接算法.

实验中分别使用文献[8]中的 workload A 和 workload B 模拟小表-大表连接和两个大表连接的情况;使用 SSB、TPC-H 和 TPC-DS 连接数据集测试 platform-oblivious 算法 AIR 面向不同应用负载时的性能.进一步地,使用文献[24]中相对于 CPU 各级 cache 大小的细粒度连接数据集测试 AIR 算法在 CPU、Phi 和 GPU 平台的性能特征,分析 AIR 算法性能与 cache 大小及 SIMT 机制之间的关系.

4.1 CPU端连接算法性能对比分析

我们首先使用 workload A 和 workload B 分析了 AIR、NPO 和 PRO 算法不同执行阶段的性能.workload A 中,两个连接表分别为 16M 行和 256M 行,AIR 算法采用宽度为 1 字节的连接向量,因此具有较好的 cache 局部性.workload B 的两个连接表都为 128M 行,模拟两个大表的连接.

图 3 显示了 AIR、NPO 和 PRO 算法的执行时间(单位为 cycle)分布.PRO 算法中,Radix 分区代价较高,但分区后哈希探测性能最高;NPO 算法在大表连接时构建哈希表代价明显升高,哈希探测性能最差;AIR 算法采用的定长向量结构降低了向量构建代价,向量地址访问性能介于 NPO 与 PRO 之间,但整体性能较高.

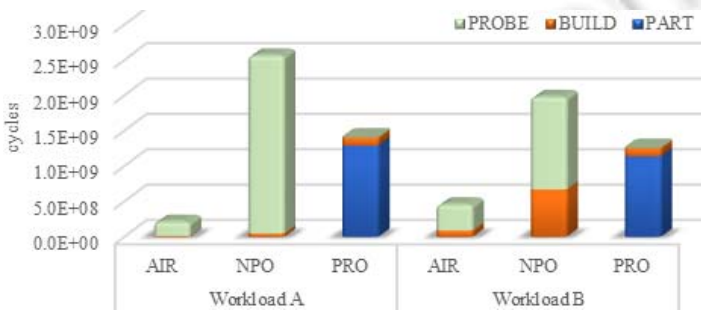

Fig.3 Breakdown time of join operations

图 3 连接操作时间分解

在第 2.2 节的基础上,对向量引用技术性能进行更细粒度分析,我们测试连接表 R 所产生的向量大小相对于各级 cache 大小不同比例时的连接性能.设置表 S 行数为 600 000 000 行,与 SF=100 时 TPC-H 事实表记录数相同.我们以 AIR 算法向量大小为基准,设置向量宽度为 1 字节,然后按各级 cache 大小 25%,50%,…,200%递增比例设置表 R 的行数.例如,25%×L1 表示代理向量大小为 25%×32KB(L1 Cache size),对应 R 表长度为 8 192 行;L3 cache 以 2.5MB 的 cache slice 为单位,最大为 cache slice 的 20 倍,即 LLC 总容量的 2 倍;之后,再按外键表 S 行数的 10%,20%,…,100%比例设置连接表 R 的行数,扩展文献[8]中的 2 个连接负载,从而获得更加全面的连接算

法性能特征.连接性能单位为平均每记录的纳秒数(ns/tuple).

图 4 显示了在 CPU 平台上,向量引用连接算法 AIR 与两种哈希连接算法 NPO 与 PRO 的性能对比.整体来看,hardware-conscious 的 Radix 分区 PRO 算法性能稳定,其通过分区操作将较大的连接表划分为适合 cache 大小的分区,从而保证哈希表的创建与探测过程有较好的 cache 局部性.hardware-oblivious 的无分区哈希连接算法 NPO 采用简单的共享哈希表模式,其性能主要受哈希表相对各级 cache 大小的影响:当哈希表小于 LLC 时,NPO 算法优于 PRO 算法;当哈希表超过 LLC 时,哈希探测延迟增长,连接性能逐渐低于 PRO 算法.AIR 算法可以看做是哈希连接算法面向 OLAP 应用模式的定制化技术,整体性能特征与 NPO 类似,在向量小于各级 cache 时,表现为自适应的高性能;在向量超过 cache 大小时,增加了连接操作延迟.但 AIR 算法基于压缩技术的向量结构在向量大小及向量内存访问延迟方面相对于哈希表链接结构具有更好的性能,因而性能较 NPO 有较大的提升,而且平均连接执行时间也低于面向 cache 分区优化的 PRO 算法.

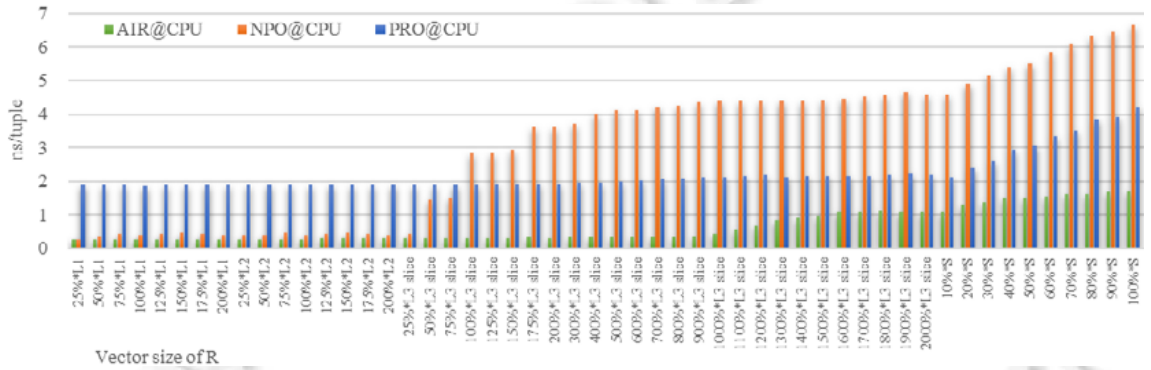

Fig.4 Performance characteristics of different join algorithms on CPU

图 4 CPU 端连接算法性能特征

对于第 2.1 节讨论的定长向量在低选择率查询时的性能损失问题,从图 4 中可以看到:当向量大小低于 LLC 大小时,AIR 算法具有较高的性能.即:实验中,R 表行数低于 25M 行时(向量宽度 1 字节,10 核处理器 LLC 大小为 25MB),选择率对向量引用连接算法性能产生的性能损失可以忽略不计;对于最新的 24 核 CPU,则最大可支持 60M 行表上不受选择率影响的连接性能.典型的 OLAP 负载中维表行数通常较小,AIR 算法的向量引用技术采用简单的定长向量连接技术具有良好的通用性.

结论 2. 设 R 表上的选择率为 s,则给定 R 表行数$|R|$时,连接性能可以通过图 4 柱状图中横轴取值,分别为$|R|$,$|R|\times s$ 和$|R|\times s$ 所对应的 AIR、NPO 和 PRO 算法曲线上的纵轴取值而确定,从而较为准确地对连接性能进行估算.

4.2 异构处理器platform-oblivious连接算法性能

本文第 3 节中讨论了向量引用技术作为 platform-oblivious 连接算法的可行性.我们分别在 Xeon Phi 5110P 和 NVIDIA K80 GPU 两种代表性的协处理器上通过 icc 编译器和 CUDA 编译器实现 AIR 算法,图 5 显示了 3 种处理器上单线程 AIR 连接算法(date⋈ lineorder)的性能,以每记录纳秒值为性能指标.

CPU 单线程性能最高,其较高的主频和功能强大的核心保证了其性能,但 CPU 上的并行物理线程数量只有 20.Phi 核心主频较低,性能低于通用 CPU 核心,单线程 AIR 算法性能低于 CPU.GPU 单线程性能远远低于 x86 平台的 CPU 和 Phi.CPU 和 Phi 通过 cache 提高连接性能,而 GPU 内存访问延迟较高,主要通过其强大的并行线程加速连接性能.

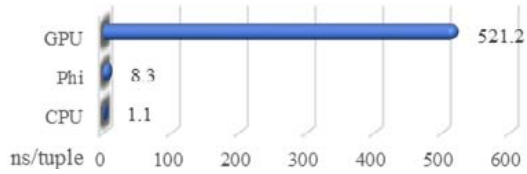

Fig.5 Performance of single-thread AIR

图 5 单线程 AIR 算法性能

我们分别在 3 个处理器平台使用最优的线程数量参数优化 AIR 算法性能,并以 SSB、TPC-H 和 TPC-DS 表连接为基础测试 AIR 算法的性能如图 6 所示.Phi 平台上 AIR 算法在 SSB 和 TPC-DS 负载的较小表连接操作中性能最高,CPU 平台上 AIR 算法在 SSB 和 TPC-H 测试集的中等大小的表连接操作中性能较高,而 GPU 平台上 AIR 算法在 SSB 和 TPC-H 的大表连接操作中性能最高.TPC-DS 测试集主要是小表连接,在 CPU、Phi 和 GPU 平台均有性能最高的情况.

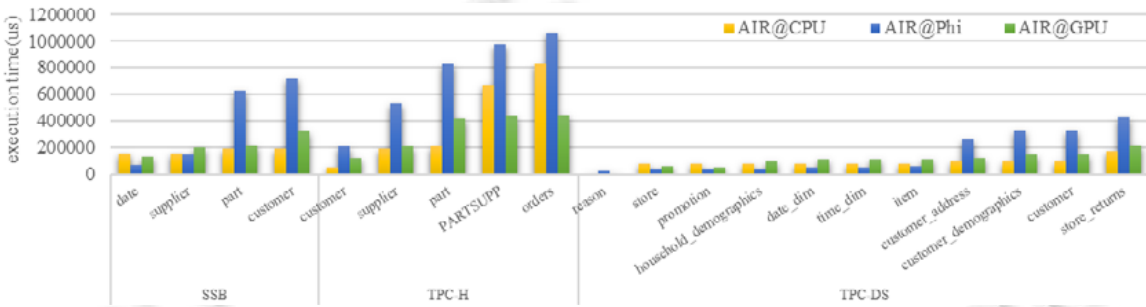

Fig.6 Join performance of AIR with SSB, TPC-H and TPC-DS on multicore, Phi and GPU

图 6 AIR 算法在多核 CPU、Phi、GPU 上基于 SSB,TPC-H 和 TPC-DS 连接操作性能

为进一步分析 AIR 算法在不同类型处理器上的性能规律,我们按第 4.1 节的测试方法在 CPU、Phi 和 GPU 这 3 种处理器平台上执行细粒度的连接测试,图 7 显示了 AIR 算法在向量大小相对于各级 cache 不同比例情况下的连接性能.Phi 平台 AIR 算法在向量大小低于 L2 cache 大小范围内优于 CPU 和 GPU 上的 AIR 算法性能(175%L2 cache 以下,Phi 的 L2 cache 为 512KB,2 倍于 CPU 的 L2 cache),其通过双向通道连接的共享 L2 cache 的性能低于 CPU L3 cache 性能,因此在较小的数据表(向量大小低于核心 L2 cache)时,通过更多的并行线程达到最优的性能;CPU 平台的 AIR 算法性能主要受 L3 cache 大小的影响,线程数少而 CPU 单核性能高,在向量大小介于 Phi 的 512KB L2 cache 和 CPU 的 L3 cache 容量之间时性能最优;GPU 单核性能最差.在向量大小低于 K80 1.5MB 的 L2 cache 时,GPU 上的 AIR 算法性能与 CPU 和 Phi 接近,在向量大小介于 GPU L2 cache 和 CPU L3 cache 时,CPU 上的 AIR 性能优于 GPU 和 Phi.在向量大小超过 CPU L3 cache 时,GPU 上的 AIR 算法性能最高,主要原因是当 cache 机制失效时,GPU 的 SIMT 机制保证了其在大表连接中仍然能够保持较好的内存访问性能.

结论 3. AIR 算法在不同架构处理器上连接操作性能与 cache 缓存及 SIMT 内存访问机制有一定依赖关系:当 AIR 算法的向量小于 LLC(第一代 Phi 处理器为 L2 大小)大小时,CPU 或 Phi 性能优于 GPU;当向量大小超过 LLC 时,GPU 性能更优.因此在查询优化器选择不同处理器平台时,可以根据连接负载特征评估连接操作在 CPU,Phi 和 GPU 平台上的性能,为算法平台选择提供依据.

结论 4. 在使用 AIR 算法执行连接操作时,查询负载处理器选择策略可以简化为计算向量大小与 LLC 大小的比值:当比值小于 1 时,连接算法适合在 CPU 平台执行;当比值大于 1 时,则连接操作适合在 GPU 平台执行.通过 AIR 连接性能曲线中的向量大小阈值估算连接性能,并选择具有较好性能的处理器作为负载处理平台.此外,文献[24]已对 3 种连接算法在 Phi 协处理器上的执行情况进行分析,这里不再赘述.

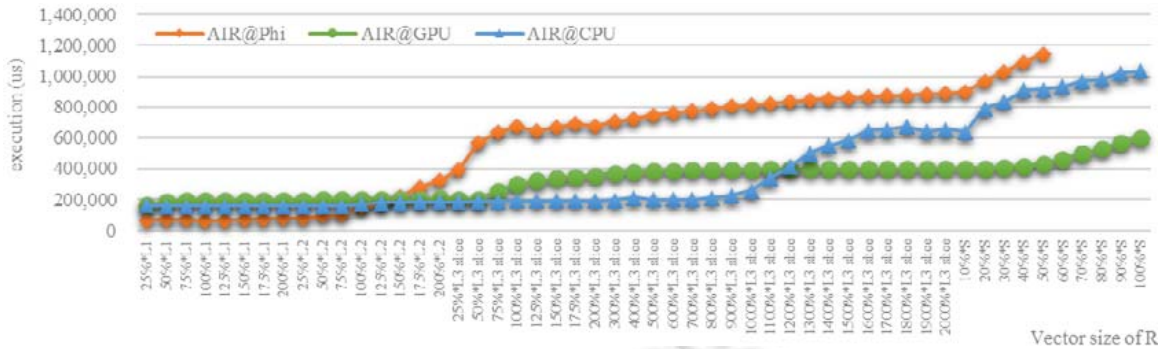

Fig.7 Performance analysis of AIR on CPU, Phi and GPU

图 7 AIR 算法在 CPU、Phi 和 GPU 平台的性能分析

4.3 代表性数据库及连接算法性能对比

我们进一步对比了当前 CPU 与 GPU 平台上代表性的数据库系统 Vector 5.0,MonetDB v11.25.15,MapD GPU 数据库以及代表性的 NPO 和 PRO 连接算法在 SSB 数据集(*SF*=100,其中,事实表记录行数为 6 亿条,4 个连接维表行数分别为 2 555、200 000、1 528 771 和 3 000 000)上的连接性能.

- 首先,从数据库连接性能来看,在实验平台上,MapD 使用 NVIDIA K80 GPU 为查询处理引擎,相对于使用多核 XeonCPU 为查询处理引擎的内存数据库 Vector 和 MonetDB 在连接性能上优势较为显著,最大连接性能提升程度分别达到 5 倍和 9 倍.基于列处理模型的 MonetDB 与 MapD 性能差距较大,图 8 中 MonetDB 在 customer 表连接中执行时间接近 3.5s,而 MapD 仅为 358ms;相对而言,基于向量处理模型的 Vector 数据库与基于向量处理模型的 GPU 数据库 MapD 性能主要相差 2 倍~5 倍.整体而言,基于 GPU 的 MapD 数据库依赖 GPU 的强大并行处理能力,相对于传统 x86 多核处理器平台的内存数据库系统性能有了较为显著的提升;
- 其次,从连接算法性能来看,由于 SSB 数据库集中维表相对 LLC 较小,因此 NPO 性能优于 PRO;同样,由于 SSB 连接中的连接向量小于 LLC,AIR 算法在 CPU 平台相对于 GPU 平台性能更高,同样高于 MapD 的 GPU 连接性能.

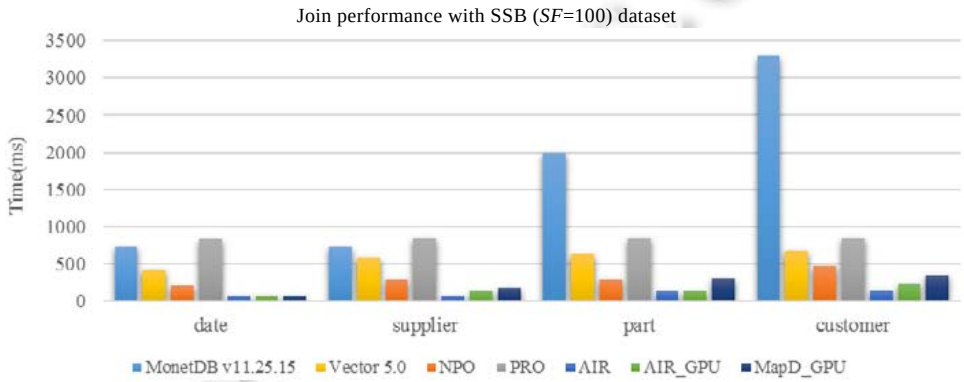

Fig.8 Comparision of join performance

图 8 连接性能对比

结论 5. 当前图数据处理面临着大数据、高性能分析处理能力的巨大压力,当前主流的技术是从 CPU 平台转向高性能 GPU 平台,通过 GPU 强大的并行处理能力加速图结构大数据分析处理性能.而图数据分析处理引擎

中的连接操作性能对数据库系统整体性能影响最为显著,本文所提出的 AIR 算法相对于已有的数据库系统的连接操作有较好的性能,能够进一步增强图结构数据库系统在面对复杂模式时的分析处理性能.

5 结论和未来的工作

图结构大数据分析对查询处理性能要求较高,在面对复杂数据模式时,连接操作进一步对图结构大数据分析处理性能提出挑战.MapD 数据库结合了新型众核处理器技术来加速社交网络图结构数据分析处理性能,其 GPU 数据库技术成为当前的代表性的高性能数据库实现技术,而且 MapD 同样支持面向多核 CPU、Phi 异构处理器平台的数据库系统,但其中重要的连接操作性能仍然需要进一步提升.本文首先基于当前学术界主流的内存连接技术分析哈希连接的极限性能以及优化技术的适应性,并探索 platform-oblivious 连接算法实现技术;然后,通过当前主流的多核 CPU、Phi 和 GPU 平台进行连接算法性能测试,揭示连接算法性能与硬件架构之间的依赖关系;通过 Benchmark 连接性能的测试分析给出了处理平台与负载之间的优化配置策略.

本文的研究表明:通过与数据仓库模式特点及 OLAP 负载特点的结合,连接操作通过索引、数据库约束、更新优化策略、数据压缩等综合技术能够实现面向应用领域的定制化连接优化技术,从而进一步简化连接操作的数据结构和算法实现,实现从 hardware-oblivious 连接算法向 platform-oblivious 算法的升级,从而更好地适应未来异构处理器平台.不同的处理器架构有其自身的性能优势区间和劣势区间,在内存数据库查询优化时,需要结合处理器特点和负载特点优化选择查询执行平台.

新型廉价、低能耗处理器技术的成熟推动数据库从传统的 CPU 平台向异构处理器平台的升级,在高性能计算 HPC 领域,GPU、Phi、FPGA 逐渐成为云计算的新兴平台,数据库技术面临着向新兴平台技术升级的压力,需要从数据库基础的关系操作算法实现层面扩展对异构处理器平台的支持.本文以多核 CPU、Phi 和 GPU 为基础,研究了新型处理器上的连接优化技术,并初步验证了连接算法设计和性能.本文的技术路线简化了连接算法设计,使其在向新型处理器平台迁移时技术门槛降低,我们在未来的工作中将进一步验证 platform-oblivious 连接算法设计在 FPGA 以及其他新型处理器平台上的实现技术与性能特征,完善面向异构处理器平台的连接优化技术.

References:

[1] https://www.mapd.com/

[2] Root C, Mostak T. MapD: A GPU-powered big data analytics and visualization platform. In: Proc. of the SIGGRAPH. Anaheim, 2016. 73:1−73:2. [doi: 10.1145/2897839.2927468]

[3] https://www.top500.org/lists/2016/11/

[4] https://www.top500.org/green500/lists/2016/11/

[5] Schuh S, Chen X, Dittrich J. An experimental comparison of thirteen relational equi-joins in main memory. In: Proc. of the SIGMOD. New York: ACM Press, 2016. 1961−1976. [doi: 10.1145/2882903.2882917]

[6] Richter S, Alvarez V, Dittrich J. A seven-dimensional analysis of Hashing methods and its implications on query processing. Proc. of the VLDB Endowment, 2015,9(3):96−107. [doi: 10.14778/2850583.2850585]

[7] Blanas S, Li Y, Patel JM. Design and evaluation of main memory Hash join algorithms for multi-core CPUs. In: Proc. of the SIGMOD. New York: ACM Press, 2011. 37−48. [doi: 10.1145/1989323.1989328]

[8] Balkesen C, Teubner J, Alonso G, OzsuT. Main-Memory Hash joins on multi-core CPUs: Tuning to the underlying hardware. In: Proc. of ICDE. Washington: IEEE Computer Society, 2013. 362−373. [doi: 10.1109/ICDE.2013.6544839]

[9] Albutiu MC, Kemper A, Neumann T. Massively parallel sort-merge joins in main memory multi-core data-base systems. Proc. of the VLDB Endowment, 2012,5(10):1064−1075. [doi: 10.14778/2336664.2336678]

[10] Lang H, Leis V, AlbutiuMC, Neumann T, Kemper A. Massively parallel NUMA-aware Hash joins. In: Proc. of the IMDM@VLDB. 2013. 3−14. [doi: 10.1007/978-3-319-13960-9_1]

[11] Pagh R, Wei Z, Yi K, Zhang Q. Cache-Oblivious Hashing. In: Proc. of the PODS. Indianapolis. 2010. 297−304. [doi: 10.1145/1807085.1807124]

[12] Mitzenmacher M. A new approach to analyzing robin hood Hashing. In: Proc. of the ANALCO. 2016. 10−24. [doi: 10.1137/1.9781611974324.2]

[13] Pagh R. Cuckoo Hashing. In: Proc. of the Encyclopedia of Algorithms. 2016. 478−481. [doi: 10.1007/3-540-44676-1_10]

[14] Barber R, Lohman GM, Pandis I, Raman V, Sidle R, Attaluri GK, Chainani N, Lightstone S, Sharpe D. Memory-Efficient Hash joins. Proc. of the VLDB Endowment, 2014,8(4):353−364. [doi: 10.14778/2735496.2735499]

[15] Zhang Y, Zhou X, Zhang Y, Zhang Y, Su M, Wang S. Virtual denormalization via array index reference for main memory OLAP. IEEE Trans. on Knowledge and Data Engineering, 2016,28(4):1061−1074. [doi: 10.1109/TKDE.2015.2499199]

[16] He BS, Yang K, Fang R, Lu M, Govindaraju NK, Luo Q, Sander PV. Relational joins on graphics processors. In: Proc. of the SIGMOD. New York: ACM Press, 2008. 511−524. [doi: 10.1145/1376616.1376670]

[17] Yuan Y, Lee R, Zhang X. The Yin and Yang of processing data warehousing queries on GPU devices. Proc. of the VLDB Edowment, 2013,6(10):817−828. [doi: 10.14778/2536206.2536210]

[18] Pirk H, Manegold S, Kersten M. Accelerating foreign-key joins using asymmetric memory channels. In: Bordawekar R, Lang CA, eds. Proc. of the Int'l Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS). 2011. 27−35.

[19] He J, Lu M, He B. Revisiting co-processing for Hash joins on the coupled CPU-GPU architecture. Proc. of the VLDB Endowment, 2013,6(10):889−900. [doi: 10.14778/2536206.2536216]

[20] Jha S, He BS, Lu M, Cheng XT, Huynh HP. Improving main memory Hash joins on intel Xeon Phi processors: An experimental approach. Proc. of the VLDB Endowment, 2015,8(6):642−653. [doi: 10.14778/2735703.2735704]

[21] Polychroniou O, Raghavan A, Ross KA. Rethinking SIMD vectorization for in-memory databases. In: Proc. of the SIGMOD. New York: ACM Press, 2015. 1493−1508. [doi: 10.1145/2723372.2747645]

[22] Halstead RJ, Absalyamov I, Najjar WA, Tsotras VJ. FPGA-Based multithreading for in-memory Hash joins. In: Proc. of CIDR. 2015.

[23] Zukowski M, Boncz PA. Vectorwise: Beyond column stores. IEEE Data Engineering Bulletin, 2012,35(1):21−27.

[24] Zhang Y, Zhang YS, Chen H, Wang S. OLAP Foreign Join Algorithm for MIC Coprocessor. Ruan Jian Xue Bao/Journal of Software, 2017,28(3):490−501 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5156.htm [doi: 10.13328/j.cnki.jos.005156]

[25] Lee JG, Attaluri GK, Barber R. Joins on encoded and partitioned data. Proc. of the VLDB Endowment, 2014,7(13):1355−1366.

附中文参考文献:

[24] 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. http://www.jos.org.cn/1000-9825/5156.htm [doi: 10.13328/j. cnki.jos.005156]

张延松(1973－),男,黑龙江牡丹江人,博士,副教授,主要研究领域为内存数据库,OLAP,数据仓库.

王珊(1944－),女,教授,博士生导师,CCF 会士,主要研究领域为数据库,数据仓库.

张宇(1976－),女,博士,副教授,主要研究领域为 GPU 数据库,OLAP,数据仓库.