软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(3):490−501 [doi: 10.13328/j.cnki.jos.005156]

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

面向 MIC 协处理器的 OLAP 外键连接算法*

张 宇[1], 张延松[2,3,4], 陈 红[2,3], 王 珊[2,3]

[1](中国气象局 国家卫星气象中心,北京 100081)

[2](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)

[3](中国人民大学 信息学院,北京 100872)

[4](中国人民大学 中国调查与数据中心,北京 100872)

通讯作者: 张延松, E-mail: zhangys_ruc@hotmail.com

摘 要: 众核架构协处理器 Xeon Phi 成为新兴的主流高性能计算平台.对于数据库应用而言,内存分析处理是一种计算密集型负载,其性能主要取决于大事实表与维表之间的内存外键连接性能.关注于一种相对于缓存相关的分区哈希连接算法和缓存不相关的无分区哈希连接算法的缓存友好型外键连接算法,以适应 Xeon Phi 协处理器较小的 LLC 和高并发线程的特点.通过挖掘 OLAP 模式中的代理键特征,基于键值匹配的哈希探测操作,可以进一步简化为事实表与维表之间基于主-外键参照完整性约束的代理键参照访问,因此,复杂的哈希表和 CPU 代价较高的哈希探测操作可以简化为通过映射外键值为代理键向量内存偏移地址的方法对代理向量直接访问.基于代理向量参照访问的外键连接算法,能够简单并高效地应用于 Xeon Phi 协处理器平台,通过更多的核心和高并发线程来掩盖内存访问延迟.实验中,对传统的哈希连接算法(无分区哈希连接算法和基数分区哈希连接算法)和基于代理向量参照技术的外键连接算法在 Xeon E5-2650 v3 10 核处理器平台和 Xeon Phi 5110P 60 核协处理器平台进行性能测试和比较,实验结果给出了主流的内存外键连接算法在不同数据集和不同平台上全面的性能特征.

关键词: 内存 OLAP;外键连接;代理键;代理键参照

中图法分类号: TP311

中文引用格式: 张宇,张延松,陈红,王珊.面向 MIC 协处理器的 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

英文引用格式: 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). http://www.jos.org.cn/1000-9825/5156.htm

OLAP Foreign Join Algorithm for MIC Coprocessor

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

[1](National Satellite Meteorological Center, China Meteorological Administration, Beijing 100081, China)

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

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

[4](National Survey Research Center, Renmin University of China, Beijing 100872, China)

Abstract: The emerging many integrated core architecture (MIC) Xeon Phi coprocessor becomes the mainstream platform for high performance computing. For database applications, in-memory analytics requires computation intensive workload in which the in-memory

* 基金项目: 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); 华为创新研究计划(HIRP 20140507, HIRP 20140510)

Foundation item: National High-Tech R&D Program of China (863) (2015AA015307); Basic Research Funds in Renmin University of China from the Central Government (16XNLQ02); Huawei Innovation Research Program (HIRP 20140507, HIRP 20140510)

收稿时间: 2016-07-18; 修改时间: 2016-09-14; 采用时间: 2016-11-01; jos 在线出版时间: 2016-11-29

CNKI 网络优先出版: 2016-11-29 13:34:57, http://www.cnki.net/kcms/detail/11.2560.TP.20161129.1334.003.html

foreign key joins between big fact table and dimension tables dominate the OLAP performance. This paper focuses on a cache-friendly foreign key join with respect to cache-conscious radix partitioning oriented hash join and cache-oblivious no-partitioning hash join to adapt to the small LLC size and massive simultaneous multi-threading mechanism of Xeon Phi coprocessor. By exploiting the characteristic of surrogate key in OLAP schema, the key matching oriented hash probing can be further simplified as surrogate key referencing between fact table and dimension tables with PK-FK reference constraint, so that the complex hash table and CPU cycle consuming hash probing can be simplified as directly referencing surrogate vector by mapping foreign key to offset address of surrogate vector. The surrogate vector referencing oriented foreign key join is simple and efficient to be implemented for Xeon Phi coprocessor for more cores, and also offers massive simultaneous multi-threading mechanism to overlap memory access latency. In experiments, the surrogate vector referencing foreign key join algorithm and traditional hash join algorithms (NPO and PRO) are compared on both Xeon E5-2650 v3 10-core CPU platform and Xeon Phi 5110P 60-core platform, the experimental results provide a comprehensive perspective for how the mainstream in-memory foreign key join algorithms perform with different datasets on different platforms.

Key words: in-memory OLAP; foreign key join; surrogate key; surrogate vector referencing

随着半导体制程工艺不断提高,处理器的晶体管集成度越来越高,例如代号 Knights Landing 的新一代 Intel Xeon Phi 处理器采用 14nm 工艺制造,拥有 72 亿个晶体管[1].晶体管集成度的大幅度提升,使处理器能够集成更多的缓存单元或处理单元,大幅度提高了处理器的数据处理能力.从晶体管的集成特点来看,通用处理器(CPU)采用了 L1-L2-L3 多级缓存结构,大量晶体管用于构建控制电路与 LLC(last level cache),仅有 5%左右用于构建计算单元 ALU.相应地,通用处理器主要通过高性能缓存加速数据处理性能;协处理器(coprocessor)中的 Cache 较小,大约 40%的晶体管用于构建计算单元,通过更多的并发线程来掩盖内存访问延迟.

从硬件特性来看,通用多核 CPU 采用以数据为中心的设计思想,而众核协处理器则采用以计算为中心的设计理念.硬件结构的不同,使多核处理器和众核协处理器在不同特征的负载上表现出不同的性能,因此,查询算法设计也需要提供不同的查询优化技术路线以适应多核处理器与众核协处理器的处理特征.当前,内存数据库以基于多级缓存的通用多核处理器架构为基础,在查询算法设计上采用以多级 Cache 为中心的 Cache-Conscious 算法,但这种适应通用多核处理器硬件特征的算法,设计思想与新兴的众核协处理器硬件架构存在较大的差异,需要针对众核协处理器的硬件特性重新设计适合的查询处理算法,以更好地发挥众核协处理器的高性能.当前,基于 Xeon Phi 众核协处理器的研究主要集中在分析主流的内存哈希连接算法(包括 Cache-Conscious 与 Cache-Oblivious 哈希连接算法)在 Xeon Phi 硬件架构下的性能特征[2,3],通过多线程与 SIMD 机制优化哈希连接性能.但与内存哈希连接研究不同,对于 Xeon Phi 协处理器不仅要考虑性能问题,还需要考虑其内存利用率和代码执行效率问题;同时,针对 OLAP 模式的特点还需要进一步优化外键连接算法的设计,提高 Phi 协处理器的查询处理性能和效率.

本文将探讨面向 Xeon Phi 众核协处理器的内存 OLAP 外键连接算法优化技术,通过数据仓库代理键技术设计代理键索引机制,实现一种基于代理向量参照访问的内存 OLAP 外键连接技术,以结构紧凑的向量代替复杂结构的哈希表,提高数据存储与访问效率,通过简化哈希连接算法为向量地址访问的设计使其更加适合于处理核心众多、高并发线程、简单计算核心的 Phi 协处理器计算特点.

本文第 1 节对内存数据库连接相关技术进行对比分析.第 2 节描述基于代理键索引机制的外键连接实现技术.第 3 节给出相关内存连接算法在多核 CPU 和众核 Phi 协处理器上的实验性能.最后对本文加以总结.

1 相关工作

当前的处理器技术可以分为 3 类:多核主处理器、众核协处理器和融核处理器.多核主处理器采用同构的多个处理核心,作为主处理器直接访问内存,处理核心数量相对较少,核心处理能力较强,主频较高,主要以 Intel Xeon 系列多核处理器以及未来的 Knights Landing 众核主处理器为代表.众核协处理器以 NVIDIA GPGPU, Xeon Phi,FPGA[4]协处理器为代表,主要特点是核心数量众多,支持大量并发线程,核心向量计算能力较强但复杂数据处理能力较弱,主频较低,拥有本地高带宽内存,与主存通过带宽较低的 PCI-E 通道进行通信.融核处理器是一种新兴的处理器技术,它在多核处理器内部集成了众核计算单元,如 GPU 单元[5]或 FPGA 单元[6],由通用处

理核心完成复杂的数据管理与处理,对计算密集型负载由 GPU 或 FPGA 单元来加速其性能.

从处理器的硬件发展趋势来看,计算将从通用计算平台走向异构众核计算平台,查询处理中的计算密集型负载主要由众核并行计算硬件加速,需要算法更好地适应高并行计算特征.

在内存数据库应用中,内存 OLAP 是基于多维数据模型的大数据实时分析处理技术,其中,庞大的事实表与众多维表之间的连接操作性能是 OLAP 查询处理性能的决定性因素.

当前的内存连接技术主要包括哈希连接和排序归并连接技术,其主要特点是能够较好地利用先进处理器强大的多核并行和 SIMD 向量处理能力.在内存计算平台上有充足的内存资源,研究主要集中在如何更好地利用多核多线程资源、Cache 资源和处理器的 SIMD 向量处理能力提高连接性能.近年来,主要的内存连接技术集中在内存哈希连接技术上,主要技术路线分为两类:一类是基于共享哈希表访问的 Cache-Oblivious 哈希连接算法,主要特点是使用简单的共享哈希表访问,不需要按照 Cache 大小优化数据集,而是依靠现代处理器多线程并发访问及自动预取等技术提高哈希连接性能;第 2 类是基于 radix 分区的 Cache-Conscious 哈希连接算法,通过对连接表按相同的基数进行分区,将大的连接表划分为适合 Cache 大小的子表,从而保证哈希表的构建与哈希表上的哈希探测具有良好的 Cache 数据局部性.这种分而治之的划分方法需要根据处理器的 TLB 及 Cache 大小等硬件参数进行优化设置,其哈希连接性能与具体的硬件参数紧密相关.

近年来,连接性能的优化工作不断更新.文献[7]验证了在现代多核处理器平台上,Cache-Oblivious 特点的简单无分区哈希连接算法在现代多核处理器硬件优化技术的支持下能够获得比复杂的 Cache-Conscious 特点的分区哈希连接算法更优的性能.研究的结论表明:简化的数据库连接算法设计能够随着处理器硬件技术的进步而自动获得较高的性能,从而在先进处理器硬件技术的支持下简化数据库哈希连接算法设计.但进一步的研究[8]表明,面向处理器 TLB,Cache 及 SIMD 等硬件的优化技术能够更好地挖掘多核处理器的硬件性能优势,使基于 Cache-Conscious 设计思想的基数分区哈希连接算法的性能优于基于 Cache-Oblivious 设计思想的无分区哈希连接算法.性能对比实验结果表明:无分区哈希连接算法的主要开销在于共享哈希表上的哈希探测代价,即,对较大哈希表的内存访问延迟代价;而 radix 分区的哈希连接算法的主要代价在于分区操作,当较大的表划分为适合 Cache 容量的小表时,其上的哈希表构建及哈希探测性能显著提高,执行代价大幅降低.实验结果表明:多核处理器上内存访问延迟仍然是最主要的性能瓶颈因素,多核处理器较大的 LLC(每核心 2.5MB)有效地提高了 Cache 数据缓存效率.文献[9]研究了基于现代处理器 SIMD 性能的排序归并算法优化技术,SIMD 能够较好地提高排序性能,从综合性能来看,分区哈希连接仍然具有最好的性能.

Xeon Phi 协处理器在硬件结构上与通用 CPU 既类似又有区别:Xeon Phi 协处理器有类似的 L1-L2 两级核心私有 Cache 结构,但缺少较大的共享 L3 Cache;处理核心的结构来源于 P54C 通用处理核心,但每核心支持 4 线程,高于通用多核处理器每核心 2 线程,具有更高的并发访问性能.从硬件特点来看,Xeon Phi 协处理器既适用于面向 TLB 和 Cache 优化的 Cache-Conscious 哈希连接算法,又适用于面向并发内存访问的 Cache-Oblivious 哈希连接算法.从 Xeon Phi 协处理器本地内存利用率的角度来看,性能较好的内存 radix 分区哈希连接及排序归并连接算法需要额外的分区及排序存储空间开销来保证连接操作的高性能,内存资源充足的内存计算平台可以提供有力支持,但对于 Xeon Phi 协处理器则会导致本地内存的有效利用率降低.从连接算法的空间开销来看,无分区的 Cache-Oblivious 哈希连接算法具有更高的内存利用率.

在 GPU 协处理器的研究中,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[10],哈希表通常表示为多级数组,简化其在协处理器上的数据结构.由于分区操作较高的内存代价,GPU 上的哈希连接通常采用简单的无分区哈希连接算法[11,12].

由于 Phi 协处理器采用与 CPU 兼容的 x86 指令集和类似的体系结构,因此,CPU 上的算法相对于需要用 CUDA 或 OpenCL 改写的 GPU 算法而言更加便于移植到 Phi 协处理器平台.文献[2]将文献[8]的开源哈希连接算法改写为 Phi 协处理器版本,增加 512 位 SIMD 指令优化哈希函数计算和哈希桶加载,并在 Phi 协处理器上测试了无分区哈希连接算法和 radix 分区哈希连接算法的性能.实验结果显示:无分区哈希连接算法在 Phi 协处理器更高的并发内存访问性能的支持下获得了较优的性能,Phi 协处理器大量的内核及高并发线程较好地提升了

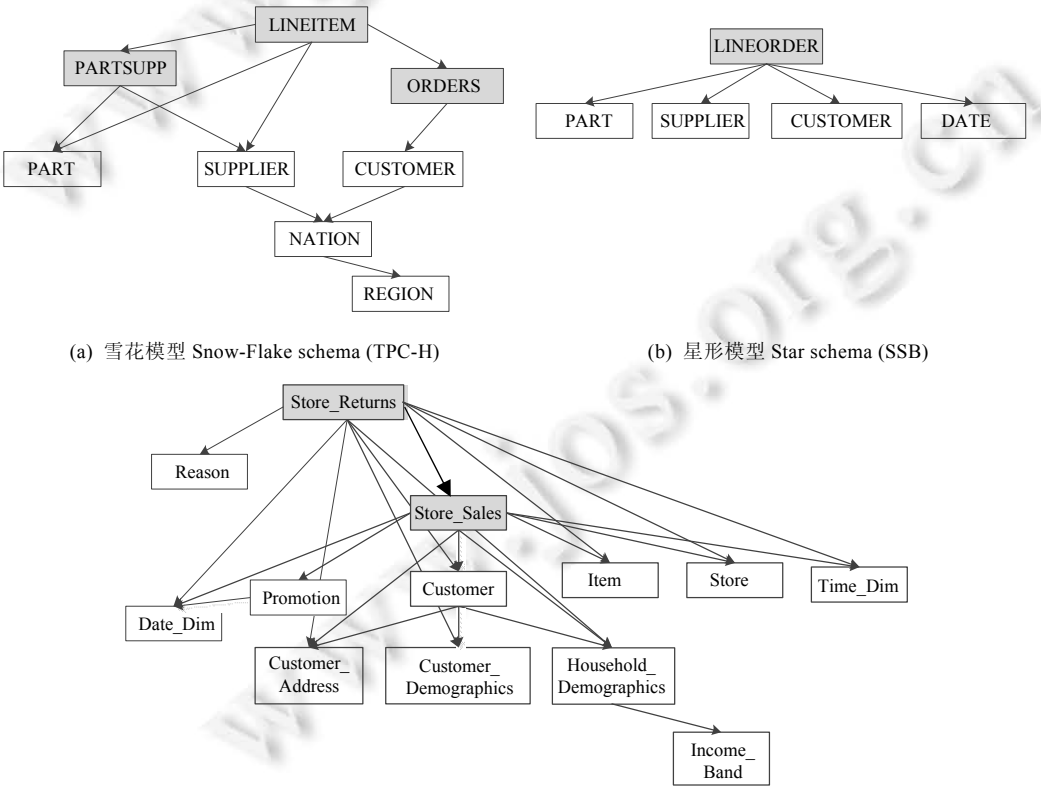
Cache-Oblivious 无分区哈希连接的性能.最新的研究[3]进一步扩展了 Phi 协处理器 512 位 SIMD 指令的应用范围,通过 SIMD 实现所有的基础操作符,从而更加充分地利用了 Phi 协处理器 512 位 SIMD 指令的高性能,提升了 radix 分区哈希连接性能.

以 OLAP 模式视角来看,哈希连接并不涉及 OLAP 模式的特点,如多维数据模型特点及维表与事实表之间的参照完整性约束特点.文献[13]进一步挖掘了数据仓库中维表代理键[14]及维表与事实表之间的参照完整性约束关系的特点,在基于数组存储的 A-Store 模型上提出了数组地址参照访问技术(array index referencing,简称 AIR),以事实表外键数组地址映射技术实现事实表与维表之间的外键连接操作.AIR 算法相对于哈希连接算法使用简单的数组代替复杂的哈希表数据结构,使用数组地址访问代替哈希探测操作,不仅具有较好的性能,而且更适合于众核协处理器简单代码大规模并行处理的特点.本文将进一步探索 AIR 算法在 Xeon Phi 协处理器上的性能.

综上所述,内存 OLAP 外键连接操作的代表性实现技术包括哈希连接与 AIR 访问等技术,我们将在本文中进一步研究其在 Xeon Phi 协处理器上的性能特征,给出内存 OLAP 外键连接操作技术选择的参考.

2 基于代理键索引机制的外键连接实现技术

如图 1 所示,典型的数据仓库模式分为雪花模型、星形模型和暴风雪模型,图中深颜色框图代表事实表,浅颜色框图代表维表,箭头代表表间的参照完整性约束关系.数据仓库面向主题和多维数据的特点表现为所有的表构成一棵有向树,即事实表与维表、事实表之间都存在主-外键参照完整性约束条件.



(a) 雪花模型 Snow-Flake schema (TPC-H)

(b) 星形模型 Star schema (SSB)

(c) 暴风雪模型 Snow-Storm schema (TPC-DS)

Fig.1 Data warehouse schema

图 1 数据仓库模式

一个 OLAP 查询相当于在一棵子树上的查询处理任务,即,通过关系子集间的外键连接操作执行的分组聚集计算任务.因此,OLAP 查询的关键技术是表间基于主-外键参照完整性约束关系的外键连接操作.

2.1 数据仓库代理键

代理键是数据仓库中一种普遍使用的主键机制,它采用连续的整数型数值作为主键,如 0,1,2,….代理键消除了数据仓库维表主键的语义信息,简化了维表主键存储和键值匹配操作.在内存列存储 OLAP 模型中,代理键代表了维表的地址语义,可以映射为 OLAP 多维数据模型的维坐标,即,代理键可以作为事实数据多维坐标在指定维上的坐标分量.因此,代理键不仅仅是数据仓库中关系存储的一种主键机制,也是数据仓库多维数据模型的一种多维空间地址映射机制.

如表 1 所示,数据库代表性的 Benchmark,如 SSB,TPC-H,TPC-DS 的维表全部采用代理键,其中,SSB 的 date 表和 TPC-DS 的 date_dim 采用日期格式或预定义的连续值.维表代理键的特点使其能够通过键值映射为维表记录的内存偏移地址,实现基于代理键值的地址访问.因此,在使用代理键的数据库中,代理外键起到连接索引的作用,即通过代理键值能够直接映射其参照表中记录的内存偏移地址.

Table 1 Surrogate key in database benchmarks

表 1 数据库 Benchmark 中的代理键

	表	代理键	代理键地址映射函数
SSB	customer	1,2,3,…	$f(key)=key-1$
	supplier	1,2,3,…	$f(key)=key-1$
	part	1,2,3,…	$f(key)=key-1$
	date	19920101,19920102,…	$f(key)=date(key)-date(key_0)$
TPC-H	customer	1,2,3,…	$f(key)=key-1$
	supplier	1,2,3,…	$f(key)=key-1$
	part	1,2,3,…	$f(key)=key-1$
	nation	0,1,2,…	$f(key)=key$
	region	0,1,2,…	$f(key)=key$
TPC-DS	call_center	1,2,3,…	$f(key)=key-1$
	catalog_page	1,2,3,…	$f(key)=key-1$
	customer	1,2,3,…	$f(key)=key-1$
	customer_address	1,2,3,…	$f(key)=key-1$
	customer_demographics	1,2,3,…	$f(key)=key-1$
	date_dim	2415022,2415023,…	$f(key)=key-key_0$
	household_demographics	1,2,3,…	$f(key)=key-1$
	income_band	1,2,3,…	$f(key)=key-1$
	item	1,2,3,…	$f(key)=key-1$
	promotion	1,2,3,…	$f(key)=key-1$
	reason	1,2,3,…	$f(key)=key-1$
	ship_mode	1,2,3,…	$f(key)=key-1$
	store	1,2,3,…	$f(key)=key-1$
	time_dim	0,1,2,…	$f(key)=key$
	warehouse	1,2,3,…	$f(key)=key-1$
	web_page	1,2,3,…	$f(key)=key-1$
	web_site	1,2,3,…	$f(key)=key-1$

2.2 代理键索引机制

代理键索引机制是将代理键作为索引,并通过代理键地址映射实现外键连接操作,其实现机制需要满足 3 个方面的条件:被参照表使用代理键;参照表使用代理外键;被参照表保持代理键值的连续性.

在数据仓库的模式设计中,维表通常默认使用代理键作为主键,可以直接应用代理键索引机制实现外键对主键列的地址映射访问.在 TPC-H 和 TPC-DS 中的事实表之间也存在着主-外键参照完整性约束,但事实表并没有使用代理键作为主键,而较大事实表之间采用哈希连接操作时产生较高的连接代价,因此,我们为非代理键的参照完整性约束的表增加代理键索引,将表间连接操作转化为基于代理键的地址映射操作.根据事实表间主、外键的实际情况,我们设计了代理键索引转化机制和增加代理键索引机制,下面结合 TPC-H 中的具体实例进行

说明.

当被参照表主键为单一键时,我们通过更新主键为代理键实现代理键索引机制.如图 2 所示:ORDERS 表的主键 ORDERKEY 为不连续整数,其 LINEITEM 表上的外键列 ORDERKEY 为 LINEITEM 表复合主键的第一关键字,因此,ORDERS 表与 LINEITEM 表均按 ORDERKEY 列聚簇存储.我们通过 ORDERS 表 ORDERKEY 列与 LINEITEM 表 ORDERKEY 列的归并扫描实现代理键索引更新,即:扫描 ORDERS 表 ORDERKEY 列的每一个键值,同步扫描 LINEITEM 表 ORDERKEY 列相同的键值,并用 ORDERS 表 ORDERKEY 列的偏移地址同步更新 LINEITEM 表 ORDERKEY 列中具有相同 ORDERKEY 值的记录,然后将 ORDERS 表 ORDERKEY 列中原始的主键值更新为 ORDERKEY 列的偏移地址,使 ORDERKEY 列成为代理键.

当被参照表为复合主键时,如图 2 中 PARTSUPP 表的主键为复合键(PARTKEY,SUPPKEY),我们为被参照表 PARTSUPP 和参照表 LINEITEM 分别增加一个代理键列 SK_PS 和代理外键列 FK_PS,代理键列 SK_PS 更新为通过偏移地址计算出的代理键序列,然后,通过 PARTSUPP 表与 LINEITEM 表间的连接操作用 SK_PS 值更新 FK_PS 列具有主-外键参照关系记录的外键值.增加的代理键列和代理外键列作为 PARTSUPP 表和 LINEITEM 表之间新的主-外键使用,并可以通过代理外键的地址映射实现连接操作.

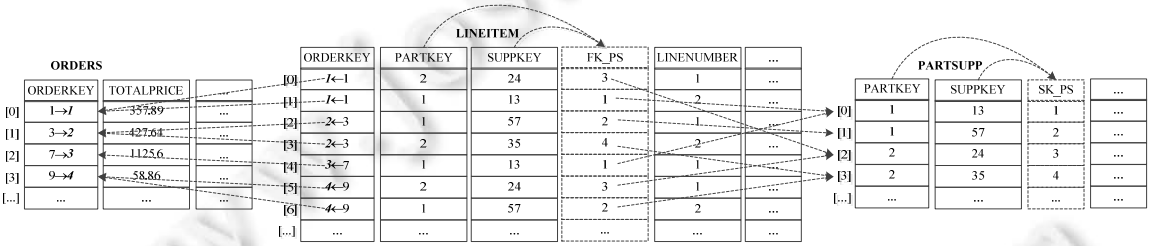


Fig.2 Example for surrogate key index mechanism

图 2 代理键索引机制示例

TPC-DS 中,事实表 store_sales 的复合主键为(ss_item_sk,ss_ticket_number),store_return 表的复合主键为(sr_item_sk,sr_ticket_number),并且该复合主键为 store_sales 表主键(ss_item_sk,ss_ticket_number)的外键.同样地,我们为 store_sales 表和 store_return 表增加一个代理键列和代理外键列,代理键列赋值为连续的整数序列,并根据 store_sales 表主键和 store_return 表外键的聚簇索引关系通过归并扫描更新 store_return 表的代理外键列,实现 store_sales 表和 store_return 表之间的代理键索引地址映射访问.

2.3 代理键索引更新机制

代理键地址映射机制的基础是代理键取值的连续性.代理键所在表上的更新操作可能破坏代理键取值的连续性,因此需要一定的机制或增加一些约束条件来保证代理键更新时的序列连续性.

首先,数据仓库存储历史数据的特点决定了其更新操作并不是通用意义的更新,而是一种 insert-only 特点的更新.事实表存储的是历史数据,新记录追加到事实表中;维表记录由于主-外键参照完整性约束关系而不能直接删除记录;代理键消除了语义信息,从而保证维表记录更新不需要更新代理键列.

其次,代理键索引机制需要保持代理键序列与记录物理存储位置的一致性,从而实现通过外键值映射到相应的参照记录地址.数据库通常将 update 操作分解为 insert 更新后的记录并 delete 原始记录的操作,从而导致代理键值与记录物理位置的不一致,破坏代理键索引访问.文献[13]采用数组存储模型和 in-place update 原位更新强约束机制来保证代理键值与记录位置的一致性,需要定制数组存储引擎.我们采用图 3 所示的代理向量机制实现了逻辑代理键索引机制.逻辑代理键只需要保证代理键值的唯一性和分配的连续性,不需要保持聚簇存储约束.在查询处理时创建一个代理向量,表上的查询结果按逻辑代理键值映射到代理向量相应的单元中.如图中输出 d_dayofweek 列按 surrogate key 列的值投影到代理向量 Dim_vec 上,作为维表的代理向量被外键列访问.

surregate key	d_datekey	d_date	d_dayofweek	d_month	d_year
2	19920102	2-Jan-92	Friday	January	1992
9	19920109	9-Jan-92	Friday	January	1992
5	19920105	5-Jan-92	Monday	January	1992
12	19920112	12-Jan-92	Monday	January	1992
3	19920103	3-Jan-92	Saturday	January	1992
10	19920110	10-Jan-92	Saturday	January	1992
4	19920104	4-Jan-92	Sunday	January	1992
11	19920111	11-Jan-92	Sunday	January	1992
1	19920101	1-Jan-92	Thursday	January	1992
8	19920108	8-Jan-92	Thursday	January	1992
15	19920115	15-Jan-92	Thursday	January	1992
6	19920106	6-Jan-92	Tuesday	January	1992
13	19920113	13-Jan-92	Tuesday	January	1992
7	19920107	7-Jan-92	Wednesday	January	1992
14	19920114	14-Jan-92	Wednesday	January	1992

Dim_vec	
[1]	Thursday
[2]	Friday
[3]	Saturday
[4]	Sunday
[5]	Monday
[6]	Thursday
[7]	Wednesday
[8]	Thursday
[9]	Friday
[10]	Saturday
[11]	Sunday
[12]	Monday
[13]	Thursday
[14]	Wednesday
[15]	Thursday

Fig.3 Surrogate vector

图 3 代理向量

在 OLAP 查询中,维表输出的是低势集的分组属性,在选择操作和字典表压缩操作处理后通常为较小的维表代理向量,用于外键连接操作.通过上述的代理向量转换机制,文献[13]中严格依赖数组偏移地址作为维表代理键的数组存储强约束机制转换为放松的逻辑代理键机制,能够应用于更加通用的数据库实现框架中.图 4 显示了基于代理向量参照访问的外键连接算法原理.R 表使用逻辑代理键作为主键,S 表使用逻辑代理外键,在 S 表执行选择、投影操作时创建代理向量,并对选择、投影出的分组属性进行压缩,以进一步缩减代理向量大小.代理向量记录逻辑代理键的物理位置和相应的压缩分组编码值,代理向量与 R 表执行实际的外键连接操作.

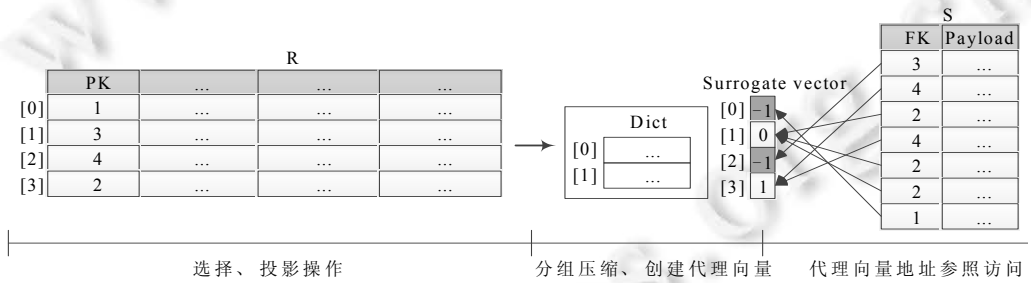


Fig.4 Surrogate vector referencing oriented foreign key join algorithm

图 4 基于代理向量参照访问的外键连接算法

算法 1 描述了 AIR 外键连接算法.当查询中 R 表无输出字段时,代理向量简化为位图.当创建代理向量后,S 表通过外键值映射到代理向量单元地址,实现 $O(1)$代价的连接操作.相对于哈希连接操作,AIR 算法使用简单的向量结构,消除连接键值哈希函数计算、哈希映射、键值匹配及可能的溢出桶检索代价,其性能主要取决于对向量的随机内存访问代价,在现代多级 Cache 结构中,向量访问的性能取决于向量相对于最后一级 Cache(LLC)的大小,当向量小于 LLC 时,基于 Cache 访问的代理向量访问具有较高的性能.

算法 1. AIR 外键连接算法.

INPUT:主键表 *R*,外键表 *S*,查询命令 *q*;

OUTPUT:连接记录集 *Res*.

BEGIN

SV←∅; //初始化代理向量 *SV* 为空;

/*根据查询命令 *q* 中的谓词操作创建代理向量 *SV* 和字典表 *Dict**/

FOR *R* 表中的每个记录 DO

FillingSV(*PK*,*q*,*SV*,*Dict*); //根据谓词表达式投影输出属性并通过 *Dict* 表压缩,压缩编码按逻辑主键 *PK* 值映射地址填充代理向量 *SV*

END FOR;

/**S* 表通过外键地址映射访问代理向量 *SV*,完成外键连接操作*/

FOR *S* 表中的每个外键值 *FK* DO

Res←*AddressReferencing*(*SV*); //映射代理向量 *SV* 中的地址并访问相应单元值

END FOR;

Return *Res*;

END.

2.4 基于代理键索引机制的Phi协处理器外键连接

文献[13]的实验结果显示:事实表外键列上的 AIR 连接操作占 OLAP 查询总代价的 85%以上,而事实表外键列的大小在总数据量中占比通常低于 20%,维表则低于 5%.因此,Phi 协处理器可以用作 OLAP 中事实表外键连接操作加速器,将较大的事实表外键列驻留存储于 Phi 协处理器内存,将 OLAP 查询产生的较小的维表代理向量传输到 Phi 协处理器执行外键连接操作.一般地,在进行主外键连接操作时,Phi 协处理器端需要执行哈希连接等操作,哈希表结构在存储和访问上比较复杂,而且需要付出额外的哈希表结构存储空间代价.为加速 Phi 协处理器端的连接性能,我们可以在 Phi 端应用代理键与代理外键机制,通过主键列与外键列的数据更新提供与 CPU 端不同的外键连接操作执行机制.在事实表外键列加载到 Phi 协处理器内存的过程中,第 2.2 节所述的代理键索引机制可以作为 ETL 过程完成原始事实表外键向代理外键的转换,从而在 Phi 协处理器上执行代码效率更高的 AIR 算法.

相对于哈希连接算法,基于代理键索引机制的 AIR 算法使用简单的向量数据结构,不需要复杂的哈希表,不需要存储代价高的分区操作,记录不需要存储键值,不需要溢出桶结构,具有更好的存储效率.数组地址参照访问相对于哈希探测更为直接,不需要哈希函数映射、哈希桶中记录键值的线性匹配等计算代价较高的操作,在存储和 CPU 计算效率上较为优化.AIR 算法将复杂的数据访问与计算操作简化为根据映射地址的向量访问,能够更好地利用 Phi 协处理器众核高并行线程内存访问的性能优势,提高 OLAP 外键连接性能.

文献[13]的实验证明,AIR 算法使用的数组地址参照连接技术在 OLAP 应用中具有较好的 Cache 数据局部性和 CPU 执行效率.本文进一步研究其在具有更高并行计算资源但 Cache 相对较小、核心性能相对较低的 Phi 协处理器上的性能特征,从而为 AIR 算法在新型 Phi 协处理器上的算法选择提供性能参考.

3 实验结果与分析

我们在一台 DELL PowerEdge R730 服务器上进行外键连接算法测试,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,512GB 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.服务器配置了一块 Intel Xeon Phi 5110P@1.053GHz 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 4.7.0.两块 Xeon E5-2650 v3 CPU 的价格与一块 Phi 5110P 的价格基本相当.

我们在文献[8]中开源的 NPO,PRO 算法的基础上增加了 AIR 算法.在数据生成器中屏蔽了对 R 表的 shuffler 过程,保证 R 表使用代理键作为主键,S 表需要进行 shuffler 过程,使之符合主-外键约束的一般场景.AIR 算法实现中模拟了 R 表上的谓词操作,在 NPO 算法的 BUILD 过程对 R 表进行过滤,生成 R 表代理向量,然后,S 表根据外键取值映射到 R 表代理向量,完成连接操作.AIR 算法生成的 R 表代理向量可以使用不同的数据类型,如 int_8,int_16 或 int_32,根据对 SSB,TPC-H 等查询中维表分组投影操作的分析,我们主要使用 int_8 作为代理向量的数据类型,模拟维过滤位图及分组向量存储.在实验中,R 表和 S 表采用 int_32 数据类型 key 和 payload,key 的

值域(2^{32}–1)能够满足数据仓库维表键值的取值范围.考虑到内存数据库普遍采用数据压缩技术,我们在实验中并未使用 int_64 数据类型.在 CPU 平台和 Phi 平台通过不同线程并行度的测试,确定在 CPU 与 Phi 平台的线程数量设置为与物理线程数相同时获得最佳的连接性能,即:CPU 平台设置并行线程数为 40,Phi 平台并行线程数为 240.

3.1 CPU端连接算法性能对比分析

在实验中,我们重点考查连接算法在数据集相对于各级 Cache 为不同比例时的性能.

图 5 显示了基于代理向量参照访问的 AIR 算法、无分区哈希连接算法 NPO 和 radix 分区哈希连接算法 PRO 在不同连接数据集下的连接性能.其中,AIR 表示代理向量宽度为 8 位,AIR_int_16 和 AIR_int_32 分别表示代理向量宽度为 16 位和 32 位.测试中:较大的 S 表记录长度为 600 000 000,相当于 *SF*=100 时,SSB 和 TPC-H 中事实表长度;较小的 R 表取不同的行数,用于表示相对于各级 Cache 不同大小的连接数据集.纵坐标连接性能的指标为平均每记录连接消耗的 CPU 指令周期数量(cycle/tuple),横坐标表示以 AIR 算法为基准的代理向量大小与 Cache 的比例.如,50%×L2 表示代理向量大小为 50%×256KB(L2 Cache size),长度为 131 072 行.由于 AIR 算法使用代理向量下标作为隐式的键值,因此相同行数的情况下,NPO 和 PRO 的哈希表大小高于 AIR 算法使用的代理向量大小.

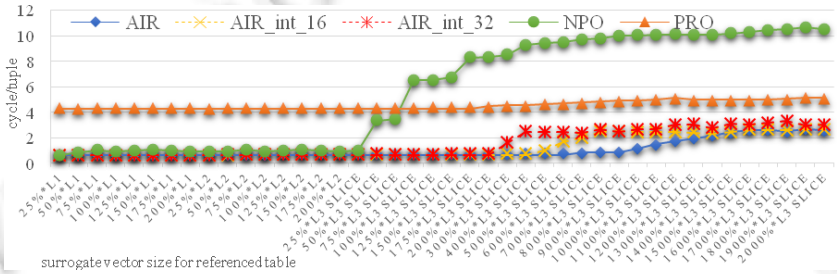


Fig.5 Cache adaptability of different join algorithms on CPU

图 5 CPU 端连接算法的 Cache 适应性

从整体性能曲线来看,Cache-Oblivious 的 NPO 算法是一种对连接数据集大小 Cache 敏感的算法,当连接数据集(哈希表)小于 Cache 时,算法具有较好的性能;而当连接数据集超过 Cache 时,算法执行时间显著增长.Cache-Conscious 的 PRO 算法则显示出连接数据集对 Cache 不敏感的特性,即:不论连接表的大小是否超过 Cache,都产生较为稳定的连接性能.这是因为在哈希连接的分区阶段,较大的数据集已划分为适合 Cache 大小的数据集,其后的哈希连接操作都是 Cache 优化的.

SSB,TPC-H,TPC-DS 中不同的维表与事实表的外键连接操作可以映射为图 5 中不同比例代理向量的连接操作.通过我们对 SSB,TPC-H,TPC-DS 的分析,维表通常较小,对于 OLAP 应用场景,NPO 算法在较大范围内优于 PRO 算法;PRO 算法是一种性能稳定的算法,主要适合大表连接操作,在与较小的表连接时有过度优化的缺点.

与 NPO 和 PRO 算法相比,AIR 算法表现出一种 Cache-Friendly 的平衡性能:当代理向量小于 Cache 时,AIR 算法像 NPO 算法一样自动获得较高的性能;当代理向量超过 Cache 时,AIR 算法像 NPO 一样增加了内存访问代价,但其性能衰减幅度极大地低于 NPO,并逐渐像 PRO 算法一样趋于稳定.

AIR 算法中,每次外键的代理向量参照访问只产生一个 Cache miss,而哈希连接则可能产生多个 Cache miss 以及相应的哈希处理 CPU 代价.因此,AIR 算法执行时间的上升阶段比 NPO 算法更为平缓.不同的代理向量大小决定了 AIR 算法执行时间上升期的起始位置,int_32,int_16 以及 int_8 性能曲线的斜率依次递减,执行时间上升阶段依次延长.

3.2 Phi协处理器端连接算法性能测试

由于 Xeon Phi 协处理器与 Xeon 处理器的兼容性,我们将 CPU 端的 AIR 算法通过 ICC 编译器编译为 Xeon

Phi 端执行的程序,将执行程序复制到 Phi 协处理器内存执行.对应的 NPO 与 PRO 的 Xeon Phi 协处理器端程序使用文献[2]提供的源码编译执行.我们首先执行第 3.1 节中的性能测试,图 6 显示了 AIR,NPO,PRO 分别在多核 CPU,Phi 协处理器上的查询性能曲线.由于 CPU 与 Phi 协处理器主频不同,因此我们使用每记录纳秒值(ns/tuple)作为查询性能指标.

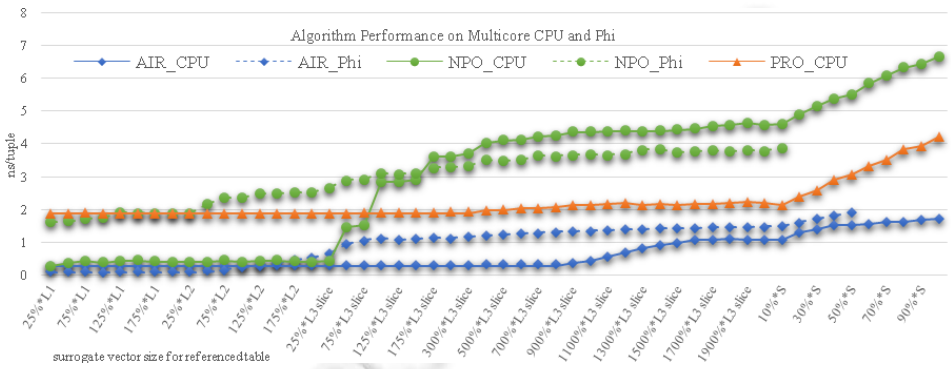


Fig.6 Cache adaptability of different join algorithms on Phi coprocessor

图 6 Phi 协处理器端连接算法的 Cache 适应性

NPO 算法对于低于 Cache 容量的哈希表连接性能较好,当连接表较大时,连接性能下降较快.连接表较小时,PRO 显示了稳定的查询执行性能;但随着连接表大小的增长,radix 分区代价随之增长,查询性能受分区代价的影响而显著下降,性能曲线上升幅度与 NPO 在大表连接时类似.AIR 在构建代理向量时代价较低,在大表连接时仍然保持较高的性能,查询执行时间增长缓慢,查询性能更加稳定.

图 6 中虚线为算法在 Phi 协处理器上的查询性能,当代理向量小于 L2 Cache 时,Phi 协处理器上的 AIR 性能优于 CPU.主要原因是:尽管 Phi 协处理器的主频低于 CPU,但 240 线程并行执行的性能优于 CPU 端 40 线程并行执行性能.Phi 协处理器只有 512 KB 的环形连接 L2 共享 Cache,不像 CPU 处理器拥有较大的 L3 Cache,当代理向量超过 L2 Cache 时,CPU 端的性能优于 Phi 协处理器端,主要原因是 CPU 的 Cache 访问优于 Phi 协处理器的并发多线程内存访问性能.NPO 算法在较小连接表时 CPU 端的性能更优,其主要原因是:Phi 协处理器大量并发的线程,增加了共享哈希表构建时的并发访问代价以及 Phi 协处理器主频与核心性能较低.当连接表较大时, Phi 协处理器的高并发访问表现了较好的内存访问性能,查询性能优于 CPU 端.PRO 算法的 radix 分区操作导致内存需求加倍,*SF*=100 的测试数据集在 Phi 协处理器上 S 表大小为 4.8GB,分区操作需要额外的 4.8GB 内存,直接导致内存空间不足而中断测试.在 Phi 协处理器算法设计中,除性能因素之外,Phi 协处理器有限的内存需要算法具有良好的内存效率,在 CPU 端性能表现较好的 PRO 算法的大数据处理能力显著低于 NPO 和 AIR 算法,其应用受到较大的制约.

3.3 Phi协处理器端连接算法性能对比分析

为进一步完整地对比 AIR,NPO 与 PRO 算法在 Xeon Phi 协处理器上的整体性能,我们缩小 S 表的大小,使 3 种算法能够完成全部的测试任务.经过测试,最终确定,在 Xeon Phi 协处理器 8GB 内存中支持的最大表为 200 000 000 行.我们将 S 表大小固定为 200 000 000 行,测试 AIR 算法代理向量按 L1 Cache 内、L2 Cache 内、L2 共享 Cache 内(注:Phi 协处理器的 L2 Cache 为 512KB,共 60 个)以及与 S 表行数 10%递增的比例设置 R 表行数.图 7 显示了 3 种算法在 Phi 协处理器上整体性能曲线,当 R 表哈希表小于 L2 Cache 时,NPO 算法的性能优于 PRO;当 R 表哈希表超过 L2 cache 时,PRO 性能优于 NPO,均保持比较稳定的性能;当 R 表超过 S 表大小 20%时,PRO 平均查询执行时间增长速度超过 NPO;当 R 表与 S 表大小相同时,PRO 算法与 NPO 算法的性能几乎相同.与 NPO 和 PRO 相对应,AIR 算法对 L1 Cache 大小较为敏感,当代理向量超过 L2 Cache 时,查询执行时间开始增长;代理向量大小在 Phi 协处理器共享 L2 Cache 大小范围内时,连接执行时间呈现非常缓慢的增长趋势;当

代理向量大小超过共享 L2 Cache 时,查询执行时间增速提高,但其增长速度显著低于 NPO 与 PRO.

在最大数据集连接操作中,NPO 算法在 Phi 协处理器上的性能优于 CPU,连接总时间分别为 1 227 785μs 和 1 353 713μs,PRO 算法在 Phi 和 CPU 上的执行时间为 1 188 729μs 和 841 234μs,AIR 算法在 Phi 和 CPU 上的执行时间为 407 087μs 和 327 758μs,PRO 与 AIR 算法在 Phi 上的查询性能略低于 CPU.需要注意的是:实验中使用两块 Intel Xeon E5-2650 v3@2.30GHz 10 核 CPU 和一块 Intel Xeon Phi 5110P@1.053 GHz60 核协处理器,以接近的价格获得了相近的连接操作性能.

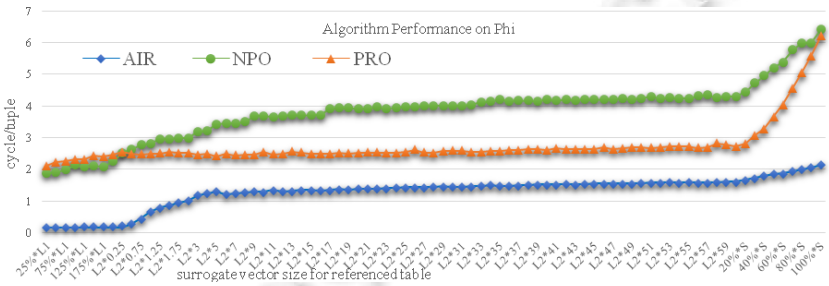


Fig.7 Comparison of different join algorithms on Phi coprocessor

图 7 Phi 协处理器端连接算法性能对比

4 结束语

随着 Xeon Phi 协处理器成为高性能计算的主流处理器,Xeon Phi 协处理器成为连接优化技术研究的新平台.本文的贡献体现在 3 个方面:首先,面向 OLAP 模式特点提出了代理键索引机制,并详细阐述了代理键索引机制的实现及维护技术;然后,通过深入的实验,揭示了基于代理向量参照访问的 AIR 算法的 Cache-Friendly 特征,揭示了 AIR 算法融合了 NPO 小数据和 PRO 大数据时的连接性能特征,验证 AIR 优于 NPO 和 PRO 算法的性能;最后,通过 Phi 协处理器平台上全面的性能对比测试,展现了 3 个算法各自的性能曲线特征,清晰地描述了各个算法在不同连接表大小下的性能特征,为 Phi 协处理器 OLAP 查询处理中连接算法的技术选择提供了详实的实验数据.

从整体实验结果来看,AIR 算法通过 OLAP 模式特点简化连接操作实现技术,通过代理向量参照访问减少内存随机访问数量和 CPU 计算代价,其算法性能与 Cache 大小、并行执行线程数量等硬件参数相关度较高,既能够适应多核 CPU 较大容量 Cache 的优化,也能适应众核协处理器较高并发线程的优化,具有较好的适应性.从总体来说,AIR 算法特点更加适合 Xeon Phi 众核协处理器弱化 Cache 和核心性能、强化并发多线程内存访问的特征,基于代理向量访问的操作简化了算法设计,提高了算法代码执行效率,并且相对于 NPO,PRO 等经典的数据库连接算法更加易于代码改写与平台迁移.

在未来的工作中,我们将继续探索在 Phi 协处理器平台上全面的 OLAP 查询优化技术,尤其是面向 PCI-E 带宽性能瓶颈的 CPU-Phi 协同 OLAP 优化技术.

References:

[1] http://www.expreview.com/44000.html

[2] 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]

[3] 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]

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

[5] 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]

[6] Morgan TP. Intel mates FPGA with future Xeon server chip. 2014. http://www.enterprisetech.com/2014/06/18/intel-mates-fpga-future-xeon-server-chip/

[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, Ozsu T. Main-Memory Hash joins on multi-core CPUs: Tuning to the underlying hardware. In: Proc. of the 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] He BS, Yang K, Fang R, Lu M, Govindaraju NK, Luo Q, Sander P. Relational joins on graphics processors. In: Proc. of the SIGMOD. New York: ACM Press, 2008. 511−524. [doi: 10.1145/1376616.1376670]

[11] 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]

[12] 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.

[13] 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 & Data Engineering, 2016,28(4):1061−1074. [doi: 10.1109/TKDE.2015.2499199]

[14] https://en.wikipedia.org/wiki/Surrogate_key

张宇(1977－),女,黑龙江绥化人,博士,副教授,主要研究领域为 GPU 数据库,数据仓库,OLAP.

陈红(1965－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据仓库,数据挖掘,传感器网络.

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

王珊(1944－),女,教授,博士生导师,CCF 会士,主要研究领域为高性能数据库,内存数据库,数据仓库.