计算机研究与发展 DOI:10.7544/issn1000-1239.2018.20160845
Journal of Computer Research and Development 55(3):651-661,2018

基于 GPU 的 RDF 类型同构并行算法

冯佳颖[1,3] 张小旺[1,3] 冯志勇[2,3]
[1](天津大学计算机科学与技术学院 天津 300350)
[2](天津大学软件学院 天津 300350)
[3](天津市认知计算与应用重点实验室 天津 300350)
(fengjiaying@tju.edu.cn)

Parallel Algorithms for RDF Type-Isomorphism on GPU

Feng Jiaying[1,3], Zhang Xiaowang[1,3], and Feng Zhiyong[2,3]
[1](*School of Computer Science and Technology, Tianjin University, Tianjin* 300350)
[2](*School of Computer Software, Tianjin University, Tianjin* 300350)
[3](*Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin* 300350)

Abstract Resource description framework (RDF), officially recommended by the World Wide Web Consortium (W3C), describes resources and the relationships of them on the Web. With the volume of RDF data rapidly increasing, a high performance method is necessary to efficiently process SPAQRL (simple protocol and RDF query language) query over RDF data, which can be reduced to the classical problem—subgraph isomorphism. As an important class of subgraph isomorphism, type-isomorphism helps many interesting queries over RDF data to get high performance such as star or linear query structures. However, many existing approaches, which are proposed to solve type-isomorphism, mostly depend on calculative capabilities of CPU. In recent years, graphic processing units (GPU) has been adopted to accelerate graph data processing widely in several works, which have better computational performance, superior scalability, and more reasonable prices. Considering the limited calculative capabilities of CPU in handling large-scale RDF data, we propose an algorithm that processes type-isomorphism problem on parallel GPU architecture over RDF datasets. In this paper, we implement the algorithm and evaluate it in the benchmark datasets—lehigh university benchmark (LUBM) through a mass of experiments. The experimental results show that our algorithm outperforms significantly than the CPU-based algorithms.

Key words resource description framework; SPARQL query processing; subgraph isomorphism; type-isomorphism; graphic processing units

摘 要 资源描述框架(resource description framework, RDF)作为 W3C(World Wide Web Consortium)组织提出的语义网数据规范,描述了资源及其之间的关系.随着 RDF 数据规模不断增加,高效地检索 RDF 数据成为当前面临的重大挑战.在 RDF 数据上的查询响应问题可以被简化为子图同构问题.作为

收稿日期:2016-11-15;**修回日期**:2017-08-09
基金项目:国家重点研发计划项目(2016YFB1000603);国家自然科学基金项目(61672377);天津市科技支撑重点项目(16YFZCGX00210)
This work was supported by the National Key Research and Development Program of China (2016YFB1000603), the National Natural Science Foundation of China (61672377), and the Key Technology Research and Development Program of Tianjin (16YFZCGX00210).
通信作者:张小旺(xiaowangzhang@tju.edu.cn)

子图同构的重要部分,类型同构(type-isomorphism)在处理部分 RDF 查询,如星状查询和链状查询等,具有较高的性能.目前,现有解决类型同构的方法匹配效率均依赖于 CPU 的计算能力.近年来,图像处理单元(graphic processing units, GPU)的发展提高了图数据处理的性能.与 CPU 相比,GPU 多处理器具有高并发、易扩展以及价格成本低等优势.由于 CPU 处理大规模 RDF 数据的计算能力有限,提出一种基于 GPU 的 RDF 类型同构算法,使类型同构问题在 GPU 架构上通过并行的方式解决.最后,实现了基于 GPU 的 RDF 类型同构算法,并在基准数据集 LUBM 上对该算法进行性能测试,实验结果表明:该算法显著优于基于 CPU 架构的算法.

关键词 资源描述框架;SPARQL 查询处理;子图同构;类型同构;图像处理单元

中图法分类号 TP392

语义网(semantic Web)是一种新型的智能网络,它旨在解决当前 Web 上信息爆炸导致人们获取目标信息十分困难的问题.资源描述框架(resource description framework, RDF)和 SPARQL 查询语言是 W3C(World Wide Web Consortium)组织推荐的用来描述语义网中的数据资源及其之间关系的语言规范和数据查询语言.随着 RDF 数据量的不断增大,如何在大规模 RDF 数据集上进行高效的查询检索是当今面对的一个重大挑战.

传统 RDF 数据集的查询方法使用 CPU 作为处理器,诸如 Jena[1],RDF-3X[2],gStore[3] 等集中式查询引擎系统,它们已将 CPU 的计算能力发挥到了极致.近年来,图形处理单元(graph processing unit, GPU)计算性能不断增强.由于 GPU 具有浮点计算能力强、带宽高、性价比高、能耗低等优势,可以极大地补充 CPU 的计算能力.利用 GPU 的并行处理能力实现算法和操作,可以大大降低 CPU 的负载.

在 RDF 数据集上的 SPARQL(Simple Protocol and RDF Query Language)查询过程就是一个子图同构的过程.子图同构问题是图处理问题中一个最基本的研究内容,属于 NP 完全问题.类型同构是一种不精确的子图同构.通过顶点和边的标签约束,将部分查询转化为类型同构问题.并结合 GPU 数据处理方面高效的的计算性能,本文提出了使用并行硬件 GPU 实现对 RDF 类型同构查询的算法,从而提高 RDF 数据集的查询性能.

本文主要贡献如下:结合 RDF 数据的特征和 GPU 极高的计算能力,提出一种基于 GPU 的匹配算法,使 GPU 多处理器调度更简便;在 GPU 的硬件加速下,对本算法进行大量的性能实验.实验结果表明:基于 GPU 的类型匹配算法性能对于现有方法有了显著的提高.

1 背景知识

1.1 RDF 和 SPARQL

资源描述框架 RDF 是 W3C 组织提出的语义 Web 标准数据模型.它是用于形式化描述 Web 资源的元数据.RDF 通过语法符号和序列化数据格式的方法来描述概念并对信息进行建模.RDF 三元组 (S,P,O) 是 RDF 数据集的基本单元,一个 RDF 数据集中包含了许多 RDF 三元组,这些三元组表示了资源和资源之间的联系,其中 S 代表主语(subject),是被描述的资源,P 代表谓语(predicate),表示资源的一个属性或关系,O 代表宾语(object),表示属性值.

由许多 RDF 三元组组成的数据集将语义 Web 的信息表示成为一个带标注的有向图,称为 RDF 图.

定义 1. RDF 图.一个 RDF 图可以表示成为给定一个带标签的有向图 $G=(V,E,\mu,\sigma)$.V 表示顶点的有限集,E 表示边的有限集.$\mu:V\rightarrow L_V$ 表示顶点到顶点标签类型的映射函数.$\sigma:E\rightarrow L_E$ 表示边到边标签类型的映射函数.L_V 和 L_E 是按标签排列的离散符号集合,表示了联系 2 个带标签的顶点或边的标签值向量.

如图 1 展示了一个 RDF 图,其中,椭圆节点表示资源,有向边表示属性,矩形节点表示字符值.例如,

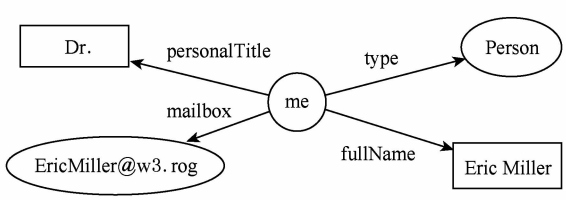


Fig. 1 RDF graph

图 1 RDF 图

对于资源“me”,其“fullName”属性的值是一个字符值“Enic Miller”.

SPARQL 查询语言是 W3C 组织推荐的 RDF 标准查询语言,由一组三元组模式(triple pattern)组成.大多数 SPARQL 查询都是由若干个基本图模式(basic graph pattern, BGP)组成.SPARQL 查询即为图模式匹配,其查询的过程为找到与每个三元组模式匹配的三元组,再连接(join)具有共享变量的三元组模式的局部结果.结合图 1,如果我们想获得某个具有属性“type”,同时具有属性“fullName”其值为“Eric Miller”的资源时,我们可以通过如下的 SPARQL 查询获得,其对应的查询图如图 2 所示.

```
SELECT ?x WHERE{
        ?x   type   ?y.
        ?x   fullName   "Eric Miller". }
```

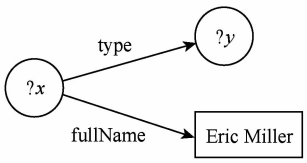


Fig. 2 SPARQL query graph

图 2 SPARQL 查询图

本工作通过研究 SPARQL 语法细节和语义的形式化定义[4]并使用 GPU 上的类型同构算法来提高特定基本图模式中查询处理的性能.

1.2 RDF 中的子图同构与类型同构

依据 RDF 图是带标签的有向图的特征,本文的研究重点仅关注带标签的有向图.在已知的 RDF 数据中进行 SPARQL 查询的过程也就是一个模式图(即图 2 所示查询图)与 RDF 图的子图同构匹配过程.

定义 2. 子图同构.给定一个 RDF 图 G 和一个模式图 P,P 包含 n 个顶点($v_0,v_1,v_2,\cdots,v_{n-1}$).如果 G 中包含 n 个顶点($u_0,u_1,u_2,\cdots,u_{n-1}$),与 P 之间存在一个单射函数 $f:P\to G$,使 G 的顶点和边与 P 的顶点和边一一匹配,则认为模式图 P 子图同构于有向图 G.

寻找子图同构就是将模式图 G_p 嵌入 RDF 图 G 的过程.当子图的所有顶点、边都与模式图的排列顺序相同,并且对应的顶点和边符合各自的标签时,匹配过程结束.

定义 3. k 边游走.假设一个图中的非空有限序列 $W=(v_0,e_0,v_1,e_1,v_2,\cdots,e_{k-1},v_k)$,其中边 e_i 连接顶点 v_i 和 v_{i+1}.这个序列的长度为 $2k+1$,被称为 k 边游走(k-edge walk).

定义 4. 类型同构.给定一个 RDF 图 $G=(V_G,E_G,\mu,\sigma)$,和相关联的模式图 $P=(V_P,E_P,\mu,\sigma)$,它们之间存在标签函数的映射关系,$\mu:V\to L_V$ 和 $\sigma:E\to L_E$ 分别定义了顶点和边与标签之间的映射关系,其中 L_V 和 L_E 为标签集合,即 $L_{V_P}\subset L_{V_G}$ 和 $L_{E_P}\subset L_{E_G}$.定义 P 的 k 边游走为

$$W_P=(v_0^P,e_0^P,v_1^P,e_1^P,v_2^P,\cdots,e_{k-1}^P,v_k^P),$$

其中,$v_i^P\in V_P$,$e_i^P\in E_P$.那么对于 G 的 k 边游走:

$$W_G=(v_0^G,e_0^G,v_1^G,e_1^G,v_2^G,\cdots,e_{k-1}^G,v_k^G).$$

定义映射关系 C_μ 和 C_σ 以判断顶点和边是否匹配,其中 $C_\mu(v_i,v_j):L_V\times L_V\to\{0,1\}$,$C_\sigma(e_i,e_j):L_E\times L_E\to\{0,1\}$.模式图 P 与 RDF 图 G 是类型同构,当且仅当 $C_\mu(v_i^P,v_i^G)=1$ 且 $C_\sigma(e_j^P,e_j^G)=1$ 时成立,其中 $i=0,1,\cdots,k$,$j=i,\cdots,k-1$.

类型同构通过 k 边游走算法,计算出 RDF 有向图中与模式图的类型相匹配的部分.如图 3(b)和图 4(b)中虚线框内 G 的子图均为满足 P 的类型同构.不同之处在于图 3(b)是对 P 的顶点不加约束的类型同构,图 4(b)是对 P 的顶点增加约束的类型同构,说明类型同构只是匹配满足映射的顶点与边,并不要求同构.

类型同构是一种不精确的子图匹配,不同于子图同构问题的是类型同构不需要找到同构结构.如图 3(a)和图 4(a)所示的 G 对应了相同的类型同构,但对应了不同的子图同构,如图 3(c)和图 4(c)所示.通过对类型同构的 k 边游走序列增加标签的限制,从而得到更精确的结果,如图 3 所示.IRSMG[5]中

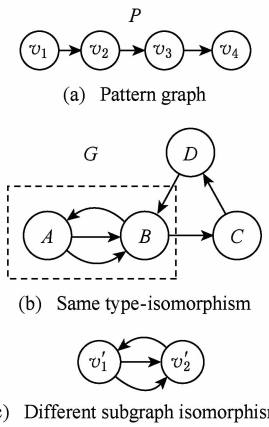


Fig. 3 Without constrained type-isomorphism of P vertexes

图 3 P 的顶点不加约束的类型同构

提出子图同构问题的解可以简化为约束的类型同构的解. 因此,本文将在 RDF 图与基本图模式之间的匹配问题,转化为 RDF 图与相应的关联模式图的附加约束的类型同构匹配问题.

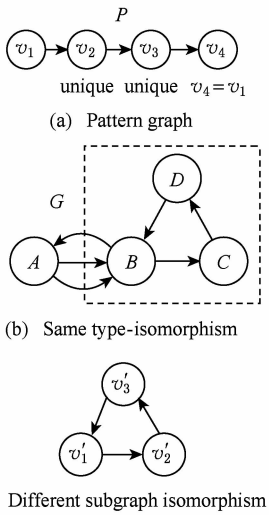


Fig. 4 Constrained type-isomorphism of P vertexes

图 4 P 的顶点增加约束的类型同构

1.3 GPU 架构和 CUDA 编程模型

图 5 所示为 CPU 与 GPU 架构示意图. 与 CPU 相比,GPU 增多了算术逻辑单元(arithmetic and logic unit, ALU),使其更适用于规则结构、算法单一的数据处理. 基于 GPU 的特征,本文设计了 GPU 多处理器的调度模型,以提高其计算性能.

GPU 由许多单指令多数据流(single instruction multiple data, SIMD)处理器组成,支持数千个并发线程,如图 6 所示为拥有 GPU 设备,Host 为 CPU 控制端,Device 为 GPU 设备端. Host 端程序的执行按照串行代码(serial code)运行,图 6 左端箭头指

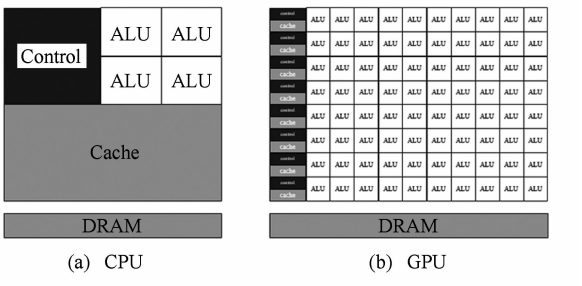


Fig. 5 Architecture of CPU and GPU

图 5 CPU 与 GPU 架构示意图

示了程序串行执行的时间顺序. 当内核(kernel)程序运行时,调用 GPU 进行计算,数以千计的 GPU 将同时进行计算,其组织形式如图 6 所示. GPU 线程具有低上下文切换和低创建线程时间的特点. GPU 的线程被组织为线程组(thread group). 同一个线程组中的所有线程共享计算资源,例如寄存器、缓存等. 线程组被分为多个调度单元方便在多处理器上动态调度. GPU 内存具有高带宽和高访问延迟等特点,这些优势使 GPU 更适合于大规模 RDF 数据的并行计算.

CUDA(compute unified device architecture)是显卡厂商 NVIDIA 推出的一款用于 GPU 并行编程的编程模型 API(application programming interface). 在 CUDA 中,并行工作负载以计算内核(kernel)的形式表示并提交到设备上执行. 内核的构造方式提供了细粒度并行性. 通常在主机的 CPU 上控制内核的提交和执行. 每个线程被分配唯一标识符,并为它们安排调度顺序. 线程标识符常用于加载和存储的内存偏移量的计算. 主进程和计算单元之间的数据传输通过全局存储器来完成,并且所有线程都有

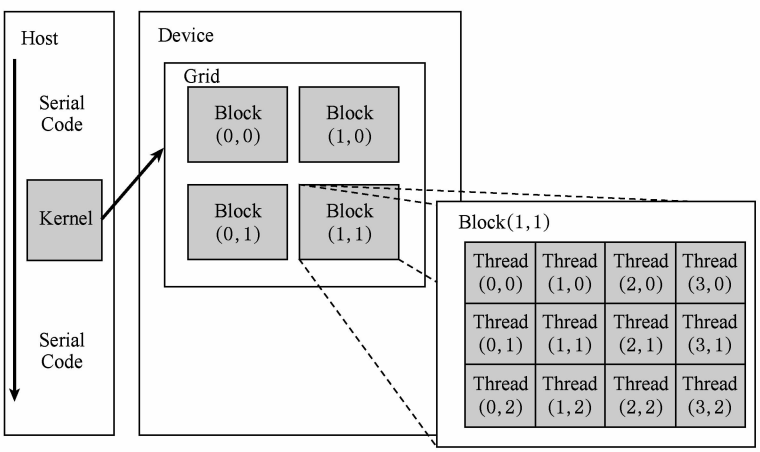


Fig. 6 Concurrent threads of GPU

图 6 GPU 的并发线程

全局存储器访问权限.但是 CUDA 编程模型不提供计算单元的内存分配方法.这意味着在执行内核之前,主进程需要分配好所需的输出缓冲区.

结合 RDF 数据和 GPU 计算能力的特征,本文主要涉及的是为内核计算的编码和执行设计一个完善的 GPU 调度策略,并提出一种基于 GPU 的类型算法来提高部分基本图模式的查询效率.

2 相关工作

子图同构是精确的图匹配,是图数据研究中的一个 NP-完全(NP-complete)问题.近年来,子图同构问题得到了广泛的关注,很多研究人员尝试了许多方法来解决通用图中的子图同构问题.1976 年,Ullmann[6]第 1 次提出了实用的子图同构的搜索算法.该算法基于回溯的思想,当部分解决方案可行时对其进行迭代直到找出完整的解决方案;如果发现部分解决方案不可行,则及时终止.目前,已有一些方法对 Ullmann 算法进行改进,例如 VF2[7],QuickSI[8],GraphQL[9],GADDI[10],SPath[11] 等.这些算法利用不同的连接顺序、剪枝规则或辅助信息对候选集进行及时有效的剪枝筛选,从而减少连接操作的运算量,提高算法性能.但是,由于时间复杂度较高,这些算法不适用于大规模图数据的应用场景.

为了加速在大规模图或者多数量图上的子图同构操作,STwig 算法[12]将查询图分解成一系列结构简单的基本单元,这些基本单元能够非常容易地在数据图中完成匹配,但是将产生的局部结果组合在一起则需要高效的连接策略.吕雪栋等人[13]设计的 FFD-Inde 是建立一种新颖的索引算法来加速星形图的查询处理.在星形查询中,这种方法在数据量巨大的情况下显著地提高了查询效率,但是星形查询只是查询图的一部分,无法推广到查询图的一般情况.基于类型同构的非精确子图同构是由精确子图同构问题的泛化引入的.这种方法比较新颖并且可以扩展到子图同构问题.Berry 等人[14]提出了基于类型同构的不精确的子图同构问题.类型同构不需要像子图同构一样结构完全相同.根据 RDF 数据的特征,考虑利用顶点和边的标签的类型信息,将子图同构问题转化成为带约束的不精确的类型同构问题.

近些年,图形处理器 GPU 在并行处理方面取得了十分显著的成绩.由于计算能力强和便于编程,GPU 已经逐渐发展成为用于通用计算中常用的协处理器[15].基于 GPU 的大规模并行处理框架已经成功应用于处理大规模图上的基本图操作,包括广度优先搜索[16]、最短路径[17] 和最小生成树[18].在 RDF 数据管理领域,Pan 等人[19]率先在 RDFS 中明确提出计算量的工作负载的概念从而允许利用 GPU 的细粒度并行,并以此获得性能的提高.该工作仅关注了 RDFS 推理的并行,没有关注在底层数据查询查询处理的优化.Choksuchat 等人[20-22]提出一种基于 Jena 的 RDF 处理系统,存储结构使用基于 Cassandra 的 Key-value 结构和基于 HDT 的二进制三元组模式(binary triple pattern),并实现了基于 JCuda 语义查询.Chantrapornchai 等人[23]提出基于 Bitstructure 的修剪算法处理查询并使用概要信息以减少数据读取量的方法,以保证可扩展性和优化内存使用状况.TripleID[24]是用于 RDF 查询和推导处理的并行框架,它提供了一种新颖的压缩 RDF 数据的文件格式,以通过 GPU 并行加速数据处理.Fu 等人在 2014 年提出的 MapGraph[25]是基于对 PowerGraph's Gather-Apply-Scatter(GAS)模型的修改而实现的.他们使用 MapGraph 实现了 4 种常见的图形分析操作:宽度优先搜索(BFS),单源最短路径(SSSP),连接组件(CC)和 PageRank(PR).2015 年,Yang 等人[26]提出了在普通图上进行并行处理子图同构的 GPU 算法,但普通图的规模有很大的局限性[27],例如语义信息含量少、节点数量少等[28],所以此方法并不适用于大规模的 RDF 数据[29].同样,对于处理大规模的 RDF 数据的类型同构问题也没有高效的基于 GPU 的求解方式.

这些方法的共同之处在于将 RDF 三元组进行编码压缩,并对其建立索引,以加速存储查询的过程.本文依据 RDF 图数据的特性,提出一种基于类型同构的子图匹配算法,以使 GPU 并行计算应用于在 RDF 图处理,并提高子图匹配的性能.

3 RDF 类型同构算法

3.1 字典构建

本文选择以串行的方式解析文件和预处理数据.依据 RDF 数据的特性,每个三元组元素都由主语、谓语和宾语 3 部分组成,并且每个元素都是字符串(string)值,例如 URI(uniform resource identifier)或字符值.程序在内存中构造字典将所有字符串值转化为整数值.这样保证了每个值都是固定长度,方

便后续程序对数据进行处理.同时,由于缓冲区需要在数据传输前进行分配,固定长度可以降低 GPU 调用的的复杂度,提高算法的性能.

字典是在 CPU 中动态构建的.本算法使用散列映射(Hash map)建造字典.散列映射是一个关联容器,用于关联 Key 类型的对象与 Data 类型的对象.散列映射中可以高速查找 Key 值元素,本算法利用这一优势来查找字典中的无序三元组.字典可以简便地加载到内存或存储在磁盘文件中.在解析三元组时,主语和宾语均描述的是资源,算法对其进行相同的处理.谓词作为资源的连接关系需要单独处理,并将三元组扩展成了五元组 $G=(S,P,O,\mu,\sigma)$. μ 映射表示标签值与每个顶点(即主语和宾语)的关系,σ 映射表示标签值与每个边(即谓语)的关系.

3.2 基于 GPU 的类型同构算法

基本图模式查询是 SPARQL 查询的基本形式和主要子集[19].SPARQL 中的其他图模式,如联合模式(union graph pattern)、可选模式(optional graph pattern)等,都可以通过额外的代数运算转换为基本图模式.

由 2.2 节中描述的概念,类型同构根据已形成的游走路径 W_P 来访问 RDF 中顶点和边.理想情况下,查询包含用作最小游走长度的欧拉路径.如果查询是可以由欧拉路径构造的基本图模式查询,并且在顶点和边添加标签约束,那么类型同构的解就是基本图模式查询的解.但是在实际情况中,基本图模式查询都是有向无环图.因此,查询可能无法追踪任何的满足欧拉路径的游走路径而导致失败.

基本图模式查询可以分解为 2 个基本操作:映射(mapping)和连接(join).假设 p 是模式图 P 中的一部分,t_p 是基本图模式查询中的 1 个三元组模式,E_{t_p} 是 RDF 图中所有满足 t_p 匹配条件的三元组候选集,这个筛选过程称为映射.2.2 节定义了类型同构的游走路径以及它的长度 k.本文所提的 RDF 类型同构算法将映射的集合分成 k 个无关部分同时进行,以提高算法的性能.注意 $E^i_{t_p}$ 和 $E^{i+1}_{t_p}$ 之间存在 1 个相同的点关联.也就是说,2 个三元组可以形成 1 个游走.假如所有 E_{t_p} 可以形成 1 个 k 步游走的通路,那么 RDF 数据图中所有基本图模式查询的类型同构都能被找到.

1) 基于 GPU 的并行映射算法.第 2 节中讨论过 GPU 和 CUDA 编程模型的特性.由于 GPU 在代码的执行期间不支持在设备上动态分配内存.所以需要利用设计好的数组结构在执行 GPU 程序之前为输入数据和输出结果分配空间.当 GPU 执行映射操作时,如果三元组满足映射条件,则将候选三元组 E_{t_p} 放置在输出缓冲器中;如果不满足映射条件,则需要为输出缓冲器设置额外的空间以存储中间结果.因此,将所有数组结构中的元素初始设置为 0,在缓冲区根据主语 ID 排序,以减少三元组主语 ID 的重复.

2) 基于 GPU 的连接算法.连接是将多个已匹配的局部匹配图合并为完整匹配图的操作.如果基本图模式包含共享变量,那么就将映射的输出直接传递到连接中.如果基本图模式没有共享变量,则无法进行接下来的连接操作.那么对于类型同构来说,如果 2 个匹配的三元组有 1 个共享变量,那么它们可以被合并成 1 个通路.

为了充分利用 GPU 并行的优点,连接过程包含 3 个阶段:①并行映射;②按 ID 排序;③去重.在并行映射阶段中,本算法根据共享变量在三元组中的位置设置标志以减少去重阶段的冗余操作,而 GPU 的单指令多数据的体系架构特点也有助于加速笛卡儿乘积的平行化.

本文提出的算法请见算法 1,是以初始化游走列表 InitWalks 开始,以选择 RDF 数据图中第 1 个可以与 W_P 模式匹配的分段.从行②开始的循环,每次迭代都会通过匹配分段,进一步扩大候选集的游走长度的范围.映射操作输出分段并以顶点 ID 作为关键字(Key).行③说明了匹配的实现,并且本算法使用双调排序法(bitonic sort)对映射操作的结果进行并行排序.其作用是通过共享变量顶点的关键字对游走的候选集进行排序.算法 1 的其余部分是通过对计算笛卡儿积而实现去重操作.算法 1 通过 1 次迭代 W_P 的一个分段以构建与该分段匹配的所有游走的候选集.显然,生成结果中完整长度的有向路径的集合即是所有类型同构游走的结果集合,也就是在 RDF 数据图中完成了类型同构的匹配过程.

算法 1. 基于 GPU 的类型同构算法.

输入:RDF 数据图中所有三元组模式的集合 S、将图分段成 k 个有序列表 T_P;

输出:在 RDF 中完成匹配的集合 R.

① InitWalks:接收 S 和 T_P 从 S 中读取分段 s;

② if s 能够匹配 W_P 中的第 1 个分段(S_1,P_1,O_1)

③ 将 S 添加入 candidate walk 为 1 的候选集中;

④ ExtendWalks:

⑤ for 每一个 Key 值小于 k 的记录 do

⑥ 从 S 中读取另一个分段 s. 如果它能够匹配 W_P 中的第 1 个分段 (S_t, P_t, O_t),将 s 添加入游走长度为 t 的候选集中,并从最近访问的候选集中读取 candidate walk;

⑦ end for

⑧ 使用双调排序发对 candidate walk 进行并行排序;

⑨ for 每个定点 do

⑩ while *CandWalkList* 的每个成员

⑪ if 笛卡儿积非空

⑫ 将 s 追加到走长度为 t 的候选集中;

⑬ end for

⑭ return 结果集 R.

4 实验设计和性能分析

本节在 GPU 架构上实现了 RDF 图上的类型同构算法,并对其进行了性能测试. 经过与 CPU 架构下的类型同构算法的效率对比,并将 gStore 中基于子图同构的方法与本文中的算法进行比较,证明了 GPU 架构解决类型同构问题的效率优于 CPU 架构,进一步说明 GPU 在解决 RDF 数据查询问题上具有突出的性能.

4.1 CPU 与 GPU 的比较实验

1) 实验环境和数据集

本实验平台的配置为 NVIDIA GTX590 1.35 GHz GPU 的显卡和运行英特尔® Core(i)i7-26 001.35 GHz 的4 核处理器. 操作系统版本为 Linux Ubuntu 14.04,内存大小为 2 GB,GPU 的设备内存为 1 536 MB. GPU 使用 PCI-E3.0 总线在主存储器和设备存储器之间进行数据传输数据,其传输带宽为 4 GB/s.

实验采用 LUBM(Lehigh University Benchmark)数据集. LUBM 是一个采用大学领域内本体的基准,可以生成任意大小的 OWL 数据类型. 这些数据中包含了教师、学生、课程等信息之间的关联信息. 由于本算法关注的是 RDF 数据的图匹配问题,需要对产生的 OWL 数据进行预处理,将其转化为系统处理所需的 RDF 三元组格式. 本实验选取了 7 个不同规模的 LUBM 数据集来测试算法的性能. 表 1 详细描述了这 7 个数据集的数据特征和规模. 为了对本文提出的算法进行测试与分析,本实验选取了 2 种查询方案,如表 2 所示.

Table 1 LUBM Datasets of Experiment

表 1 实验使用 LUBM 数据集

Datasets	TripleNumber	Datasets Size/MB	Entity
LUBM1	103 104	12.55	17 191
LUBM2	237 210	28.90	38 351
LUBM4	493 844	60.19	78 596
LUBM8	1 036 086	126.30	163 569
LUBM20	2 782 126	341.41	437 572
LUBM40	5 495 742	676.19	864 239
LUBM60	8 287 974	1 020.73	1 302 481
LUBM80	11 108 166	1 372.16	1 744 943
LUBM100	13 879 970	1 711.10	2 179 783

Table 2 Benchmark Test Queries

表 2 基准测试查询语句

No.	Query
Q_1	SELECT?x WHERE{?x rdf:type ub:GraduateStudent. ?x ub:takesCourse GraduateCourse0.}
Q_2	SELECT?x,?y WHERE{?x rdf:type ub:Chair. ?y rdf:type ub:Department. ?x ub:worksFor?y. ?y ub:subOrganization-Of University0.}

2) CPU 与 GPU 实验结果和分析

本文选取了 LUBM 查询中具有代表性的查询语句作为实验的基准测试查询,分别对 7 个不同规模的数据集上进行测试,以比较相同环境下 CPU 和 GPU 的性能. 表 3 和表 4 中分别记录了每个查询仅使用 CPU 计算单元计算类型同构匹配算法的响应时间、使用基于 GPU 计算单元加速类型同构匹配算法的查询响应时间、以及它们之间的加速比. 表 3 和表 4 中的查询响应时间均为多次测试后的平均响应时间. 本实验所测试的算法查询响应时间仅包括从用户提交查询到查询结束之间的时间,不包括数据的载入时间和数据 IO 的传输时间.

Table 3 Query Response Time of Q_1 over LUBM Datasets

表 3 Q_1 在 LUBM 数据集上的查询响应时间 ms

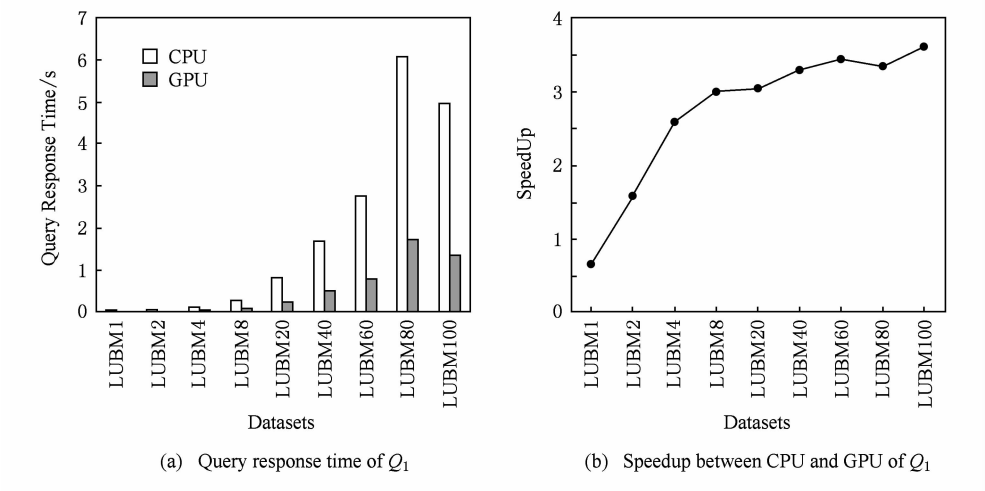
Datasets	Query Response Time		
	CPU	GPU	SpeedUp
LUBM1	28.30	43.12	0.65
LUBM2	66.82	42.03	1.58
LUBM4	142.63	55.02	2.59
LUBM8	306.48	102.05	3.00
LUBM20	826.40	270.86	3.05
LUBM40	1 725.57	524.12	3.29
LUBM60	2 781.56	808.58	3.44
LUBM80	6 088.31	1 744.51	3.34
LUBM100	4 955.43	1 371.61	3.61

Table 4 Query Response Time of Q_2 over LUBM Datasets
表 4 Q_2 在 LUBM 数据集上的查询响应时间 ms

Datasets	Query Response Time		
	CPU	GPU	SpeedUp
LUBM1	56.96	114.37	0.49
LUBM2	134.47	112.55	1.19
LUBM4	286.61	141.03	2.03
LUBM8	616.35	227.71	2.71
LUBM20	1 660.88	608.44	2.72
LUBM40	3 466.52	1 103.94	3.14
LUBM60	5 587.44	1 599.62	3.49
LUBM80	7 248.28	2 208.06	3.28
LUBM100	9 616.86	2 616.41	3.67

图 7(a)为查询 Q_1 在不同规模 LUBM 数据集上的查询响应时间,图 7(b)为基于 CPU 的查询响应时间与基于 GPU 查询的加速比. 图 8 则分别为查询 Q_2 在不同规模 LUBM 数据集上的查询响应时间和基于 CPU 的查询响应时间与基于 GPU 的加速比.

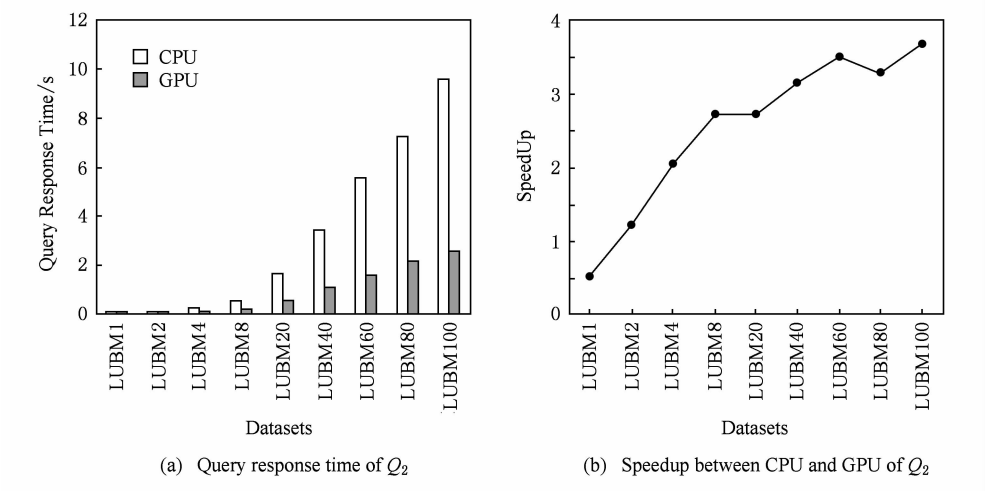
由实验结果分析,在数据量较少的情况下,例如 LUBM1,本算法的查询响应时间与基于 CPU 的算法的时间近似. 原因是在初始化时 CPU 调用 GPU 进行数据分配调用的过程需要额外耗费一定通信代价. 但是随着数据集的不断增大,由于 GPU 并行计算能力强,GPU 的查询响应时间远远低于 CPU 的运行时间. 图 7 和图 8 所示,随着数据量的不断增大,GPU 的查询响应时间明显低于 CPU. 且 CPU 和 GPU 的加速比随着数据量的增大而逐渐上升,最后稳定在 3 倍左右. 说明在类型同构并行算法中,GPU 的查询效率远高于 CPU.



(a) Query response time of Q_1 (b) Speedup between CPU and GPU of Q_1

Fig. 7 Query response time and speedup between CPU and GPU of Q_1

图 7 Q_1 的查询响应时间及 CPU 与 GPU 运算的加速比



(a) Query response time of Q_2 (b) Speedup between CPU and GPU of Q_2

Fig. 8 Query response time and speedup between CPU and GPU of Q_2

图 8 Q_2 的查询响应时间及 CPU 与 GPU 运算的加速比

对比实验结果数据,本文提出的基于 GPU 的 RDF 类型同构的并行算法性能显著高于 CPU 处理的性能.

4.2 类型同构与子图同构算法比较结果和分析

1) 实验环境和数据集

本实验平台的配置有 NVIDIA Tesla M40 GPU 的显卡和运行英® Xeon® E5-2603 v4 1.7 GHz 的 6 核处理器.操作系统版本为 Linux Ubuntu 14.04 Server,内存大小为 8 GB,GPU 的设备内存为 24 GB.本实验使用数据集及查询语句如表 5 和表 6 所示.

Table 5 LUBM Datasets of Experiment

表 5 实验使用 LUBM 数据集

Datasets	TripleNumber	Datasets Size/MB	Entity
LUBM10	1 316 700	160.56	207 443
LUBM20	2 782 126	341.41	437 572
LUBM30	4 109 002	505.10	645 971
LUBM40	5 495 742	676.19	864 239
LUBM50	6 890 640	848.32	137 216
LUBM100	13 879 970	1 711.10	2 179 783

Table 6 Benchmark Test Queries

表 6 基准测试查询语句

No.	Query
Q_3	SELECT?x WHERE{?x ub:worksFor http://www.Department0.University0.edu. ?x rdf:type ub:FullProfessor. ?x ub:name? y_1. ?x ub:emailAddress? y_2. ?x ub:telephone? y_3.}
Q_4	SELECT?x?y WHERE{?x rdf:type ub:Student. ?y rdf:type ub:Course. ?x ub:takesCourse?y. AssociateProfessor0 ub:teacherOf?y.}

为了与 4.1 节中实验区分,本文选取了 LUBM 查询中的较为复杂的查询语句(作为实验的基准测试查询,分别对 8 个不同规模的数据集上进行测试.比较相同环境下基于子图同构的 RDF 数据查询引擎与本文所提算法性能.

2) CPU 与 GPU 实验结果和分析

本文选取了 LUBM 查询中具有代表性的查询语句作为实验的基准测试查询,分别对 7 个不同规模的数据集上进行测试.

表 7 和表 8 中的记录了 Q_3 和 Q_4 的查询响应时间与加速比.并在图 9 和图 10 中以图表形式分别进行展示.

Table 7 Query Response Time of Q_3 over LUBM Datasets

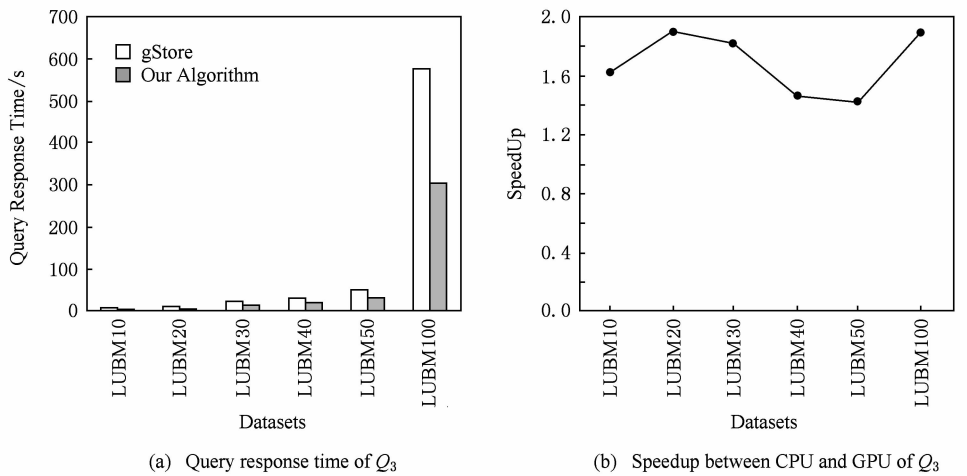
表 7 Q_3 在 LUBM 数据集上的查询响应时 ms

Datasets	Query Response Time		
	gStore	Our Algorithm	SpeedUp
LUBM10	6 282	3 877	1.62
LUBM20	10 615	5 596	1.89
LUBM30	23 178	12 792	1.81
LUBM40	31 696	21 730	1.45
LUBM50	49 821	35 162	1.41
LUBM100	574 950	304 579	1.88

Table 8 Query Response Time of Q_4 over LUBM Datasets

表 8 Q_4 在 LUBM 数据集上的查询响应时间 ms

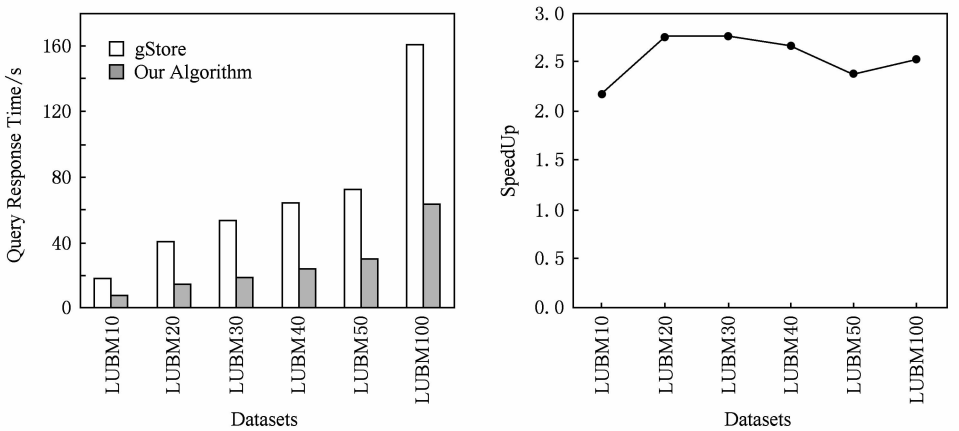
Datasets	Query Response Time		
	gStore	Our Algorithm	SpeedUp
LUBM10	19 165	8 842	2.16
LUBM20	41 031	14 972	2.74
LUBM30	54 126	19 655	2.75
LUBM40	64 730	24 368	2.65
LUBM50	73 133	30 755	2.37
LUBM100	161 380	64 137	2.51



(a) Query response time of Q_3 (b) Speedup between CPU and GPU of Q_3

Fig. 9 Query response time and speedup between CPU and GPU of Q_3

图 9 Q_3 的查询响应时间及 CPU 与 GPU 运算的加速比



(a) Query response time of Q_4 (b) Speedup between CPU and GPU of Q_4

Fig. 10 Query response time and speedup between CPU and GPU of Q_4

图 10 Q_4 的查询响应时间及 CPU 与 GPU 运算的加速比

由图表分析所得,本实验所提方案与目前性能较好的集中式查询引擎 gStore 进行比较. 与 gStore 相比,Q_3 的加速比约在 1.4～1.9 倍,本算法查询响应时间仅为其响应时间的 1/2 左右. Q_4 的加速比约在 2.1～2.7 倍,本算法查询响应时间仅为其响应时间的 1/3 左右. 本算法总体性能提高了 1.4～2.7 倍. 随着数据量的增大,本实验所提方案提升的性能显著.

经过查询对比发现,本算法在包含链状的查询(如 Q_1 和 Q_4),展现了良好的性能. 但是在只含有星状查询(如 Q_3)中,其查询性能则略低于包含链状的查询. 结合本文的理论基础分析,算法在星状查询中展开成 k 边游走的过程中可能导致边的重复遍历,因而导致了算法的性能有所下降.

通过对比实验,本文提出的基于 GPU 的 RDF 类型同构的并行算法性能显著高于基于 CPU 的 RDF 子图同构的处理方法.

5 总 结

依据 RDF 图本身的数据特征,传统的子图同构的问题可以用来解决 SPAQRL 中基本图模式查询问题. 本文基于 IRSMG 的方法和理论,提出了一种基于 GPU 的 RDF 类型同构并行算法,以高效地完成基本图模式查询,减少查询响应时间. 本文通过研究约束的类型同构问题,将问题引申至 RDF 图的模式匹配和查询. 通过设置游走等级的方法,对满足查询图标签的顶点进行遍历检查,确定后续查找范围. 最后得到每个三元组模式的局部解,使用 GPU 并行的进行连接操作. 因此最大程度利用了 GPU 的计算能力,使基本图模式查询性能有了显著的提高.

尽管本工作研究的问题仅限于基本图模式查询,但是其性能优于 CPU 的处理方式. 这为以后 RDF 数据处理的发展提供了新的思路. 由于现在本文所提出的算法仅能支持部分基本图模式,例如涉及星状和链状的查询,对于包含环状的查询则体现了较为一般的性能. 因此如何将本算法的思想推广到基本图模式一般情况将是本文后续工作.

参 考 文 献

[1] Mcbride B. Jena: Implementing the RDF model and syntax specification [C] //Proc of the 5th ISWC'01. New York: ACM, 2001: 23-28

[2] Neumann T, Weikum G. RDF-3X: A RISC-style engine for RDF [J]. VLDB Journal, 2008, 1(1): 647-659

[3] Zou Lei, Özsu M, Chen Lei, et al. gStore: Answering SPARQL queries via subgraph matching [J]. VLDB Journal, 2014, 23(4): 565-590

[4] Pérez J, Arenas M, Gutierrez C. Semantics and Complexity of SPARQL [M]. Berlin: Springer, 2006: 1-45

[5] Zhang Junzhao, Zhang Bingyi, Zhang Xiaowang, et al. IRSMG: Accelerating inexact RDF subgraph matching on the GPU [C] //Proc of ISWC'16. Berlin: Springer, 2016

[6] Ullmann J. An algorithm for subgraph isomorphism [J]. Journal of the ACM, 1976, 23(1): 31-42

[7] Cordella L, Foggia P, Sansone C, et al. A (sub) graph isomorphism algorithm for matching large graphs [J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2004, 26(10): 1367-1372

[8] Shang Haichuan, Zhang Ying, Lin Xuemin, et al. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism [J]. VLDB Journal, 2008, 1(1): 364-375

[9] He Huahai, Singh A. Graphs-at-a-time: Query language and access methods for graph databases [C] //Proc of ICDM'08. New York: ACM, 2008: 405-418

[10] Zhang Shijie, Li Shirong, Yang Jiong. GADDI: Distance index based subgraph matching in biological networks [C] //Proc of EDBT'09. New York: ACM, 2008: 405-418

[11] Zhao Peixiang, Han Jiawei. On graph query optimization in large networks [J]. VLDB Journal, 2010, 3(1-2): 340-351

[12] Sun Zhao, Wang Hongzhi, Wang Haixun, et al. Efficient subgraph matching on billion node graphs [J]. VLDB Journal, 2012, 5(9): 788-799

[13] Lyu Xuedong, Wang Xin, Li Yuanfang, et al. FFD-Index: An efficient indexing scheme for star subgraph matching on large RDF graphs [G] //LNCS 9052: Proc of Database Systems for Advanced Applications. Berlin: Springer, 2015: 240-245

[14] Berry J, Hendrickson B, Kahan S, et al. Software and algorithms for graph queries on multithreaded architectures [C] //Proc of IPDPS'07. Los Alamitos, CA: IEEE Computer Society, 2007: 1-14

[15] Xu Xinhai, Lin Yufei, Yi Wei. A survey of techniques of CPU-GPGPU heterogeneous architecture [J]. Computer Engineering & Science, 2009, 31(S1): 24-26 (in Chiness)
(徐新海, 林宇斐, 易伟. CPU-GPGPU 异构体系结构相关技术综述[J]. 计算机工程与科学, 2009, 31(S1): 24-26)

[16] Merrill D, Garland M, Grimshaw A. Scalable GPU graph traversal [J]. ACM Sigplan Notices, 2012, 47(8): 117-128

[17] Katz G, Kider J. All-pairs shortest-paths for large graphs on the GPU [C] //Proc of GH'08. New York: ACM, 2008: 47-55

[18] Vineet V, Harish P, Patidar S, et al. Fast minimum spanning tree for large graphs on the GPU [C] //Proc of HPG'09. New York: ACM, 2009: 167-171

[19] Heino N, Pan J. RDFS Reasoning on massively parallel hardware [G] //LNCS 7649: Proc of ISWC'12. Berlin: Springer, 2012: 133-148

[20] Choksuchat C, Chantrapornchai C. Large RDF representation framework for GPUs case study key-value storage and binary triple pattern [C] //Proc of ICSEC'2013. Piscataway, NJ: IEEE, 2013: 13-18

[21] Choksuchat C, Chantrapornchai C. Experimental framework for searching large RDF on GPUs based on key-value storage [C] //Proc of JCSSE'13. Piscataway, NJ: IEEE, 2013: 171-176

[22] Choksuchat C, Chantrapornchai C. On the HDT with the tree representation for large RDFs on GPU [C] //Proc of the 19th ICPADS'13. Los Alamitos, CA: IEEE Computer Society, 2013: 651-656

[23] Chantrapornchai C, Choksuchat C, Haidl M, et al. Entailment processing for large RDF data sets using GPU [C] //Proc of SoMeT'16. Clifton, NJ: IOS, 2016: 333-345

[24] Haidl M, Gorlatch S. TripleID: A low-overhead representation and querying using GPU for large RDFs [C] //Proc of BDAS'16. Berlin: Springer, 2016: 400-415

[25] Fu Zhisong, Personick M, Thompson B. MapGraph: A high level API for fast development of high performance graph analytics on GPUs [C] //Proc of GRADES'14. New York: ACM, 2014: 1-6

[26] Yang Bo, Lu Kai, Gao Yinghui, et al. GPU acceleration of subgraph isomorphism search in large scale graph [J]. Journal of Central South University, 2015, 22(6): 2238-2249

[27] Fu Lizhen, Meng Xiaofeng. Reachability indexing for large-scale graphs: Studies and forcasts [J]. Journal of Computer Research and Development, 2015, 52(1): 116-129 (in Chiness)
(富丽贞, 孟小峰. 大规模图数据可达性索引技术:现状与展望[J]. 计算机研究与发展, 2015, 52(1): 116-129)

[28] Zou Lei, Peng Peng. A survey of distributed RDF data management [J]. Journal of Computer Research and Development, 2017, 54(6): 1213-1224 (in Chiness)
(邹磊, 彭鹏. 分布式 RDF 数据管理综述[J]. 计算机研究与发展, 2017, 54(6): 1213-1224)

[29] Li Jianhui, Shen Zhihong, Meng Xiaofeng. Scientific big data management: Concepts, technologies and system [J]. Journal of Computer Research and Development, 2017, 54(2): 235-247 (in Chiness)
(黎建辉, 沈志宏, 孟小峰. 科学大数据管理: 概念、技术与系统[J]. 计算机研究与发展, 2017, 54(2): 235-247)

Feng Jiaying, born in 1993. Master from the School of Computer Science and Technology, Tianjin University. Member of CCF. Her main research interests include parallel processing and RDF data management.

Zhang Xiaowang, born in 1980. Received his PhD degree from Peking University in 2011. Now he is associate professor at Tianjin University. Member of CCF. His main research interests include artificial intelligence, databases and knowledge graph etc.

Feng Zhiyong, born in 1965. PhD and professor in the School of Software, Tianjin University. Senior member of CCF. His main research interests include knowledge engineering, service technology and security software engineering.