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.

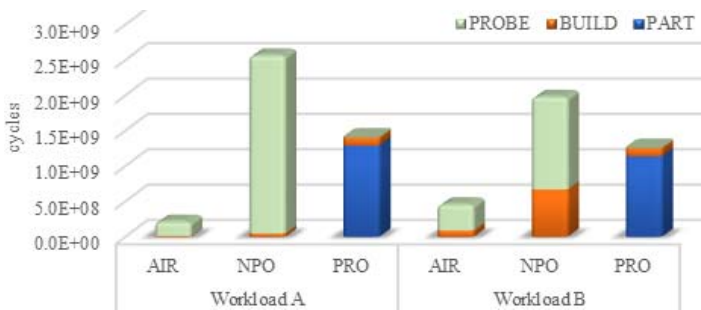

Fig.3 Breakdown time of join operations

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.

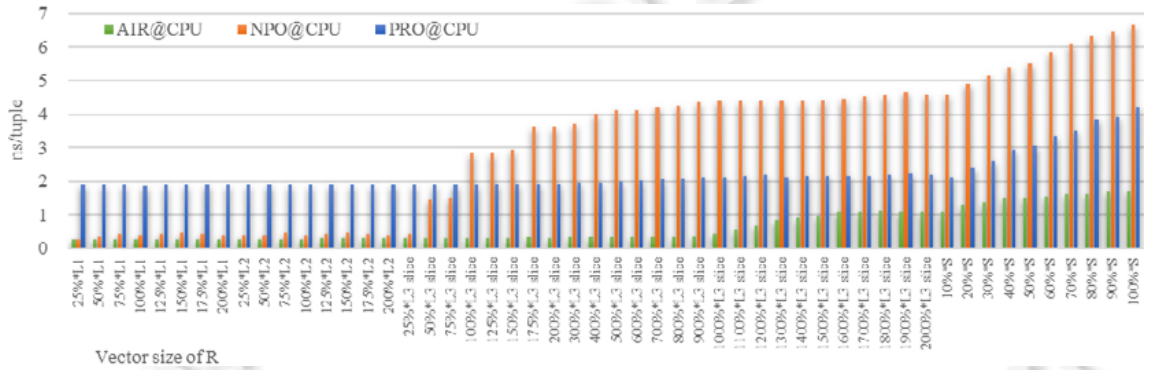

Fig.4 Performance characteristics of different join algorithms on CPU

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.

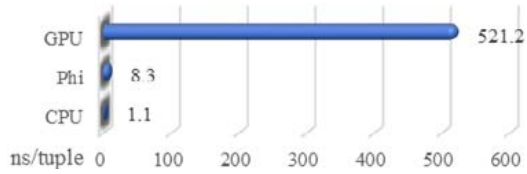

Fig.5 Performance of single-thread AIR

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.

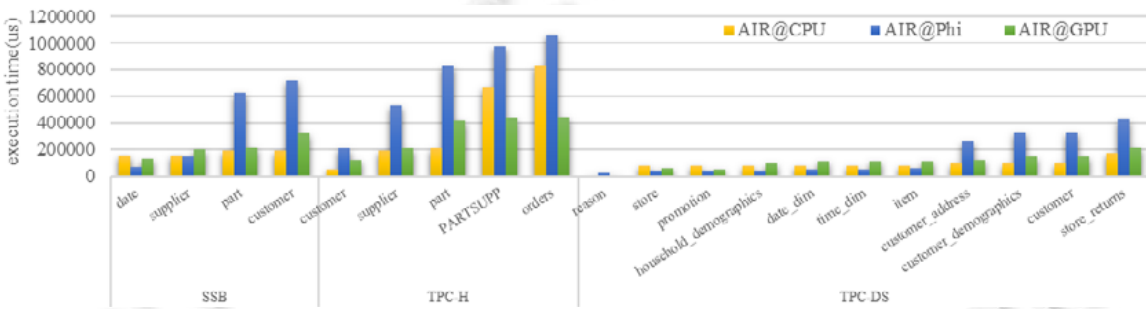

Fig.6 Join performance of AIR with SSB, TPC-H and TPC-DS on multicore, Phi and GPU

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.

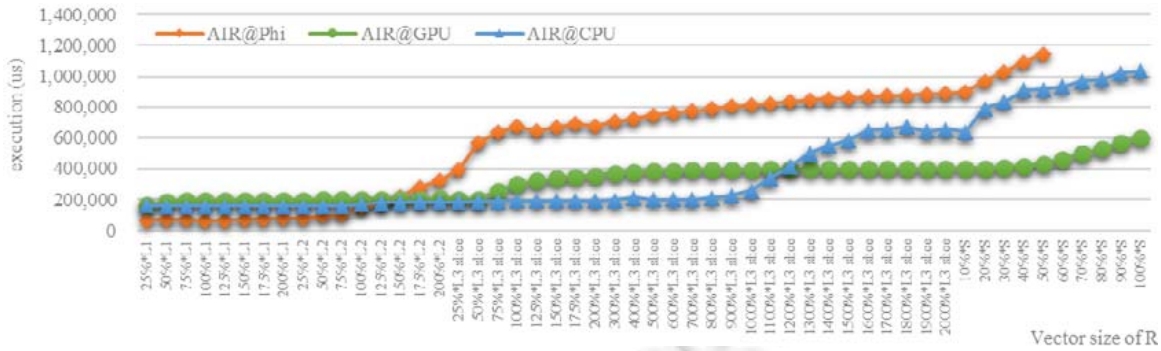

Fig.7 Performance analysis of AIR on CPU, Phi and GPU

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.

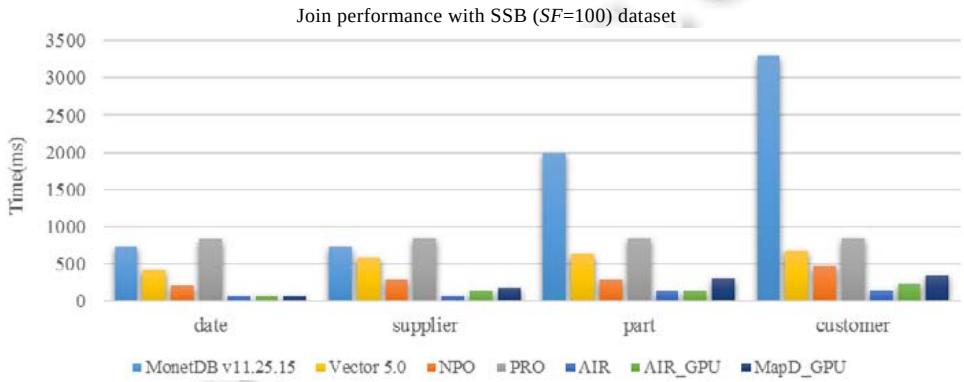

Fig.8 Comparision of join performance

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.