计 算 机 学 报
CHINESE JOURNAL OF COMPUTERS

面向 GoldenX 软硬协同优化的异构加速列式存储引擎研究

屠要峰[1),2)] 陈河堆[2)] 王涵毅[2)] 闫宗帅[2)] 秦小麟[1)] 陈兵[1)]

[1)](南京航空航天大学 计算机科学与技术学院, 南京市 211106)

[2)] (中兴通讯股份有限公司, 南京市 210012)

摘 要 万物互联时代到来，传统数据处理方式逐步向云边协同数据处理的方式演进。云端和边缘侧复杂的负载连接和快速增长的数据量，对数据库系统性能和 SQL 加速提出了新的挑战。另一方面，以 GPU/FPGA 异构算力、NVM（non-volatile memory，非易失内存）存储、RDMA（remote direct memory access，远程直接内存存取）网络为代表的新型硬件技术的快速发展和应用，将对现有软件架构体系产生革命性的影响，也为数据库系统的演进和性能提升提供了变革基础并指明研究方向。如何利用这些不断涌现的新型硬件技术来为工业界使用的真实数据库系统赋能是一个十分重要且有挑战的课题。首先介绍了中兴通讯 GoldenX 数据库系统架构和列式存储引擎的设计，然后重点阐述 GoldenX 列式存储引擎利用新型硬件特性在计算层和存储层进行的软硬件协同设计和优化，主要包括：（1）将压缩/解压、加密/解密任务从 CPU 卸载到 FPGA，利用 FPGA 的可编程特性，设计专用 MISD（multiple instruction stream single data stream，多指令流单数据流）架构处理器，采用“软件接口级-计算核心级-功能模块级”三级流水线设计，提高数据流处理的效率；（2）为列式存储定制向量化执行引擎，充分利用 CPU/GPU 的 SIMD（single instruction multiple data，单指令多数据流）新特性优化传统火山模型，降低了函数调用开销；（3）对 SQL 执行引擎进行优化，动态评估和利用 GPU 计算资源，采用 JIT 编译技术，将过滤/排序/聚集等具有矩阵运算特征的统计分析型 SQL 运算任务下推到 GPU 上，利用 GPU 的超高并行计算能力提高查询分析性能。实验表明，本文提出的软硬件优化方法有效提升了 GoldenX 的系统性能，TPC-H 基准测试场景的 22 个查询中，优化后的 GoldenX 性能比优化前提升了 2.5~10 倍，比开启向量化的 openGauss 执行时长减少了 17%~78%。

关键词 数据库；存储引擎；GPU；FPGA；OLAP

中图法分类号 TP311.13

Research on Heterogeneous Accelerated Columnar Storage Engine Based on Co-optimization of Software and Hardware for GoldenX

Tu Yaofeng[1),2)] Chen Hedui[2)] Wang Hanyi[2)] Yan Zongshuai[2)] Qin Xiaolin[1)] Chen Bing[1)]

[1)](Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106)

[2)] (ZTE Corporation, Nanjing 210012)

Abstract With the advent of the Internet of Everything era, traditional data processing methods have gradually evolved towards cloud-edge collaborative data processing. The complex load connections and fast-growing data volume on the cloud and edge sides pose new challenges to database system performance and SQL acceleration. On the other hand, the rapid development and applications of new hardware technologies represented by GPU/FPGA heterogeneous computing power, NVM (non-volatile memory) storage, and RDMA(remote direct

本课题得到国家重点研发计划项目(No. 2018YFB1003301)、江苏省重点研发计划项目（BE2019012）资助. 屠要峰(通讯作者)，男，1972 年生，博士研究生，研究员，CCF 高级会员（74168M）,主要研究领域为大数据、分布式系统和机器学习等.E-mail: 13605151819@qq.com. 陈河堆，男，1972 年生，硕士，高级工程师，CCF 会员（D7767M），主要研究领域为数据库、分布式系统和异构计算等，E-Mail：chenhedui@163.com
. 王涵毅，男，1982 年生，硕士，高级工程师，主要研究领域为数据库、分布式系统和异构计算等，E-Mail：304538363@qq.com. 闫宗帅，男，1987 年生，硕士，高级工程师，主要研究领域为数据库、非易失性存储，E-mail：yanzongshuai87@126.com.. 秦小麟，男，1953 年生，教授，博士生导师，CCF 高级会员（ES200009060S），主要研究领域为时空数据库、分布式数据管理与安全.E-mail:qinxcs@nuaa.edu.cn.陈兵，男，1970 年生，教授，博士生导师，CCF 高级会员（33601S），主要研究领域为大数据、云计算和认知无线电网络.E-mail:Cb_china@nuaa.edu.cn.

memory access) network, will have a revolutionary impact on the existing software architecture system, and will also provide a foundation for the evolution and performance improvement of the database system and point out the research direction. How to use these emerging new hardware technologies to empower the real database system used in the industry is a very important and challenging topic. At first,this paper introduces the architecture of ZTE GoldenX database system and the design of columnar storage engine, and then focused on the software and hardware co-design and optimization of the GoldenX columnar storage engine using new hardware features at the computing layers and storage layers, including: (1) Offload the tasks of compression/decompression, encryption/decryption from the CPU to the FPGA, and use the programmable characteristics of FPGA to design a dedicated MISD (multiple instruction stream single data stream) architecture processor, using "software interface level- Computing core level-functional module level" three-stage pipeline design improves the efficiency of data stream processing; (2) Customized vectorized execution engine for column storage, making full use the new features of CPU/GPU namely SIMD (single instruction multiple data) to optimize the traditional volcano model, reducing the cost of function calls; (3) Optimize the SQL execution engine, dynamically evaluate and utilize GPU computing resources, and the JIT compilation technology is used to push statistical and analysis SQL operation tasks with matrix computing characteristics (such as filtering/sorting/aggregation) down to the GPU, so that the ultra-high parallel computing capabilities of the GPU can be used to improve query and analysis performance. Our experiments show that the software and hardware optimization method proposed in this paper can effectively improve the system performance of GoldenX. Among the 22 queries in the TPC-H benchmark test scenario, the system performance after optimization is 2.5 ~ 10 times higher than that before optimization. Compared with openGauss with vectorization turned on, the optimized GoldenX reduces the execution time by 17%~78% .

Key words Database; storage engine; GPU; FPGA; OLAP

1 引言

随着物联网的快速发展和 5G 的到来，我们迎来了一个全新的万物互联时代，全球的数据量呈爆炸式增长。传统数据处理机制逐步向云边协同数据处理的方向演进，把计算放在离数据最近的地方，云端算力下沉，终端算力上移，算力在边缘端汇聚，充分发挥云-边-端协同计算的效用。关系数据库作为重要的数据基础设施，也在适应这种变化，出现了从本地部署到云数据库部署，从交易型行式存储到分析型列式存储，从SQL到NoSQL再到NewSQL等不同应用形态，这对数据库系统的性能不断提出新的挑战。

性能是数据库系统最重要的指标，也是数据库研究长期关注的目标[1]。数据库的工作负载主要分为 OLTP（online transaction processing，在线事务处理）和 OLAP（online analytical process，在线分析处理）两大类，这两类工作负载的差异非常明显。OLTP 操作涉及数据少，但实时性和事务要求高，并发量大；OLAP 操作实时性和事务要求低，但涉及数据量大，并且查询模式不固定，很难通过索引来覆盖。早期的数据库是没有 OLTP 和 OLAP 类型之分的，在一个数据库里进行 OLTP 和 OLAP 类型的数据相关的操作。

随着数据量变大，传统数据库系统难以同时支撑 OLTP 和 OLAP 两种负载需求，这主要是受到系统性能的制约。一方面是系统资源限制，传统数据库系统多是单机系统，CPU、内存、磁盘容量和速度都有限，同时支持 OLTP 和 OLAP 部署会带来 1+1<1 的效果。系统还缺乏细粒度资源管理技术以使得资源在公平性、确定性和最大资源利用率之间取得平衡。另一方面是这两种负载对数据库的技术要求不同。OLTP 系统的瓶颈主要在 I/O，主要优化手段是索引和缓存，借助索引快速地定位到相关记录，借助缓存将频繁访问和最近访问的数据块放在内存，从而减少磁盘访问次数。OLAP 系统的瓶颈在于算力和存储，算力的优化主要是利用 CPU 多核/众核（multi-/many-core）或者异构算力加速，存储的优化手段主要是列式存储和数据压缩等。

硬件技术的飞速发展对数据库性能提升具有重要推动作用[1]。崔斌等人[2]总结了基于新硬件技

术的数据管理方法，认为目前研究较多的新硬件主要是高性能和专用处理器、高速网络、大内存和非易失性内存等。潘巍等人[3]、肖仁智等人[4]认为融入 NVM 的新型存储环境有望跨域 CPU 与外存之间的性能鸿沟，消除计算机系统中制约上层软件设计的 I/O 瓶颈。但是数据库系统的高性能不仅仅取决于存储，还取决于以可重构逻辑电路（field programmable gate array，FPGA）、通用图形处理单元（graphic process unit，GPU）等为代表的新型算力。

学术界和工业界存在着的异构计算加速硬件可以概括为多核/众核、专用集成电路（application specific integrated circuit，ASIC）、通用图形处理单元 GPU、可重构逻辑电路 FPGA 等四类。它们往往表现出不同的并行粒度，适用于不同的应用场景，并且也能够相互结合形成异构系统来充分发挥不同加速器件的处理能力[5]。多核/众核处理器可以更好地支持并行运算，大大提高了 CPU 对数据密集型计算的处理能力。GPU 在处理能力和存储器带宽上天然具有优良的并行处理能力。

本文介绍了中兴通讯新一代关系数据库系统 GoldenX 应用 CPU/GPU/FPGA 等新型硬件特性对列式存储引擎进行软硬协同设计优化所做的研究与实践，主要的工作和贡献体现在：（1）面向异构加速的列式存储引擎数据组织方式优化及实现；（2）“软件接口级-计算核心级-功能模块级”三级流水线并行处理设计，将数据流的压缩/解压、加密/解密等计算密集型操作从 CPU 卸载到 FPGA；（3）设计向量化执行引擎[6]，改进传统火山（volcano-style）模型[7]，并利用 SIMD（single instruction multiple data，单指令多数据流）[8]新特性实现并行处理，提升查询计划执行效率；（4）基于 GPU 的 SQL 加速设计，采用 JIT 机制动态生成 GPU 二进制代码，将扫描、连接、分组等具有矩阵运算特征的任务下推到 GPU 上执行，利用其超高核数的并行处理能力提升 SQL 执行效率。

本文第 2 节介绍研究背景和相关工作。第 3 节介绍 GoldenX 列式存储引擎设计，重点描述基于 FPGA 的数据流加速、向量化执行引擎和基于 GPU 的 SQL 加速等软硬协同设计技术；第 4 节通过实验进行将压缩/解压计算卸载到 FPGA 对比测试、将扫描/连接/分组运算卸载到 GPU 对比测试、向量化执行引擎对比测试、列式存储引擎综合优化前后性能对比测试以及与 openGauss 的性能对比测试，并对测试结果加以分析；最后对本文进行总结。

2 背景与相关工作

20 世纪 70 年代是关系数据库理论发展的黄金年代。关系模型把现实世界抽象为二维表，借助关系代数的集合运算和关系运算，具有强大的查询表达能力，有力地支撑了信息时代早期对数据库的使用需求。关系模型迅速取代了层次模型和网状模型，成为事实上的数据库标准，这个时期诞生的数据库系统绝大部分都属于关系数据库，包括 INGRES[9]（PostgreSQL 前身）、Oracle 等。

上个世纪 90 年代中期，随着互联网的快速发展，数据量急剧增加。关系数据库因为严格的事务一致性要求制约了系统扩展能力，低成本的弹性扩展成为数据库的首要需求。在此背景下，以 Google 为代表的互联网公司开发了 NoSQL 数据库，典型的系统有 BigTable[10]、Dynamo[11]、Cassandra[12]等，在牺牲数据库的事务特性和某些 SQL 功能的前提下获得了较强的可扩展性。

2000 年之后，随着云计算和移动互联网的大规模普及和深入应用，产业界迫切需求新型的数据库系统，既具有 NoSQL 数据库的高性能和可扩展性，又保持关系数据库强大的 SQL 能力和可靠性。新型硬件技术的出现提供了变革的基础并指明了研究方向，即新型硬件技术的应用和优化，包括新型算力、新型存储 PM（persistent memory，持久内存）[13][14]和新型网络 RDMA（remote direct memory access，远程直接内存存取）等。产业界进行了积极的探索，催生了一批软硬一体化设计的新型数据库系统，比如 VoltDB[15]、SAP HANA[16]、Oracle RAC、Teradata 等。

使用专用硬件对数据库加速的研究有很多，研究思路是使用 FPGA/GPU 与 CPU 互补，主要集中在将数据分析处理中的表扫描卸载到 FPGA[17][18]或 GPU[19][20]构建的专用加速器上。也有工作在研究卸载更复杂的查询处理的可能性[21][22]。

Zhang T 等人[23]提出了一种在基于 LSM-Tree（log-structured merge tree）的键值存储中将 compaction 卸载到 FPGA 的方法，从而降低 compaction 对 CPU 的影响和对上层吞吐抖动的影响，但这种实现方法不具有通用性。Huang G 等人[24]将上述方法与 X-Engine 集成实现了一种最新的 LSM-Tree 型 KV 存储引擎，应用于阿里巴巴的大规

模电子商务交易处理场景，较好地解决了因促销引起的海啸问题和数据热点快速移动的问题。

Owaida M 等人[25]提出一种 CPU-FPGA 混合数据库系统框架 Centaur，允许动态采集信息，以 pipeline 方式通过用户自定义函数将操作并发和动态地卸载到 FPGA 上以加速处理数据流。但是 Centaur 基于内存数据库系统开发，不适合用在大数据量的 OLAP 场景。

J.Doetal 等人[26]提出一种智能存储引擎，将查询处理下推到 SSD 上，但是处理器和 SSD 很快就成为性能瓶颈。L. Woods 等人[18]在此基础上结合 FPGA 提出一种智能存储引擎 Ibex，支持将 GROUP BY 和 WHERE 等复杂查询卸载到 FPGA 加速器，以提高性能和减小能耗。Salami B 等人[27]使用现代 SATA-3、PCIe-3、DDR-3 接口，基于 FPGA 构建了快速查询的引擎 AxleDB，该引擎通过将查询卸载到 FPGA 上，以加速 FILTER、AGGREGATE、GROUP BY、JOIN、SORT 等复杂查询。然而，SQL 查询分析是算力密集型操作并带有通用计算和矩阵特征，更适合通过 GPU 进行加速。

T.Kersten 等人[28]认为向量化与编译执行都使数据库查询执行性能得到较大提升，但是这两个模型是不相容的，分别解析了向量化执行和编译执行的原理并进行对比，认为向量化执行模型处理逻辑清晰，能更好地进行并行数据访问和发挥 SIMD 特性，在访问大型哈希表以进行聚集或联接的内存绑定查询中具有优势。Menon 等人[29]则是在编译执行的基础上，在 Pipeline（管道）中插入 Materialization（物化）而将 Pipeline 分割为 Stage（阶段），在 Stage 内采用 tuple-at-a-time 的推送模型，保留了编译执行数据驻留寄存器的优点；在跨 Stage 或 Pipeline 时，则以 Block 为单位传递数据以利用 SIMD 特性。由此在编译执行的基础上融合了向量化，但仅仅是实现了原型系统。张延松等人[30]针对 MapD[31]在面向复杂数据模式时连接操作的性能瓶颈问题，提出基于向量引用 platform-oblivious 内存连接优化技术，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影响，通过索引、数据库约束、更新优化策略、数据压缩等综合技术实现从 hardware-oblivious 连接算法向 platform-oblivious 算法的升级，从而更好地适应异构处理器平台。该技术路线需要在 FPGA 及其他新兴处理器平台上进一步验证。

张宇等人[1]提出一种以数组存储和向量计算为特点的 GPU semi-MOLAP 多维分析处理模型，将一个多维查询划分为在不同数据集上独立的处理阶段，实现了以最小数据传输为目标的基于水平分片的 CPU 和 GPU 协同计算，减小了 PCIe 通道上的数据传输延迟。张延松等人[32]提出基于向量索引的 OLAP 框架，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，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。上述向量化分析处理主要着重于理论模型探讨，尚未沉淀到向量化执行引擎层面，难以在工程上实施应用。Bakkum P 等人[33]结合 CUDA（Compute Unified Device Architecture，统一计算设备架构）对 SELECT 操作卸载到 GPU 上进行了探索。Yuan Y 等人[34]对 GPU 未能应用到主流数据仓库产品的问题进行了研究，结合 CUDA 和 OpenCL 设计实现了查询执行引擎的原型，将查询操作卸载到 GPU 进行优化。骆歆远等人[35]采用行列混合存储架构，在规则挖掘和编码中使用 GPU 作为协处理器并行处理算法提高效率。然而，这些研究需要将数据从持久化介质传输到内存，然后再传输到 GPU，开销很大。

Cao W 等人[36]在云原生关系数据库 PolarDB 中实现了近存储计算，将表扫描等密集访问数据的分析型负载从计算层下推到分布式异构计算架构的存储层物理介质上，有效降低计算节点到存储节点之间的带宽，同时利用存储节点富余的计算能力提高整体的资源利用率。但是这种“计算下推”需要在最终执行侧实现低成本的可计算存储（FPGA+NAND 的软硬件一体化方案），还需要整个软件栈完全支持任务的下推。不但改造工作量大、技术要求高，而且无法使用通用的存储层软硬件。

上述基于 FPGA/GPU 异构加速的数据库系统优化方法，虽然在学术界有着广泛研究和讨论，但偏向于理论模型探讨，没有应用到通用数据库系统。而且，大多数现有工作专注于探索单一异构算力的加速方法。阿里等少数大厂也开展了相关工作，但其优化技术是针对其特有应用场景的，且与其自研数据库的内部实现机制强相关，不具有通用性。本文提出了一种与现有主流数据库架构兼容、综合应用 CPU-GPU-FPGA 新硬件特性在计算层和存储层同时进行优化加速的解决方案，探索在工业数据库中使用多种异构计算资源协同提升系统性

能的方法。

3 GoldenX 列式存储引擎设计

在很多实际应用中，虽然数据量不断膨胀，但具有明显的热点数据，且对冷热数据的访问性能要求不同。使用同一种设备或方法来存储数据存在性能和成本的问题，而使用不同的存储系统来存储数据会带来数据同步、冗余存储等额外问题，因此在一套系统中使用混合存储成为一种必然。

GoldenX 的核心理念是在性能和成本之间取得平衡以获取最大收益，这就要对不同应用场景和热度的数据使用不同的处理方法。

GoldenX 在业界行/列混合存储方案基础上更进一步，把行式存储和列式存储逻辑分离，形成各自独立的行式存储引擎和列式存储引擎，并提出和实现了动态可拔插的多存储引擎架构，以容纳不同类型的存储引擎。针对不同业务场景灵活启用不同存储引擎，OLTP 负载选择行式引擎，OLAP 负载选择列式引擎，对某些特殊应用场景，比如超高性能、实时系统等场景，可选择采用软硬件协同设计定制优化的存储引擎。

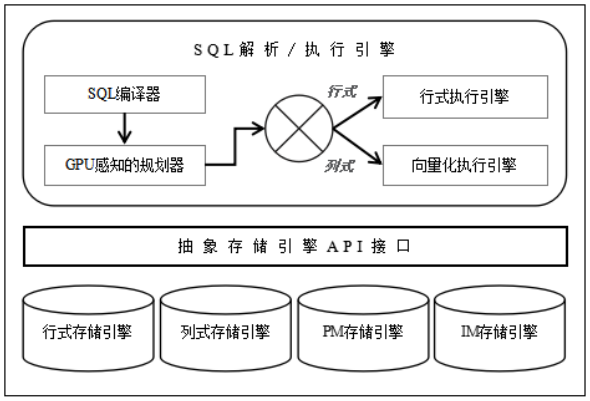


图 1 GoldenX 系统架构

GoldenX 的系统架构如图 1 所示，总体上分成 3 层，最上层是 SQL 解析/执行引擎，中间是一层抽象的存储引擎 API 接口，最底层是各个具体实现的存储引擎。在执行查询计划过程中，SQL 解析/执行引擎只调用抽象的存储引擎 API 函数，这是一些抽象的虚函数，它们会根据传入的实例化存储引擎对象指针自动适配到具体的存储引擎实现函数入口，从而实现对不同存储引擎的透明调用。这种机制屏蔽了不同存储引擎之间的实现差异，使得数据库系统可以根据需要动态挂载和卸载任何一个存储引擎，具有热拔插特性。

针对不同应用场景，我们为 GoldenX 设计了多种存储引擎，有行式存储引擎、列式存储引擎，利用 PM 非易失性和字节可寻址特性开发的高性能的 PM 存储引擎，针对实时系统场景的基于纯内存的低时延的 IM 存储引擎（in-memory storage engine）。结合最新硬件技术和应用场景特点，这些存储引擎各自进行软硬件一体化设计，以达到最佳执行效率。本文重点讨论 GoldenX 列式存储引擎软硬件一体化设计的关键技术。

3.1 面向异构加速的列式数据存储优化

在 OLAP 应用场景中，一般很少对数据进行更新，也很少会查询整行记录的所有字段，大多情况下只对少数几个列进行扫描分析。行式存储按行组织数据，哪怕只访问一个列，也会取出所有其它无关列的数据，从而产生大量无效 I/O。列式存储按列组织数据，具有压缩比高、投影灵活等优势，一方面可以节约存储空间，另一方面只读取关联列数据，无关列不会被读取，避免无效 I/O。

GoldenX 每个列使用一组单独的文件进行存储。列式存储文件由一个个 Block（也称为数据页）组成，如图 2 所示，每个 Block 大小在建表前指定，取值范围在 4KB ~ 4MB 之间。由于每次调用 FPGA 请求的最低响应时延是毫秒级别，因此，当启用 FPGA 异构计算加速时，系统自动把 Block 设置成最大值（4M）。Block 包含页头（header）和数据体（body）两部分。header 存放该 Block 的元数据信息，包括压缩算法、起始物理行号（rownum）、最大值、最小值等，header 本身不压缩。body 是根据指定的压缩算法压缩后的列数据集合。body 是压缩的基本单位，一个 body 对应一个压缩单元。在某些场景下，body 数据需要加密存储，因此在压缩之后还需进行加密处理。为简化叙述，本文压缩/解压操作隐含了可能的附加加密/解密操作。

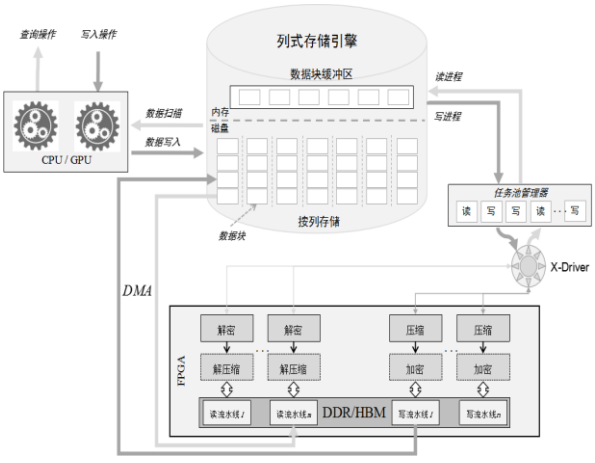


图 2 基于 FPGA 数据流加速的列式存储引擎

压缩和解压都需要耗费计算资源，应该尽量消除不必要的压缩/解压操作。数据查找时，可以根据上述 Header 统计信息做初步过滤，如果待查找的列值不在当前 Block 范围内，则跳过该 Block。对于匹配的 Block，先解压再扫描，筛选出满足查询条件的字段记录。在每个字段记录中除了列值之外还额外存储了一个 rownum 属性，通过它执行引擎把这些离散存储的字段值关联在一起，生成一条条记录行。

传统存储引擎 I/O 模块维护着一个数据页缓存区，当执行引擎发生缺页时通知 I/O 模块从磁盘加载新页，磁盘中的数据页是压缩的，加载之后需解压还原；当接收到 Checkpoint 线程的刷脏页指令时，I/O 模块对数据页先进行压缩，再刷写回磁盘。CPU 完成上述过程的任务调度管理，并实际执行数据的压缩/解压/拷贝/传递等操作。为了利用 FPGA 处理数据压缩和解压任务，GoldenX 在列式存储引擎的 I/O 模块增加了 FPGA 设备的感知能力，当检测到 FPGA 设备可用时，CPU 把上述数据处理任务卸载到 FPGA 设备上，然后实时监控 FPGA 设备任务完成情况，并进行必要的干预。

大多数情况下，OLAP 应用自身不会生成原始数据，数据主要从外部数据源抽取转换后批量导入，因此，批量入库性能一直是衡量 OLAP 数据库的一个关键指标。传统的面向按行插入/更新的数据组织方式不能支持超高性能数据导入，这主要受 2 个因素制约：（1）需要定位每行记录的待插入或更新位置；（2）物理盘的顺序写入性能远高于随机写入性能，特别是机械盘更明显，而频繁定位文件位置导致磁盘访问退化为随机 I/O。为了克服上述局限性，我们在设计列式存储引擎时采用追加写（append-only）的数据文件组织机制，其算法思想如下：

1）不论是新增记录，还是对已有记录进行更新，都在各个列存文件末尾以追加方式写入最新字段值；

2）为了支持更新和删除操作，每个 Append-Only 列存表配置了一个辅助的行映射表，当处理 DELETE 和 UPDATE 操作时，不会直接在数据文件中更新或者删除对应的字段记录，而仅仅在行映射表的 bitmap 位图上置上相应标记位；

3）对于 INSERT 操作，在位图对应比特位上打上有效标记；

4）对于 DELETE 操作，在位图对应比特位上打上删除标记，被标记删除的记录将交由后台管理线程统一回收清理；

5）UPDATE 操作是 DELETE 和 INSERT 两步组合实现。

虽然列式存储方式已显著提升 OLAP 场景数据库处理性能，但是还有两个方面需要改进优化。一是传统行式执行引擎处理不能发挥 CPU/GPU 新特性和列式存储模式的执行效率，二是对于计算密集型场景 CPU 高负荷成为关键瓶颈。GoldenX 通过引入向量化执行引擎、GPU/FPGA 异构计算对上述问题进行了优化和解决。

3.2 基于FPGA的数据流加速

OLAP 查询分析操作往往涉及到大范围表记录扫描，需解压大量的压缩块，这将消耗 CPU 算力，而 CPU 作为一种通用型算力，并不擅长计算密集型场景。在传统存储引擎实现中，压缩/解压、加密/解密等数据流处理任务都由 CPU 直接完成，难以大规模并行处理，效率低下。生产环境监测和实验室测试结果都证实，在分析型计算场景中，CPU 最容易成为制约性能的一个关键因素。

为了解决上述问题，采用的解决方案是尽量降低 CPU 负荷，把它不擅长的计算任务直接卸载到其他异构计算硬件上，让 CPU 有更多时间去处理调度之类的通用型计算任务。异构计算处理器类型较多，在选型方面，重点考察了 GPU、FPGA 和 ASIC 这三类处理器。最终选择 FPGA 作为加速数据流处理的辅助硬件，原因有四：

1）高并行处理：流水线并行和数据并行（GPU 只有数据并行）；

2）可编程特性：算法和处理逻辑可随时更新（ASCI 烧制完后处理逻辑不可改变）；

3）MISD（multiple instruction stream single data stream，多指令流单数据流）特性：压缩/解压、加密/解密具有明显的多指令流单数据流（即 MISD）处理特征，可利用 FPGA 的可编程特性把它设计成一个 MISD 架构处理器。

4）低功耗，高性价比[5]：在功耗上，同等负载下的 FPGA 的功耗大约是 GPU 的四分之一，CPU 的一半左右；在价格上，一块 FPGA 芯片的价格和 CPU 相当，但在相同时间内 FPGA 比 CPU 可以更高效地完成计算密集型数据流处理任务。

每块 FPGA 卡都配置了大量的计算硬件资源，GoldenX 把它们设计成 4 类计算核心（computing

kernel)：压缩（compression）、解压（decompression）、加密（encryption）、解密（decryption），然后把这些计算核心编排成多条读流水线和写流水线，如图 2 下半部分 FPGA 卡所示。考虑到 OLAP 场景读多写少的特点，我们设计的读流水线数量要远多于写流水线数量。

FPGA 带有大容量的卡内独立内存（HBM），或外挂 DDR 内存，可支撑高强度的并行计算任务。GoldenX 放弃了传统的串行化任务调度策略，采用基于多流水线的全并行调度策略，在 FPGA 硬件适配层之上构建了一个并行化的计算任务池。数据库执行引擎把计算任务异步地发送到任务池，任务池管理器（tasks pool manager）通过 FPGA 驱动程序（X-Driver）把批量待处理任务下推到 FPGA 卡中的多个流水线，让多个计算核心并行作业，从而最大限度地挖掘 FPGA 计算潜力。

在查询流程中，执行引擎读取一个数据块的算法如下：

算法 1 数据块读取算法 ReadBlock.

输入：表对象标识 oid，数据块号 bno

输出：数据块缓冲区 block

```
1. IF BlockIsInBuffer(tid,bno) THEN
     //已在缓冲区则直接返回结果
2.   block ← BlocksBuffer[tid][bno];
3. ELSE
     //生成计算任务，指定回调函数 PostRead
4.   task←CreateTask(tid,bno,PostRead);
     //将读计算任务发送给任务池管理器
5.   TasksPoolManager.PushTaskToPool(task);
     //将批量计算任务下推给 X-Driver
6.   TasksPoolManager.BatchToXDriver();
     //等待数据块读取完毕
7.   WaitForDataAvailable();
     //数据块读取结束后通过回调函数唤醒本进程
8.    IF 读取失败 THEN
9.      block ← null; //返回空
10.   ELSE
        //FPGA 读取成功，数据块已调入缓冲区
11.     block ← BlocksBuffer[tid][bno];
12.   END IF
13. END IF
```

在查询流程中，任务池管理器 TasksPoolManager 汇总多个并发处理线程同时发送过来的一批读写计算任务，一次性通过 X-Driver 下推给 FPGA 卡，以发挥 FPGA 多流水线并行作业能力。FPGA 根据表对象标识 oid 和块号 bno 在 NVMe SSD 上定位数据块位置，然后进行解密和解压缩运算，处理结束后把还原后的数据块写入内存中的数据块缓冲区，并通过回调函数通知上层读进程。

写入流程是查询流程的逆过程，由列式存储引擎通过写进程，经过任务池管理器，再经过 X-Driver，驱动 FPGA 多条写流水线并行压缩和加密数据块，FPGA 通过 DMA 方式直接写入 NVMe SSD 中，最终处理结果通过中断方式和回调函数逐级返回给上层处理流程。

3.2.1 FPGA 片内多流水线设计

在 FPGA 内部，计算任务是通过烧制在 FPGA 内部的计算核心实现的，一个计算核心就是一个独立的调度单元。对于复杂的计算任务，往往需要多个计算核心协作才能完成。

主机对 FPGA 的控制是通过轻量级硬件协议 AXI-Lite 实现的，主机端应用通过 AXI-Lite 向计算核心发送数据读取、写入和同步等命令，计算核心在接受命令后通过 HBM 读入数据，利用固化的代码逻辑进行处理和协同，最终计算结果再通过 HBM 返回。

为了充分挖掘 FPGA 硬件潜力，GoldenX 列式存储引擎分别在接口软件层、计算核心层、功能模块层构建了三级并行流水线：

1）接口软件级：由任务池管理器和 X-Driver 组成，它们把数据流不同处理步骤的计算核心串联在一起，构建成一个个任务队列，每个任务队列是一条流水线，如图 2 所示，不再赘述；

2）计算核心级：每个计算核心拆分成多个功能模块，构成流水线结构，它们依次处理数据流，如图 3 所示；

3）功能模块级：对于个别存在性能瓶颈的关键功能模块，再将其拆成成子功能模块，并构建成多条并行流水线。

下面以 zlib 压缩算法为例，介绍计算核心级的并行流水线设计。zlib 算法包含 3 个主要步骤：

1）**LZ77 算法压缩**（LZ77）：对输入序列利用 LZ77 算法进行子字符串匹配，输出相应的字符长度和距离流，用小的标记来代替数据中多次重复出现的长串，实现初步数据压缩；

2）哈夫曼树生成（Tree gen）：对步骤 1 输出的两个字符流进行哈夫曼编码，再对这两个哈夫曼编码进行游程编码后再次进行哈夫曼编码，最后生

成哈夫曼树；

3）哈夫曼编码（Huffman）：根据步骤 2 生成的哈夫曼树输出标头、哈夫曼数据以及编码后的字符长度和距离数据。

压缩计算核心将上述算法步骤封装成 LZ77、Tree gen 和 Huffman 3 个独立逻辑模块，加上 AXI 读、AXI 写和 Header Gen 3 个辅助模块，组装成流水线结构，如图 3 所示。同时，在 FPGA 卡上构建多个这样的压缩计算核心，每个计算核心就是一条可供主机端应用调用的独立流水线，从而提供多路并行处理能力。

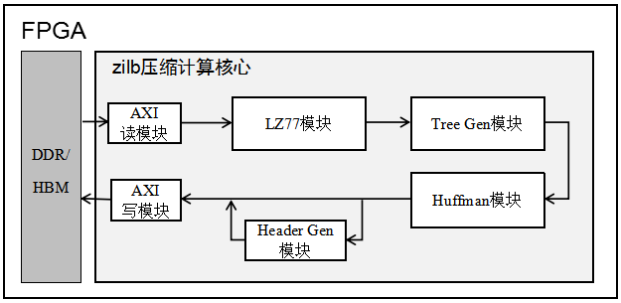


图 3 压缩计算核心内部的多流水线结构

zlib 本身是串行化算法，直接转换为 FPGA 代码，这样的压缩计算核心处理效率低。LZ77、Tree gen 和 Huffman 这 3 个关键逻辑模块都包含复杂的多步骤处理，需要进一步做功能模块级的并行流水线改造。图 4 展示了 LZ77 模块的并行流水线设计。

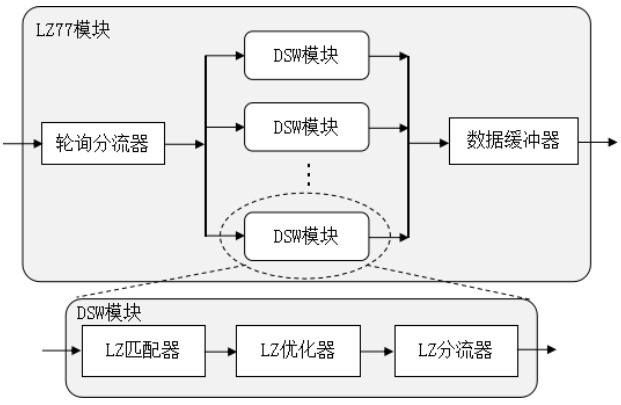


图 4 LZ77 模块的内部结构

LZ77 模块用于实现 LZ77 算法，该算法主要是利用字典和滑动窗口将输入的字符串拆分为相应的字符/长度和距离流，算法时间复杂度高，容易成为 zlib 压缩流水线性能瓶颈。因此我们将 LZ77 模块的字典和滑动窗口（Dictionary and Sliding Window）功能抽取出来，封装为独立处理模块，并将其扩展为 n 个对称的 DSW 模块，如图 4 所示。LZ77 模块读取到输入字符串后，被轮询分流器拆分为 n 个分片，每个 DSW 模块分别读取其中一个分片进行处理，最后数据缓冲器收集汇总所有处理结果。并行化设计后，LZ77 模块的整体处理时延降低为原来的 n 分之一。

在上述流水线设计中，每一个 DSW 模块被拆分成 3 个子模块：LZ 匹配器、LZ 优化器和 LZ 分流器，如图 4 下半部分所示。流水线化的 DSW 模块实现了时间上的并行，一个 DSW 模块在同一时间可处理三份数据，从而减少数据堆积的发生概率。

对于 zlib 压缩计算核心的 Tree gen 和 Huffman 模块也进行了类似的设计，此处不再冗述。

经过以上改造，zlib 压缩算法从串行化处理变成多级流水线并行处理，压缩效率获得提升。

3.2.2 FPGA 数据流通路设计

在 FPGA 异构计算中，FPGA 与 CPU 之间的数据交换是不可避免的，但是，如果一个计算任务的中间处理过程需要二者频繁交互，性能就会受到严重影响。这种交互包括 FPGA 与 NVMe SSD 的数据块传输（情景 1）以及不同 FPGA 计算核心之间的数据交换（情景 2），传统的处理方法是采用内存缓存交换机制，这需要 CPU 参与其中。

对于情景 1，简单地把压缩/解压计算任务卸载到 FPGA 后，虽然释放了部分 CPU 资源，但数据传递仍然需要通过 CPU 中转，不仅数据流路径变长了，而且还得消耗一定的 CPU 资源处理数据拷贝与传递，如图 5（a）所示。为了消除这部分开销，GoldenX 压缩/解压计算核心使用了 PCIe Direct 技术：对于处于同一个 Root Complex 下的 FPGA 和 NVMe SSD 设备，FPGA 通过 PCI Express 接口直接使用 DMA 方式读写 NVMe SSD 盘上的数据，绕过 CPU，如图 5（b）所示，这既缩短了 I/O 路径，又减少了 CPU 中转开销。在新的 I/O 模块实现中，CPU 与 FPGA 职责进行了重新划分：CPU 只负责任务的响应和调度；FPGA 负责计算密集型数据流操作，如压缩/解压、加密/解密等。

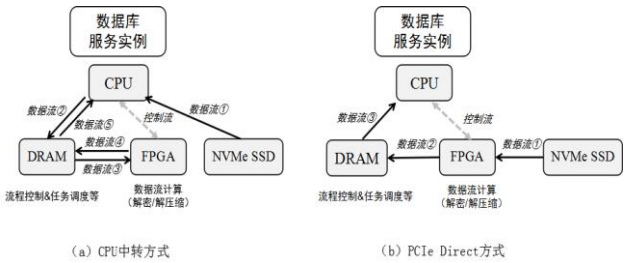


图 5 FPGA 两种 I/O 路径对比

一个计算核心的输出往往是另一个计算核心的

输入，这就涉及到不同计算核心之间数据流通路设计问题。计算核心间的数据流通路可分为两类：同一 FPGA 卡内部的计算核心间数据通信和不同 FPGA 卡上的计算核心间数据通信。为了避免计算核心之间的数据传递通过 CPU 中转，GoldenX 列式存储引擎做了优化处理：

1）对于同一 FPGA 卡，计算核心之间的数据流通路采用编程方式实现直接硬件相连，数据通信不需要通过全局内存（比如 HBM）缓存中转。

2）对于不同 FPGA 卡上的计算核心间通信，使用 Direct Streaming 流技术，将两块 FPGA 的全局内存缓冲映射到相同主机 I/O 内存空间，通过 DMA 硬件来同步两块 FPGA 上的全局内存缓冲对象，从而实现数据交换，这不需要经过 CPU 拷贝。

3.3 向量化执行引擎

随着数据库软硬件技术的发展，经典的 SQL 计算引擎逐渐成为数据库系统的性能瓶颈，尤其是在涉及到大量计算的 OLAP 场景。

经典的 SQL 查询执行引擎主要包括 Expression 级别和 Operator 级别的计算，Expression 层面一般采用 Expression Tree 模型来解释执行，而 Operator 层面则大多采用火山模型。火山模型存在解释开销昂贵、CPU Cache 命中率低等问题，而向量化执行引擎和列式存储的完美结合，可以在很大程度上解决传统模型的弊端。向量化执行其本质是通过一条 CPU 指令实现多次数据操作，通常是对不同的数据执行同样的一个或一批指令。

GoldenX 向量化执行引擎在原有查询树基础上构建向量化处理模型，具有用户无感知、架构灵活等特点。在火山模型中，叶子节点返回一个符合过滤条件的 Tuple（即记录行）给上一层节点处理，每一层节点在处理完该 Tuple 之后继续往上层节点传递该 Tuple 的中间处理结果。总体处理方式是，每次读取一个 Tuple，处理完成后，再读取下一个 Tuple。其缺点是每个 Tuple 被孤立地解析执行，处理效率低下。为了解决这个问题，GoldenX 执行引擎在如下方面做了优化：

1）将标量形式的数据存取方式转换为向量化的存取方式，向量块是数据存取的基本单元，经由执行计划树在不同计算节点间传输，从而实现数据向量化计算；

2)将查询计划的计算密集型算子(比如 AGG、SCAN、JOIN 等）改造成向量化执行方式；通过编写简单而有效的 for 循环处理向量块，构建适合向量数据的运算方法，提高代码到向量指令的转换率，提升 CPU 利用率；

3）对于列式存储引擎，在进行数据扫描时，扫描方式由一次获取一个 Tuple 改为一次获取一组列记录，并转化为一个向量块，通过合理控制向量块大小，保证其接近但不超过 1 个 CPU Cache Line，这样，既增加 CPU Cache 的命中率，又提升 CPU 与 DRAM 数据交换性价比。

通过对数据存取、传输、执行和扫描模块的向量化改造，实现了向量化执行引擎。同时，在代码编译阶段，编译器会自动检测 CPU 支持的 SIMD 指令集，并生成最优指令，进一步提升向量化执行引擎的效率。

如果一个向量块包括 N 条列记录，那么处理流程的函数调用次数就减少了 N 倍，从而提高运行效率。以一条 SQL 语句的执行过程为例进行说明。如图 6 所示，该 SQL 查询计划包含 3 个操作节点：投影操作、聚合运算和扫描过滤，其中投影操作是整个查询计划的根节点，其执行过程说明如下：

1）由根节点逐级向下递归调用下级节点，直至叶子节点，包括步骤①②③；

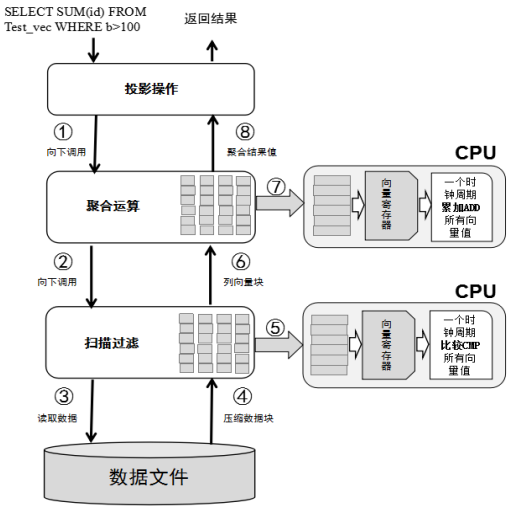


图 6 向量化执行引擎流程图

2）步骤④扫描过滤节点从数据文件获取压缩数据块，解压后组装成一个个列向量块；

3）步骤⑤调用 Intel 的 SIMD 指令集 SSX，执行流程为：首先通过指令集 LOAD 指令将一组向量数据加载到寄存器，然后通过 CMP 比较指令进行数据过滤，最后通过 STORE 指令将计算结果保存到内存中；

4）步骤⑥将符合条件的列向量块返回给聚合运算节点进行下一步处理；

5）步骤⑦调用 SIMD 的 ADD 指令进行中间结果累加计算；

6）聚合运算节点处理完一个向量块后，又会通过步骤②向下调用扫描过滤节点返回下一组符合过滤条件的向量块；

7）直到遍历完所有符合条件的记录，最后聚合运算节点在步骤⑧把聚合结果返回给上层投影操作节点，并将最终结果返回给用户。

上述优化减少了函数调用开销，即使 CPU/GPU 不支持 SIMD，也会自动切换到高效的循环模型来完成向量块中各值的计算。

当前，主流 CPU 厂商都推出支持 SIMD 指令集的产品，比如 Intel Pentium Dual Core E2X 系列 CPU 支持 MMX、SSE，ARM 的 Cortex-A 系列 CPU 支持 NEON。由于各种查询不同运算的差异性和多样性，GoldenX 向量化执行引擎将结合最新计算硬件和动态代码生成技术[37]做进一步的研究探索。

3.4 基于GPU的SQL加速

SQL 查询分析是一种算力密集型操作，不具有 MISD 特征，因此不适合 FPGA 加速。实际上，SQL 查询分析的主要算力消耗还体现在对海量数据进行条件扫描、模糊匹配、排序、分组、聚合等运算，这些操作同时带有通用计算和矩阵运算特征，CPU 并不擅长。一块先进的 GPU 卡可集成成百上千个计算核心（Core），具有超高并行处理能力，更擅长向量和矩阵运算。从峰值性能和通用计算方面看，GPU 远胜于 FPGA。目前业界已经出现了一些比较成熟的 GPU 数据库，比如 Kinetica[38]、SQream[39]、MapD 等等。

基于上述分析，我们最终选择 GPU 协助处理统计分析型 SQL 负载，利用 GPU 的并行计算实现高速数据分析能力。这涉及到 SQL 规划器的改造，主要是通过构建 JIT（just-in-time compilation）框架来替代传统的查询执行器。当 SQL 优化器判定在 GPU 上执行的代价更小时，就调用 GPU Module 模块，通过 JIT 编译机制生成该 SQL 语句对应的 GPU 可执行二进制代码，将过滤、排序、汇聚等计算密集型任务卸载到 GPU 上执行。

将这些算子卸载到 GPU 的目的是降低 SQL 总体执行时长和 CPU 开销，然而 JIT 动态编译生成 GPU 二进制代码本身需要时间代价，也需要消耗一定的 CPU 资源，因此只有那些需要较长时间运行的计算密集型操作才有价值。GoldenX 以 CUDA C++代码的形式实现了适合于 GPU 向量化并行处理的算子，包括表达式估算、元组解析、扫描、连接、分组、排序等。当且仅当优化器评估查询计划的某个分支适合在 GPU 执行且执行代价高于指定配置值时才启用 JIT 动态编译。

GoldenX JIT 框架利用英伟达 GPU 的 CUDA 动态编译机制，让 GPU Module 调用 NVRTC（NVIDIA RunTime Compilation）库将预先开发好的用于实现上述 GPU 运算的 CUDA C++程序源代码，动态编译成 PTX（Parallel Thread eXecution）指令集代码串，替换原始查询计划中的部分 CPU 代码，最后生成优化后的查询计划。执行器在执行过程中通过 GPU 驱动程序将上述 PTX 指令集代码串转换成目标 GPU 硬件机器码，并推送到 GPU 运行。

下面用一个例子来说明 GPU 异构加速过程，如图 7 所示，这条查询语句经过 SQL 解析后，查询优化器判定该执行计划适合在 GPU 上运行，便指示 GPU Module 通过 JIT 编译将任务转化成 GPU 本地可执行代码，然后查询执行器通过异步方式驱动 GPU 完成上述运算。

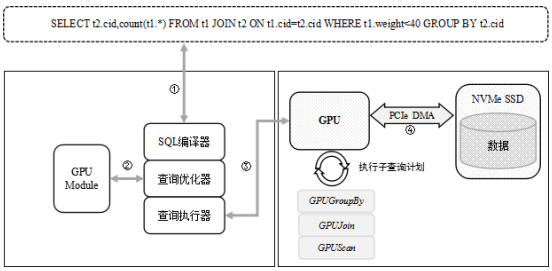


图 7 基于 GPU 的 SQL 加速

该查询语句的具体执行流程如下：

1）SQL 编译器解析步骤①接收到的 SQL 语句，生成初始查询计划；

2）查询优化器判定该执行计划中的 JOIN、SCAN 和 GROUP BY 操作适合在 GPU 上运行，便通过步骤②指示 GPU Module 动态生成 PTX 目标代码；

3）将 PTX 目标代码串替换初始查询计划的原 CPU 执行代码，生成优化后的查询计划；

4）查询执行器执行该查询计划的过程中将该 PTX 目标代码串下推到 GPU 中运行（步骤③）；

5）执行器等待 GPU 执行完毕后获取中间结果集，在 CPU 环境中执行收尾工作。

简单地把该查询计划的 SCAN、JOIN 和 GROUP BY 计算任务下推到 GPU 执行并不能获得

最佳性能，这是因为前后 2 个计算任务之间需要反复在 DRAM 缓存区与 GPU 之间来回传递待处理数据，开销极大。为了解决上述问题，GPU Module 将 SCAN、JOIN 和 GROUP BY 整合在一起，生成一个完整的子查询计划，然后将该子查询计划一次性下推到 GPU 中执行，从而消除了低效的数据来回传递开销。

为了避免数据通过 CPU 加载到内存再传递给 GPU 的额外开销，与 FPGA 类似，步骤④采用了 PCIe Direct（NVMe-to-GPU Direct）技术。GPU Module 在其动态生成的 JIT 二进制代码中应用 NVIDIA GPUDirect RDMA 特性，该特性允许 GPU 与处于同一个 PCIe 总线的第三方设备实现端到端的数据传输，具体实现方法为：让 GPU 通过 PCIe 总线直接访问 NVMe SSD，先将数据加载到 GPU 内嵌的 HBM 内存，再使用 GPU 内部的上千个核并行执行 SCAN、JOIN 和 GROUP BY 等操作。该技术把 GPU 当做 CPU/DRAM 与存储之间的 SQL 预处理器，减少需要 CPU 中转处理的数据量，从而缩短 I/O 路径和减少 CPU 开销。

4 实验结果及分析

本节对 GoldenX 系统异构加速效果进行测试验证，实验内容包括将压缩/解压计算卸载到 FPGA 测试、将扫描/连接/分组运算卸载到 GPU 测试、向量化执行引擎对比测试、列式引擎综合优化后性能对比测试以及和 openguass 的对比测试。所有实验基于同一台服务器环境，具体配置如表 1 所示。

表 1 实验环境配置

名称	规格参数
CPU	Intel® Xeon® Gold 6240L CPU @ 2.60GHz ×2
GPU	Tesla M40 24GB
FPGA	Xilinx Alveo U50
DRAM	32G DDR4-2400 ×2
SSD	1600GB Ultrastar DC SN200 ×4

本节所有实验使用数据库通用测试工具。

4.1 压缩/解压卸载到FPGA

GoldenX 把数据块的压缩/解压、加密/解密计算任务卸载到 FPGA 上执行，为了验证其效果，本小节验证了只读、只写以及读写混合场景下的性能对比。其中，被测试表包含 40 个字段，涵盖整数、字符串、日期 3 种数据类型，单条记录最大长度为 1KB，采用列式存储方式，压缩算法为 zlib，预置 1000 万条记录。测试工具采用 Sysbench，当客户端并发线程数达到 120 个时 TPS 性能接近最高点，采纳此时的测试结果进行对比分析。

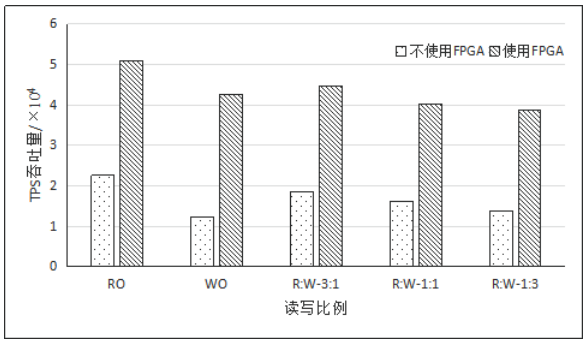
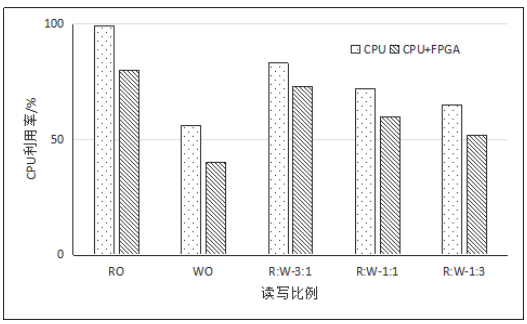
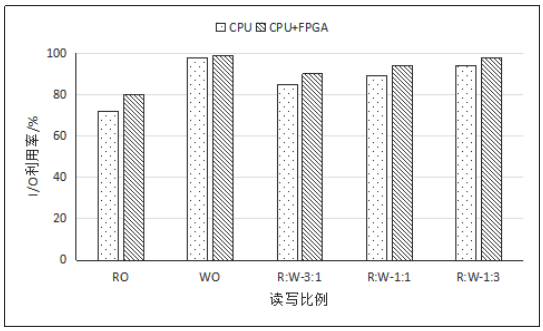


图 8 FPGA 加速压缩/解压

性能对比测试结果如图 8 所示。在只读（RO）工作负载下，使用 FPGA 性能比不使用 FPGA 性能提升了 1.2 倍；在只写（WO）工作负载下，使用 FPGA 性能比不使用 FPGA 性能提升了 2.4 倍；读写混合场景下，工作负载为读写比 3:1（R:W-3:1）时，性能提升了 1.4 倍；工作负载为读写比 1:3（R:W-1:3）时，性能提升了 1.8 倍；工作负载为读写比 1:1（R:W-1:1）时，性能提升了 1.5 倍。实验结果说明应用 FPGA 异构算力加速压缩/解压可以有效提升系统性能。与读密集型相比，当工作负载中有大量写操作时，应用 FPGA 加速的效果更好。这是因为只读场景消耗在 I/O 上的时间占比远低于只写场景，导致 FPGA 异构计算加速效果没那么明显。



（a）CPU 利用率对比



（b）I/O 利用率对比

图 9 系统资源利用率对比

图 9（a）展示了在同等负载情况下计算任务卸载到 FPGA 和只基于 CPU 计算的 CPU 利用率对比。上述 5 种场景下 CPU 利用率分别降低了 19%、16%、10%、12%、13%，说明使用 FPGA 可有效减少 CPU 资源占用，效果符合预期。

图 9（b）展示了上述 5 种场景下 I/O 利用率对比，CPU+FPGA 相比 CPU，分别上升 8%、1%、5%、5%、4%，说明使用 FPGA 可以进一步提升 I/O 利用率，进而提高吞吐率。这是因为 FPGA 通过 DMA 方式读写 NVMe SSD，可消除 CPU 中转开销，缩短 I/O 路径，从而提高 I/O 利用率。需要说明的是，在只写场景下，使用 FPGA 加速仅提升了 1%的 I/O 利用率，其原因是在不使用 FPGA 时其 I/O 开销就已接近 100%，可提升空间有限。

使用 FPGA 加速的效果是和 FPGA 硬件特性以及结合具体工作负载特征的定制设计相关的。今后还会结合 FPGA 硬件技术的发展探索在排序、分组、连接等 SQL 运算方面进行异构加速的可行性。

4.2 扫描/连接/分组运算卸载到GPU

扫描、连接和分组是数据库常见运算，在交易型和分析型场景中都会涉及到，这些运算相当耗费 CPU 资源。在估算查询计划代价时，如果优化器判定这些操作在 GPU 上运行比 CPU 更好，则 GoldenX 会把这些运算任务卸载到 GPU 上。为了验证其效果，本小节设计了对应这 3 种运算的 SQL 语句：

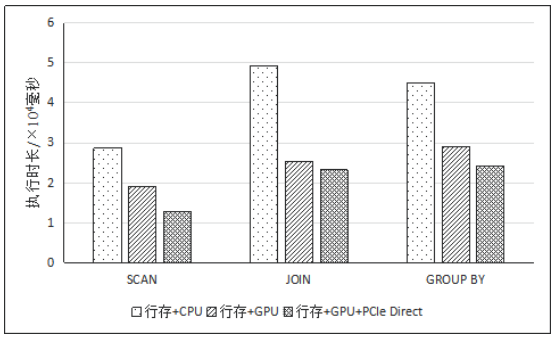
SCAN：SELECT count(*) FROM t2 WHERE c1=？ AND c2=？ AND c3=？

JOIN：SELECT count(t2.id) FROM t1 JOIN t2 ON t1.id=t2.id

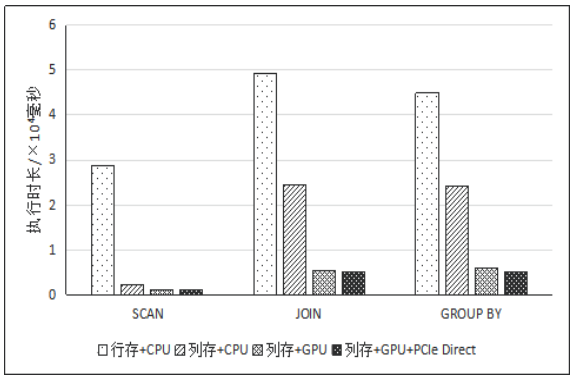
GROUR BY：SELECT t1.c1, count(*) FROM t1 JOIN t2 USING(id) GROUP BY 1

分别基于 CPU、GPU、GPU+PCIe Direct 进行对比测试。上述 3 条 SQL 语句带有明显的统计分析型特征：聚合运算和少数列投影，为了考察列式存储引擎对这类分析型操作的改善程度，我们额外增加了行式存储引擎测试场景，以对比它们的不同性能表现。在测试环境中，被测试表 t1 预置 1000 万条数据，t2 预置 1 亿条数据。每个场景测试 100 组数据，取平均值。

行式存储模型的测试结果如图 10（a）所示，JOIN 的 GPU 卸载效果最为明显，执行时长下降了 48%，SCAN 下降了 33%，GROUP BY 下降了 35%。采用 GPU 叠加 PCIe Direct 技术后，这 3 种场景都进一步降低了时延，其中 SCAN、JOIN、GROUP BY 分别在 GPU 基础上再下降了 33%、8%和 16%。实验表明，即使在面向 OLTP 场景的行式存储模式下，GPU 异构计算也能产生较大收益。



（a）基于行式存储的执行时长



(b)基于列式存储的执行时长

图 10 SCAN/JOIN/GROUP BY 行存/列存对比测试

列式存储模型的测试结果如图 10（b）所示（为了便于对比分析，在第一列放置了行存+CPU 测试结果）：

1）采用列式存储技术后，执行效率改善效果非常显著，与行存+CPU 模式相比，这 3 种场景在列存+CPU 下的处理时长平均下降幅度超过 62%，其中 SCAN 下降了约 92%，JOIN 下降了 50%，GROUP BY 下降了 46%。这是因为列式存储只读取涉及到的列，消除了无关列的 I/O 开销及其列值解析开销。SCAN 处理时长下降更显著的原因是，在

CPU 模式下列式存储相比行式存储的主要优势体现在对记录的扫描与过滤运算，它减少了无关列 I/O，而 JOIN 和 GROUP BY 除了包含记录扫描与过滤运算外，还包含连接或分组运算。

2）与列存+CPU 模式相比，列存模式应用 GPU 加速后，SCAN 下降了 48%，JOIN 下降了 78%，GROUP BY 下降了 75%。这说明在列存模式下 GPU 加速效果比行存模式更为显著，这是因为列式存储模式更有利于快速构建向量化流水线，发挥 GPU 庞大计算矩阵的高并行处理能力。

3）与列存+GPU 模式相比，列存模式叠加应用 GPU+PCIe Direct 技术后，SCAN 进一步下降了 17%，JOIN 下降了 7.8%，GROUP BY 下降了 16.8%。这说明通过 PCIe Direct 技术缩短 I/O 路径对改善性能有正面效果。

综上所述，对于上述统计分析型场景，应用列式存储、GPU 卸载和 PCIe Direct 技术后，GoldenX 的处理性能获得显著提升。

4.3 向量化执行引擎

GoldenX 的向量化执行引擎实现了 AGG、SCAN、JOIN 等计算密集型算子的向量化执行。为验证其效果，本小节基于 TPC-H 基准测试模型的表和数据，设计了 5 个典型 SQL 语句：

SCAN: EXPLAIN ANALYZE SELECT l_orderkey FROM lineitem

AGG1: SELECT sum(l_extendedprice*3 + l_discount +1) FROM lineitem

AGG2: SELECT count(CASE WHEN o_orderpriority ='1-URGENT' OR o_orderpriority ='2-HIGH' THEN 1 ELSE 0 END) FROM orders;

JOIN: SELECT o_custkey FROM orders JOIN customer ON orders.o_orderkey=customer.c_custkey LIMIT 1

AGG+JOIN: SELECT avg(o_custkey*3 - o_shippriority+1) FROM orders JOIN customer ON orders.o_orderkey = customer.c_custkey

预置数据量为 110GB。GoldenX 列存模式开启和关闭向量化执行引擎的对比测试结果如图 11 所示。

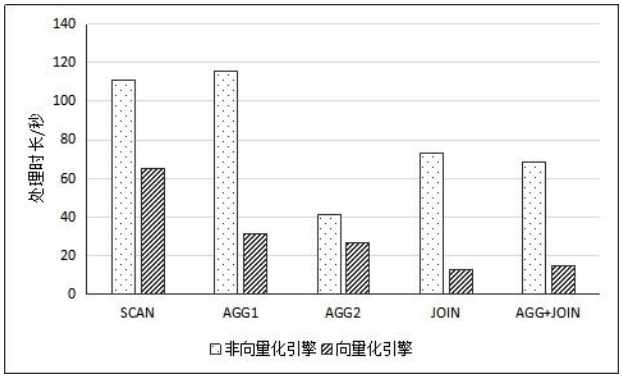


图 11 向量化执行引擎对比测试

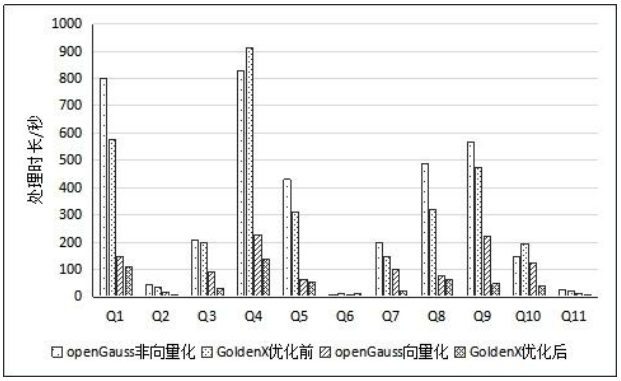
开启向量化引擎后，SCAN、AGG1 和 JOIN 操作的处理时长分别下降了 41%、73%、83%，带 AGG 和 JOIN 的混合 SQL 场景下降了 78%。这是因为通过向量化引擎，SCAN、AGG 和 JOIN 操作减少了执行过程中函数的调用次数，同时 AGG 和 JOIN 操作减少了数据在扫描后由列存数据格式转换为行存格式的操作，并且能够使用 SIMD 特性提升运算效率。AGG2 语句的聚集函数是带有 CASE WHEN 逻辑运算的，执行时需要对每条记录进行多次逻辑判断，从而无法对字段进行批量向量化处理，所以执行时长只下降了 35%，去掉 SCAN 操作向量化后已获得的加速效果，实际性能提升很小；而 AGG1 则是直接对字段进行向量化计算，所以性能提升很明显。实验表明，在列式存储模式下，向量化执行引擎能获得显著性能提升。

4.4 系统实验

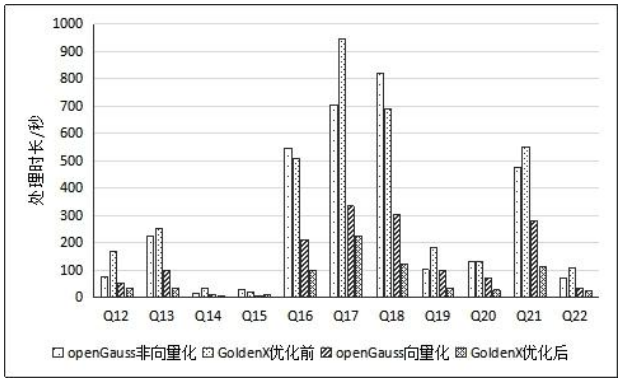
GoldenX 基于软硬件一体化设计，采用 FPGA/GPU 异构计算加速、向量化执行引擎等手段优化系统实现。为了验证 GoldenX 系统优化效果，基于 TPC-H 22 条基准测试模型进行对比测试，同时采用优秀开源项目 openGauss1.0.1 版本作为参照。预置数据量为 110GB。对比测试结果如图 12 所示，其中，openGauss 向量化指开启向量化执行引擎等优化手段，openGauss 非向量化指只采用列式存储，GoldenX 优化前指只采用列式存储引擎，GoldenX 优化后指采用列式存储引擎+上述 3 种三种优化手段。

在 TPC-H 基准测试场景的 22 个查询中，优化前的 GoldenX 列式引擎和未开启向量化的 openGauss 相比，有 14 个测试场景执行时间相当，GoldenX 在 Q6、Q12、Q14、Q19 稍弱，Q1、Q5、Q7、Q8 稍强。可见，两个列式存储引擎总体而言性能相当，不同测试场景的表现各有所长，GoldenX 对含多个表 JOIN 的 FROM 子查询优化效果较好，

而对带 CASE WHEN 的聚集函数等优化效果稍差。优化后的 GoldenX 与开启向量化的 openGauss 相比，执行时长减少了 17%～78%。其中，Q7、Q9、Q10 减少 65%以上，这是因为它们包含大量条件过滤、多表连接和分组聚合等适合 GPU 加速的操作，GoldenX 充分利用到了 GPU 加速；Q18 需要遍历大量数据记录，这使得它显著得益于 FPGA 解压操作加速，执行时长减少 60%；而 Q1、Q6 属于单表查询，使用 GPU/FPGA 加速提升效果有限，执行时长只减少了 20%左右。优化后的 GoldenX 性能比优化前提升了 2.5-10 倍，其中，提升最大的为 Q9，性能提升了 10 倍；提升最小的为 Q6，性能提升了 2.5 倍。Q9 语句为带有分组、排序、聚集、子查询操作并存的查询操作，优化前，JOIN 操作使用 Hash 表分布，数据的分布比较随机，数据经常会被从寄存器中置换到内存中，导致 Hash 数据查找时 CPU Cache 的命中率不高，需要经常访问内存；优化后，GoldenX 将 GROUP BY、JOIN、SCAN 操作卸载到 GPU 上执行，并且使用 PCIe Direct 技术降低数据流的路径，同时向量化执行模型执行逻辑循环较短，并发度高，总的指令等待时间较短，所以性能提升非常明显。Q6 语句的特点是带有聚集操作的单表查询操作，在对 Q6 执行过程进行性能分析时发现，优化前所有 Operator 的查询执行过程所花费的时间不到总时间的一半，虽然通过向量化引擎减少了嵌套调用次数，提高运算过程中 CPU Cache 的命中率，但由于该语句无法通过 GPU 进一步提升性能，所以提升效果比其它场景小。



（a）TPC-H 基准测试 Q1~Q11 处理时长



（b）TPC-H 基准测试 Q12~Q22 处理时长

图 12 列式引擎优化前后 TPC-H 基准测试对比

上述实验结果表明，GoldenX 基于 GPU/FPGA 异构计算和向量化执行引擎的软硬件一体化设计能够有效提升系统性能。

5 总结

新硬件技术的发展为数据库系统带来了新的机遇和挑战，数据库系统软件需要针对新硬件的发展做出适应性调整，以便充分利用新硬件带来的红利来提升数据库系统的性能和应用范围。本文介绍了 GoldenX 应用 FPGA 和 GPU 进行软硬件一体化设计的关键机制和数据库性能优化的实践。使用新型算力来提升数据库系统性能目前在推广应用上仍面临一系列的挑战性，如何进行更好地设计以更有效地整合利用 FPAG、GPU 以及其它异构计算资源来提升数据库系统效能也是关键问题之一。

展望未来，数据库向着新型硬件应用、云原生和 AI 原生等技术方向发展。数据库运行环境将逐步转移到以 FPGA/GPU/NPU/TPU 为代表的异构算力、以 NVM 为代表的新型存储和以 RDMA 为代表的新型网络的软硬件一体化平台上。如何利用这些不断涌现的新型硬件技术来为大规模使用的真实工业系统赋能是一个十分重要的和有挑战的问题，我们将持续探索和不断优化数据库系统对新型硬件的感知、适配和优化技术。此外，随着人工智能技术研究的深入开展，AI-Native[40]也将给数据库系统的优化与实现带来新的思路，AI4DB 也将是为工业数据库系统赋能的一个重要方向。

致 谢 *致谢内容.* 致谢

参 考 文 献

[1] Zhang Yu, Zhang Yansong, Chen Hong, Wang Shan. GPU adaptive hybrid OLAP query processing model. Journal of Software, 2016, 27(5): 1246-1265(in Chinese)
(张宇, 张延松, 陈红, 王珊. 一种适应 GPU 的混合OLAP 查询处理模型. 软件学报, 2016, 27(5): 1246-1265)

[2] Cui Bin, Gao Jun, et al. Progress and trend in novel data management system. Journal of Software, 2019, 30(1): 164-193 (in Chinese)
(崔斌, 高军, 等. 新型数据管理系统研究进展与趋势. 软件学报, 2019, 30(1): 164-193)

[3] Pan Wei, Li Zhanhuai, et al. State-of-the-Art Survey of Transaction Processing in Non-Volatile Memory Environments. Journal of Software, 2017, 28(1): 59-83(in Chinese)
(潘巍, 李战怀, 等. 新型非易失存储环境下事务型数据管理技术研究. 软件学报, 2017, 28(1): 59-83)

[4] Xiao Renzhi, Feng Dan, Hu Yuchong, Zhang Xiaoyi, Cheng Liangfeng. A Survey of Data Consistency Research for Non-Volatile Memory. Journal of Computer Research and Development, 2020, 57(1): 85-101(in Chinese)
(肖仁智, 冯丹, 胡燏翀, 等. 面向非易失内存的数据一致性研究综述. 计算机研究与发展, 2020, 57(1): 85-101)

[5] Wang Chao, Wang Teng, et al. Research Progress on FPGA-based Machine Learning Hardware Acceleration. Chinese Journal of Computers, 2020, 43(6): 1161-1182(in Chinese)
(王超, 王腾, 等. 基于FPGA的机器学习硬件加速研究进展. 计算机学报, 2020, 43(6): 1161-1182)

[6] Gao W, Han L, Zhao RC, et al. Loop vectorization method guided by SIMD parallelism. Journal of Software, 2017, 28(4): 925−939 (in Chinese)
(高伟, 韩林, 赵荣彩, 等. 向量并行度指导的循环SIMD向量化方法. 软件学报, 2017, 28(4): 925-939)

[7] Graefe G, McKenna W J. The volcano optimizer generator: extensibility and efficient search//Proceedings of the Ninth International Conference on Data Engineering. NW Washington, DC, USA, 1993: 209-218

[8] Polychroniou O, Ross K A. High throughput heavy hitter aggregation for modern SIMD processors//Proceedings of the data management on new hardware. New York, NY, USA, 2013: 6

[9] Held G, Stonebraker M, Wong E. INGRES: A relational data base management system. Proc Afzps Ncc, 1975, 17(1): 23-31

[10] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data. ACM transactions on computer systems, 2008, 26(2): 1-26

[11] Decandia G, Hastorun D, Jampani M, et al. Dynamo: Amazon's highly available key-value store//Proceedings of the acm Symp on Operating Systems Principles. Washington, USA, 2007: 205-220

[12] Lakshman A , Malik P. Cassandra - A Decentralized Structured Storage System. Operating systems review, 2010, 44(2): 35-40

[13] Arulraj J, Pavlo A, Dulloor S R. Let's talk about storage & recovery methods for non-volatile memory database systems//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. Melbourne, Victoria, Australia, 2015：707-722

[14] Arulraj J, Pavlo A. How to build a non-volatile memory database management system// Proceedings of the 2017 ACM International Conference on Management of Data. Chicago Illinois, USA, 2017: 1753-1758

[15] Stonebraker M, Weisberg A. The voltDB main memory DBMs. IEEE Data Engineering Bulletin, 2013, 36(2): 21-27

[16] Franz Färber, Cha S K , Jürgen Primsch, et al. SAP HANA database: Data management for modern business applications. Acm Sigmod Record, 2011, 40(4): 45-51

[17] Nguyen X T, Nguyen H T, Hoang T T, et al. An efficient FPGA-based database processor for fast database analytics// Proceedings of the IEEE International Symp on Circuits and Systems (ISCAS). Montreal, Quebec, Canada, 2016: 1758-1761

[18] L. Woods, Z. Istvón, G. Alonso. Ibex: An intelligent storage engine with support for advanced SQL offloading.Proc VLDB Endow, 2014, 7(11): 963－974

[19] Yong K , Karuppiah E K , See S . Galactica : A GPU parallelized database accelerator//Proceedings of the third Ase Int Conf on Big Data Science & Computing. Beijing, China, 2014: 10

[20] E. A. Sitaridi， K. A. Ross. Optimizing select conditions on GPUs// Proceedings of the data management on new hardware. New York, USA, 2013: 4

[21] R. J. Halstead, I. Absalyamov, W. A. Najjar, V. J.Tsotras. FPGA-based multithreading for in-memory hash joins//Proceesings of the conference on innovative data systems research, CA, USA, 2015

[22] M. Najafi, M. Sadoghi, H.-A. Jacobsen. Flexible query processor on FPGAs//Proceedings of the very large data bases.Riva del Garda, Trento, Ital, 2013, 6(12): 1310-1313

[23] Zhang T, Wang J, Cheng X , et al. FPGA-Accelerated compactions for lSM-based key-value store FPGA-accelerated compactions for LSM-based key-value store//Proceedings of the file and storage technologies. Santa Clara, CA, USA, 2020: 225-237

[24] Huang G, Cheng X, Wang J, et al. X-Engine:An optimized storage engine for large-scale e-commerce transaction processing//Proceedings of the international conference on management of data. Amsterdam, The Netherlands, 2019: 651-665

[25] Owaida M , Sidler D , Kara K , et al. Centaur: A Framework for Hybrid CPU-FPGA Databases//2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines, Napa, CA, USA, 2017

[26] J.Doetal. Query Processing on Smart SSDs: Opportunities and Challenges//Acm Sigmod International Conference on Management of Data, NewYork, USA, 2013

[27] Salami B , Malazgirt G A , Arcas-Abella O , et al. AxleDB: A novel programmable query processing platform on FPGA. Microprocessors & Microsystems, 2017, 51(jun.): 142-164

[28] T.Kersten, V.Leis, A.Kemper, et al. Everything you always wanted to know about compiled and vectorized queries but were afraid to

ask. Proceedings of the VLDB Endowment, 2018, 11(13): 2209-2222

[29]Menon, Prashanth, Todd C. Mowry, Andrew Pavlo. Relaxed operator fusion for in-memory databases: Making compilation, vectorization, and prefetching work together at last. Proceedings of the VLDB Endowment, 2017, 11(1): 1-13

[30] Zhang Yansong, Zhang Yu, Wang Shan. Vector Referencing Oriented Platform-Oblivious In-Memory Join Optimization Technique. Journal of Software, 2018, 29(3): 883−895 (in Chinese)

(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.)

[31] Root C, Mostak T. MapD: a GPU-powered big data analytics and visualization platform//Proceedings of the international conference on computer graphics and interactive techniques. Anaheim California, USA, 2016: 73

[32] Zhang Yansong, Zhang Yu, Wang Shan. A Novel In-Memory OLAP Star Join Acceleration Technique with Vector Index. Chinese Journal of Computers, 2019, 42(8): 1686-1703(in Chinese)

(张延松，张宇，王珊. 一种基于向量索引的内存OLAP星型连接加速新技术. 计算机学报， 2019, 42(8): 1686-1703)

[33] Bakkum P , Skadron K . Accelerating SQL database operations on a GPU with CUDA// Workshop on General Purpose Processing on Graphics Processing Units, Pittsburgh, Pennsylvania, USA, 2010

[34] Yuan Y , Lee R , Zhang X . The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. Proceedings of the Vldb Endowment, 2014, 6(10): 817-828

[35] Luo Xinyuan, Chen Gang, Wu Sai. A GPU-Accelerated Highly Compact and Encoding Based Database System. Journal of Computer Research and Development, 2015, 52(002): 362-376(in Chinese)

(骆歆远，陈刚，伍赛. 基于GPU加速的超精简型编码数据库系统. 计算机研究与发展, 2015, 52(002): 362-376)

[36] Cao W, Liu Y, Cheng Z, et al. POLARDB meets computational storage:Efficiently support analytical workloads in cloud-native relational database//Proceedings of the file and storage technologies. Santa Clara, CA, USA, 2020: 29-41

[37] Lang H, Muhlbauer T, Funke F, et al. Data blocks: Hybrid OLTP and OLAP on compressed storage using both vectorization and compilation//Proceedings of the international conference on management of data. New York, NY, USA, 2016: 311-326

[38]Kinetica.Kinetica,https://www.kinetica.com/products/gpu-accelerated-database/ 2019,8,3

[39] Sitaridi E A, Ross K A. GPU-accelerated string matching for database applications. The VLDB Journal, 2016, 25(5): 719-740

[40] Li GL, Zhou XH. XuanYuan: An AI-native Database Systems. Journal of Software, 2020, 31(3): 831-844(in Chinese)

(李国良， 周煊赫.轩辕:AI原生数据库系统. 软件学报, 2020, 31(3): 831-844)