软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(5):1053−1072 [doi: 10.3724/SP.J.1001.2012.04008]

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

非规则流中高维数据流典型相关性分析并行计算方法*

周 勇[1+], 卢晓伟[1], 程春田[2]

[1](大连理工大学 软件学院,辽宁 大连 116620)

[2](大连理工大学 水利学院,辽宁 大连 116624)

Parallel Computing Method of Canonical Correlation Analysis for High-Dimensional Data Streams in Irregular Streams

ZHOU Yong[1+], LU Xiao-Wei[1], CHENG Chun-Tian[2]

[1](School of Software, Dalian University of Technology, Dalian 116620, China)

[2](School of Hydraulic Engineering, Dalian University of Technology, Dalian 116024, China)

+ Corresponding author: E-mail: kevinzh@dlut.edu.cn

Zhou Y, Lu XW, Cheng CT. Parallel computing method of canonical correlation analysis for high-dimensional data streams in irregular streams. *Journal of Software*, 2012,23(5):1053−1072. http://www.jos.org.cn/1000-9825/4008.htm

Abstract: This paper addresses an approach that uses GPU (graphic processing unit)-based processing architecture model and its parallel algorithm for high-dimensional data streams over the irregular streams in order to satisfy the real-time requirement under the resource-constraints. This six layers model combines the GPU high wide-band property of data processing with analysis data stream in a sliding window. Next, canonical correlation analysis is carried out between two high-dimensional data streams, by a data cube pattern, and a dimensionality-reduction method in this framework based on compute unified device architecture (CUDA). The theoretical analysis and experimental results show that the parallel processing method can detect correlations on high dimension data streams, online, accurately in the synchronous sliding window mode. According to the pure CPU method, this technique has significant speed advantage and conducts the real-time requirement of high-dimensional data stream very well. It provides a common strategy for the applied field of data stream mining.

Key words: graphic processing unit (GPU); high-dimensional data stream; canonical correlation; compute unified device architecture; dimensionality-reduction technique

摘 要: 为了满足在计算资源受限的环境下高维数据流处理的实时性要求,提出一种方法——基于 GPU(graphic processing unit)的非规则流中高维数据流的处理模型和具体的可行架构,并分析设计了相关的并行算法.该六层模型是将 GPU 处理数据的高宽带性能结合进滑动窗口中数据流的分析,进而在该框架下基于统一计算设备架构(compute unified device architecture,简称 CUDA),使用数据立方模型以及降维约简技术并行分析了多条高维数据流的典型相关性.理论分析和实验结果均表明,该并行处理方法能够在线精确地识别同步滑动窗口模式下高维数据流之间的相关性.相对于纯 CPU 方法,该方法具有显著的速度优势,很好地满足了高维数据流的实时性需求,可以作为

* 基金项目: 国家杰出青年基金(51025934)
收稿时间: 2010-04-26; 修改时间: 2010-12-09; 定稿时间: 2011-02-17

通用的分析方法广泛应用于数据流挖掘领域.

关键词: 图形处理器;高维数据流;典型相关性;统一计算设备架构;降维约简技术
中图法分类号: TP311 文献标识码: A

现实生活中,电力实时监测数据流、零售业务中的交易数据流、医学监测的数据流、通信领域中的电话记录数据流、网络监测中的数据包流、环境温度的监测数据、各类传感器网络中的检测数据流以及卫星传回的图像数据流都以高维属性的形式发挥着各方面的作用,形成了一种与传统数据库中静态数据以及单维属性流数据不同的数据形态.而高维数据流之间的相关性的应用越来越广,主要目的是分析多条并行到达的数据流之间的关联.关联度计算是指在多条数据流中,计算每对数据流之间的关联系数,从而发现具有高的正关联或负关联的数据流对.例如,风险投资公司在投资决策时,对各种股指、债券的实时相关性分析信息,可以帮助决策者捕获稍纵即逝的投资良机;在 Web 和 Internet 的管理中,对访问模式的相关性分析可以指导预取和缓存策略,有助于预测未来可能的请求等.

现有的多数据流关联分析主要采用 3 种方法,即计算数据流对之间的关联系数[1,2]、计算多条数据流的主分量[3,4]以及计算多条数据流中存在的聚类[5,6].本文采用第 1 种方法来进行数据流相关性分析.

由于数据流本身不断变化且难以预测的特点以及数据流突发的产生对数据流负载能力提出了更高的要求[7].同时,计算数以千计的高维数据流中的任意两对数据流的相关系数,n 条数据流需要计算 $n(n-1)/2$ 个相关系数,其串行的精确算法因为时间开销过大而难以满足实时性需求,因此,研究和利用高速的流处理器提高数据流处理吞吐量成为数据流处理研究领域的热点问题之一.

GPU(graphic processing unit)作为一种新型流处理器具备了流处理模型的特点,而且集成了流加速部件.GPU 以其高速的浮点运算能力迅速地吸引了人们的眼球.近 5 年来,GPU 以超过摩尔定律的每年 2.5 倍~3 倍的速度不断增长.主流的家用图形处理器的浮点运算能力已经达到 950GFLOP/s,而同期的主流 CPU 浮点运算能力只有 100GFLOP/s,在向量计算方面能够获得比 CPU 高出 10 倍的计算效率.如今,第五代可编程图形处理器已近发展成一种多核多线程、高内存带宽、高并行计算能力的通用并行计算设备.

CUDA 是用于 GPU 计算的开发环境,它是一个全新的软硬件架构,可以将 GPU 视为一个并行数据计算的设备,对所进行的计算进行分配和管理.在 CUDA 的架构中,这些计算不再像过去所谓 GPGPU 架构那样必须将计算映射到图形 API(OpenGL 和 Direct 3D)中,因此对于开发者来说,CUDA 的开发门槛大大降低了.CUDA 的 GPU 编程语言是基于标准的 C 语言,因此任何有 C 语言基础的用户都能够很容易地开发 CUDA 的应用程序.因此,图形处理器通用并行计算已经成为一个研究热点[8].

1 相关工作

目前,多数据流分析挖掘领域有许多关于相关性分析的工作,王永利等人[9]提出一种新颖的基于典型相关性分析(CCA)的高维数据流相关性分析算法 StreamCCA,针对传统的 CCA 计算中的性能瓶颈,提出为样本方差阵与协差阵组成的乘积阵降维的高效低价近似方法,在保持分析精度的前提下显著地提高了计算效率.Zhu 和 Shasha[2]提出一种基于时间序列数据流模式的 StatStream 模型,用于发现在任意时刻相关的两条流.文中采用基窗口的离散 fourier 变换来近似计算流的相关性,计算相关系数时只考虑了数据流单个属性值.他们没有对数据流对之间存在滞后关联的情况作太多讨论,但是这种情况在应用中比较常见.Bulut 和 Singh[10]提出一种多分辨索引模式,监视不同的时间尺度,有效地解决了可变长度的查询问题.其数据流计算模型是…$x[i]$,…,解决的 3 种查询聚集、模式监控和相关性查询都是针对单维数据流的,其中,计算两个流序列 x 和 y 的相关性被简化为计算它们之间 z-norm 之内的 Euclidean 距离,属于单变量统计.文献[11]给出了检测两条单属性数据流之间相关性的方法,在给定阈值和时间窗长度的情况下检测相关系数小于阈值的流对(S_i,S_j),并记录 i 与 j 之间的时间偏移量;文献[12]提取多时间序列数据流上的时间相关性,探测所有相关对,从相关方向、敏感度、延迟时间等几个方面定义几种相关性规则的指标;MUSCLES[13]模型使用序列历史信息和其他有关序列信息,基于线性回归实现了

多数据流相关性的预测.上述 3 种方法都是处理时间序列模式的数据流,不具有最一般的通用性,而且其共同的缺点是要在相当长的序列上才能完成检测,需要计算复杂性更高的增量分析技术.

图形处理器通用并行计算以及在数据流领域的应用研究主要有:王磊等人[14]针对 GPU 图形处理的特点,分析了其应用于通用计算的并行处理机制和数据映射,提出了一种 GPU 通用计算模式的映射机制和一般性设计方法,并针对 GPU 的吞吐量、数据流处理能力和基本数学运算能力等进行性能测试,为 GPU 通用计算的算法设计、实现和性能优化提供参考依据.苏畅等人[15]设计了一种在图形处理器(GPU)上完成大型矩阵快速运算的方法,主要通过使用 Kahan 求和公式来确保计算精度,根据 GPU 特点设计矩阵分块方式和内存分配机制来减少对数据访问频次,以发挥 GPU 的并行体系结构特性来提高计算速度.李建民等人[16]为了改善遗传算法对大规模多变量求解的性能,提出一种基于图形处理器(GPU)加速细粒度并行遗传算法的实现方法.将并行遗传算法求解过程转化为 GPU 纹理渲染过程,使得遗传算法在 GPU 中加速执行.曹锋、周傲英等人[17]提出了使用 GPU 对数据流做聚类的加速方法.

综上所述,首先,目前数据流挖掘领域中有关相关性分析的工作主要是针对单属性数据流的分析,本质上都是属于单变量统计分析的方法,无法有效地解决检测 2 条乃至多条高维数据流之间的相关关系,即多从多个角度进行探测事物的性质.目前,数据流研究领域基于多变量统计分析理论分析高维数据流之间相关性的文献还比较少.

然而,利用图形渲染流水线进行通用计算存在数据交换和寻址模式过分依赖于纹理映射和纹理贴图本身数据精度较低的缺陷.而统一计算设备架构(compute unified device architecture,简称 CUDA)[18]则可以解决这些缺陷.基于 CUDA 的数据流并行处理研究在国内外还处于新领域,鲜有相关的研究成果,而使用 GPU 技术进行高维数据流之间的相关性分析的研究成果更是一个非常有意义的工作,也是本文的创新之处.

本文的主要工作如下:

(1) 结合图形处理器和数据流自身的特点,分析 GPU 协处理数据流的六层模型以及具体可行的实施框架,这是数据流并行处理的难点,也是本文研究得重点和难点;

(2) 提出基于 CUDA 架构的支持滑动窗口的高维数据流典型相关性分析的并行算法,以及在显存上维护高维数据流概要数据结构的并行算法,减少主机与设备间的数据交换,进而提高数据流吞吐量和算法整体性能.

2 非规则流中高维数据流的 GPU 处理模型

2.1 基于GPU的非规则流中高维数据流的处理模型

数据流实际上就是连续移动的元素队伍,其中的元素是由相关数据的集合组成.

定义 2.1. 一维顺序访问的定长流称为规则流[19],此外的流称为非规则流.

非规则流中元素的访问序列可能是间断的、跳跃的、动态产生的或重复的,还可能依据条件取舍.其实际长度与访问长度并无特殊关系,是不确定的.元素的结构可能是多种多样的,不同元素所占的存储空间可以不一样.可以针对各个流元素的位置、值的不同进行计算处理.因此,需要解决非规则流的实时处理问题.

流处理模型是一种高可预知的结构化模型.在流程序级,Kernel 和 Stream 之间可以看作生产者和消费者的关系[20]将任务级并行显示地暴露出来,同时,流的数据成批的特性和 Kernel 内部的密集计算分别揭示了丰富的数据级并行和指令级并行.流处理模型核心是将应用分解成一连串对流进行操作的计算核心(kernel),流在 Kernel 之间传递,在 Kernel 内部完成对流中元素的处理.

本文从一个全新的角度构建高维数据流实时处理的系统模型,在体系结构上采用并行思想和前端预处理技术,国内外还缺乏这方面的研究成果.本文在滑动窗口模型的基础上提出了基于 GPU 的高维数据流并行处理的六层架构模型.如图 1 所示.

(1) 时序数据处理层:时序数据层中的数据是符合非规则流的一些约束的含高维属性的元组,该层的任务是由 CPU 进行处理的:数据 IO 和数据过滤.在这一层中,对原数据流分 3 次处理:流速率调节、属性包装和数据

过滤.

流速率调节负责负载平衡和决定何时启动GPU内核.当数据流的流量达到一定规模时,可以启动GPU内核对数据流进行处理,作为一种提高数据流处理性能的手段.如果启动 GPU 内核协处理数据流,CPU 每次可以较大的分块将数据交换到 GPU 设备,然后利用图形处理器密集的数据计算能力和高内存带宽的特性对数据流进行高速处理.

属性包装和数据过滤负责对数据流进行加工以改善数据流的质量,为流数据的连续查询和复杂分析打下基础,包括对流数据进行去噪、压缩编码、修正以减少存储空间和传输时间.在整个层中元组按时间 t 有序,如果任一高维数据流在时刻 t 未接收到元组,则以全 0 的元组代替,即允许稀疏数据流矩阵存在.

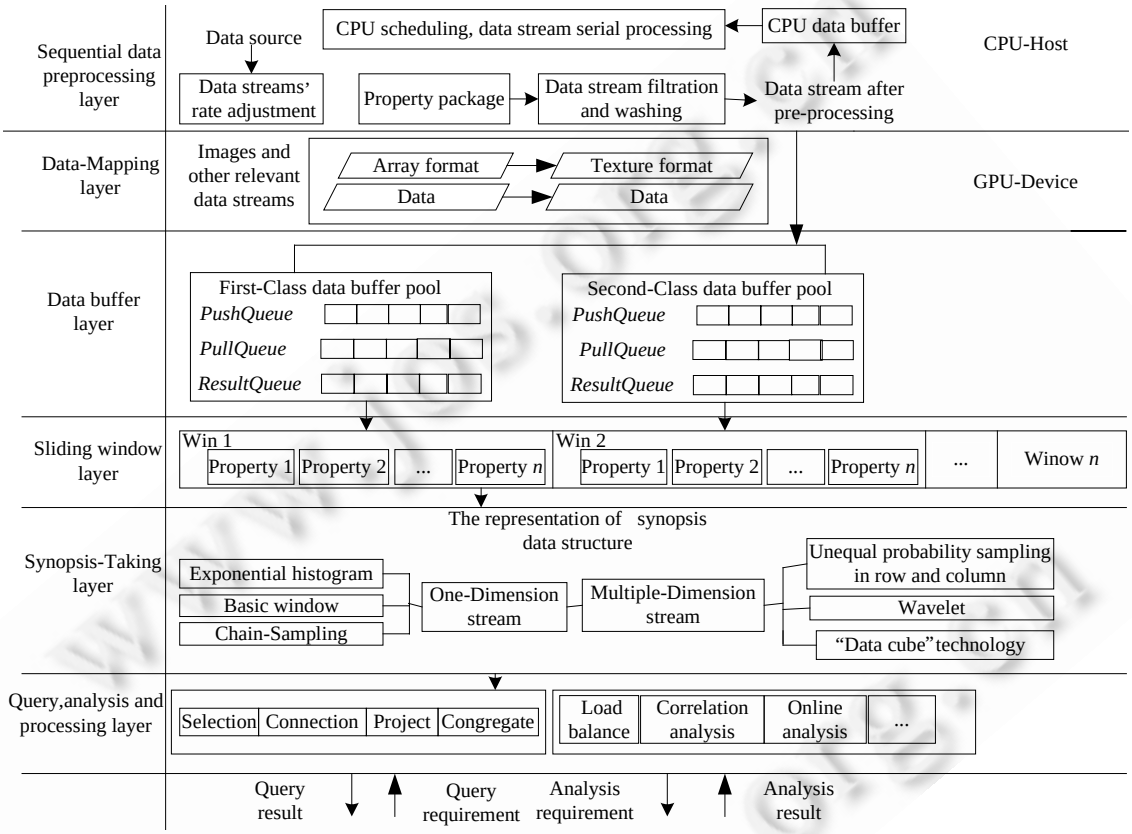

Fig.1 Six-Level architecture model of high dimensions data streams processing based on GPU

图 1 基于 GPU 的高维数据流并行处理的六层架构模型

(2) 数据映射层:对于图像相关的数据流,将 CPU 数组结构映射为 GPU 纹理结构,事先建立对应表达式 $C(x)=G(y)$,通过数据 $C(x)$属性和纹理数据属性 $G(y)$建立映射关系,找出 CPU 和 GPU 两类数据的一一对应关系,利用通用计算数据结构对应纹理在 GPU 存放纹理各个属性的方式.

(3) 数据缓冲层:包含两级数据缓冲池,主要维护几类缓冲,包括注册的查询计划缓冲、同步数据查询操作工作区、数据流历史概要缓冲、数据流缓冲这 4 种性质的存储.数据缓存借助两种类型的队列实现推式(传感器数据)和拖式(传统关系数据)操作,减少主机和设备间的数据交换频率,提高高维数据流并行计算的密集程度.数据缓冲层是图形处理器对高维数据流并行处理的基本单位.经过数据清理以及映射处理的缓冲窗口作为普通的子窗口更新到滑动窗口中.

(4) 滑动窗口层:流矩阵以队列的方式实现,队列中存放流数据的高维属性,进入滑动窗口.更新矩阵时(即

时间窗前滚),用新元组替换掉最旧的元组,实现同步高维数据流概要数据结构增量归一化.

(5) 概要矩阵抽取层:生成高维数据流的概要结构矩阵,存放概要数据矩阵的层次.实际上,高维数据流最便于用二维矩阵表示,每行或每列表示一条数据流,每个元素表示各个维度的属性.对于高维数据流,此层分 3 个区域,根据高维数据流的特点分别采用不等概行向兼列向采样技术、小波技术、数据立方技术进行.

(6) 查询分析处理层:完成各种基本查询和复杂分析.基本查询处理操作完成一般的选择、连接、投影和聚集等连续查询请求.复杂分析处理操作则是在信息提取层生成的概要信息基础上,进一步研究数据流中各种因素的性质和相互之间的关系.基本查询操作既可以直接在预处理过的数据流上执行,又可以在概要上执行.

2.2 基于GPU的高维数据流的处理模型分析

2.2.1 高维数据流 GPU 处理模式的优势

(1) 宏观粗粒度分析

在各个实时数据领域,数据密度高,多维属性构成了庞大的流数据,计算模式已经过渡到了众核计算上来.由于计算问题变得很复杂,必须寻找新的方式来提高计算速度.这些高维数据流处理的性能需求非常大,使得之前的计算处理器性能无法满足要求.本文的数据流处理模型和框架式建立在 NIVIDA Tesla C1060 的 Fermi 基础上,利用图形处理的技术思想,创新性地放在数据流的处理上,解决了性能瓶颈问题和复杂处理架构.由于 Tesla C1060 拥有 240 个处理器核心,使得本文研究的数据流处理模型具有超级并行计算能力,可以替代小型集群性能,能够同时处理数以千计的计算线程.通过这一性能,能够更快地获得精确的结果.在 GPU 这样的处理器上能够保证高速运行本文提出的数据流计算的高性能,使得高维数据流的处理中遇到的难题在粗粒度都可以解决.

(2) 微观细粒度分析

根据 GPU 的特点,GPU 非常适合处理那些能够表示为数据并行计算的问题,这些数据并行计算的算术操作和存储器操作的算术计算密度非常高.由于同一计算程序在多个元素上执行和高计算密度,访存延迟可以被计算隐藏,利用 GPU 的图像块和像素映射等技术,将本文提出的模型中的数据流元素映射到并行处理的线程上,而且现有数据流的算法有很多适于并行计算的特质,可以将相同指令应用于不同数据,利用并行处理模型加速具有很好的优势.

在技术实施上,对于同一个 CPU 主机线程而言,CPU 代码和 GPU 内核程序是串行的,即对同一个 CPU 线程而言,在该线程内的 GPU 内核程序执行时,CPU 只能等待内核程序执行完毕才能继续执行.不过,主机上可以采用多线程的方式或者 GPU 中的并行优化策略来提高性能,采用多线程的方式可以实现 GPU 设备和 CPU 主机同时进行计算不同的任务,本文进而并行优化策略流技术,流是按照顺序执行的一系列操作,而不同的流与其他的流之间则是乱序执行或是并行执行.这样,可以使一个流的计算与另外一个流的数据传输同时进行,充分利用了 GPU 中的资源.这样完成 CPU 与 GPU 的协作,使得高维数据流处理的具体实施在微观细粒度上可以解决.

2.2.2 GPU 处理模式中流数据的流动模式分析

(1) 流数据流动延迟

基于 GPU 的高维数据流处理最大的瓶颈就是主机端和设备端的数据传输的开销.GPU 不能单独为某个处理核心分配任务,因此必须先缓冲一定量的高维流数据,再交给 GPU 进行计算,从而获得很高的数据吞吐量.为了满足数据流处理的实时性要求,通过减少缓冲,可以减小延迟,但缓冲的大小应该保证每个内核程序处理的一批数据能够让 GPU 满负荷工作.

(2) 流数据的流动量

对于数据密度大的高维数据流量,衡量其计算量有相对和绝对两种方式:

① 从绝对量来说:如果待优化的高维数据流处理程序使用频率比较低,并且每次调用需要的时间可以接受,那么使用基于GPU架构的高维数据流处理模型并不会显著改善使用体验.因为在GPU上的计算执行时间无法隐藏访存和数据传输的延迟,此时,整个应用程序需要的时间反而会比 CPU 更长.虽然 GPU 的单精度浮点处理能力和显存带宽都远远超过了 CPU,但是由于 GPU 使用 PCI-E 总线与主机连接,因此其输入和输出的吞吐量受到了 I/O 带宽的限制,此时无论如何提高浮点处理能力和显存带宽都无法提高系统带宽.

② 从相对计算量来说,如果高维数据流并行处理的部分在整个应用中所占比例不大,那么 GPU 对计算整体性能的提高也不会非常明显.只有在并行计算占用了绝大多数计算时间的应用中,使用 GPU 加速才能获得很高的加速比.

(3) 研究的可行策略

本文充分考虑了由于 CPU 和 GPU 的 IO 开销导致高性能计算性能的下降问题,在计算框架中减小这种开销.基于 Tesla C1060 的 Fermi 计算卡显存可达 4G,这样可以在使用 GPU 并行生成概要数据结构的应用时,减小数据流的流动次数;同时,尽可能地规划利用 GPU 显存容量来完全存放高维流数据.由于本文的计算框架模型使用了 GPU 对全局存储的真正缓存,用于过滤对存储器控制器的请求,减少对显存的访问,节约显存带宽,执行原子内存操作的时候要比以前的快 5~20 倍,从而提高高维度数据流处理的性能.

在实现技术上,可采用 GPU-CUDA 中如下的技术:

① 使用 cudaMallocHost 分配主机端存储器,可以获得更大的带宽;

② 一次缓存较多的数据,然后一并传输,可以获得较高的实际带宽;

③ 需要将结果显示到屏幕上时,直接使用与图形学 API 互操作功能完成,避免将数据返回;

④ 使用流和异步隐藏与主机的通信时间;

⑤ 使用 zero-copy 技术和 Write-Combined memory 提高可用带宽.

(4) 数据流的流动单位

本文在计算框架模型中,首先尽量减少数据交换次数,即提高每次交换到 GPU 设备上的数据块大小.GPU 进行通用计算的时候,每个内核函数是以数据块 block 为单位执行的,各 block 是并行执行的,block 间无法通信也没有执行顺序.这就要求 GPU 处理数据流时,数据交换是以一定粒度的数据块为单位的.

其次,本文使用 CUDA 构建计算模型,采用了 SIMT(single instruction,multiple thread,单指令多线程)执行模式,SIMT 是对 SIMD(single instruction,multiple data,单指令多数据)的一种改进.SIMD 中,向量宽度受到硬件限制,是显示的、固定的,数据打包成向量才能被处理.在 SIMT 中,执行数据的宽度将作为硬件细节被隐藏起来,硬件可以自动地适用不同的执行宽度.另外,SIMT 中每个线程的寄存器都是私有的,线程之间只能通过共享存储器和同步机制进行通信,所以数据流动时都体现了各自的特点,都是相互独立的.在典型的数据流研究领域中,基于基本窗口的滑动窗口方法和基于批处理的数据流聚类算法都是以一块数据为单位进行计算的.

所以,本文研究的数据流并行计算要符合以上两点,以一定粒度的数据块为单位进行数据交换和数据处理,在数据流相关性分析上设计了合理的数据块集合单元.

2.2.3 高维数据流 GPU 处理的模型操作

界标窗口和滑动窗口是数据流处理中比较常用的窗口模型,本文的计算模型是基于滑动窗口进行研究的.最一般意义下的滑动窗口是,每到达一个新的数据元素,窗口跟着向前滑动一个位置.在这里,我们采用一种称为基于基本窗口的滚动窗口类型的的滑动窗口模型,即基于基本窗口的滚动窗口:假设滑动窗口中存放 n 个数据,每个数据的属性维数为 p,按照时间次序将窗口划分为 w 个等宽子窗口,而每个子窗口的大小是可以存放下一个数据的 p 个多维属性值的.如果每当包含 p 个属性值的一个数据元素完全到达时,滑动窗口才向前滑动一个基本窗口的位置,这样的窗口被称为滚动窗口.在高维数据流模式中,一个数据如果有 p 个属性,则一个基本窗口中存放这 p 维的数据,当有新数据的属性值完全到来时,窗口向前滑动,如图 2 所示.将大小为 W 的窗口按照时间次序划分成 k 个等宽的子窗口,称为基本窗口.每个基本窗口包含 W/k 个元素,且由一个子概要数据结构表示该基本窗口的特征.

在图 2 中,双箭头所指示的区域为滑动窗口大小,每个块表示一个基本窗口,窗口中的条纹表示不同维度的属性.

滑动窗口内的数据大小可以发生变化,也可以固定不不变.在这两种窗口中都可以执行插入新元素或从老窗口中删除元素,但它们在处理时有所区别.在大小固定的滑动窗口中,必须加以限制,成对执行操作,否则会破坏窗口;在大小变化的窗口中则没有这样的限制.

使用GPU技术对高维数据流进行处理过程中,在GPU上进行概要数据结构维护时,是以一个个基本窗口的数据子集作为输入的,从而形成许多个概要数据结构的子集.这种生成算法又是以基本滑动窗口为模型,通过不同的 GPU 线程处理最近流入的数据.

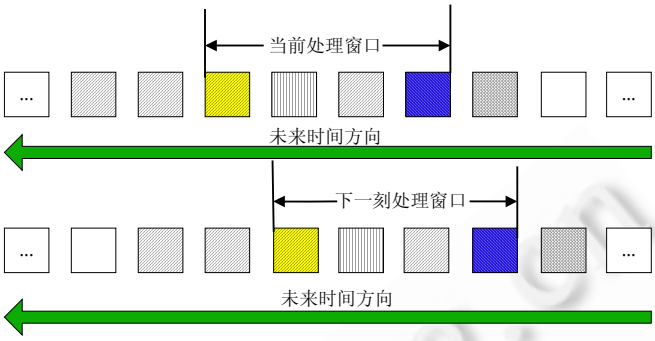

Fig.2 Scrolling window chart

图 2 滚动窗口示意图

2.3 具体可行的GPU协处理高维数据流的框架

基于上面介绍的六层架构模型,具体分析一下每层具体的实施方案,着眼于以更高效的方式利用图形处理器对多条高维数据流进行并行计算,从而提高数据流的实时处理能力.本文的图形处理器数据流并行处理框架在统一计算设备架构(CUDA)上实现.框架如图 3 所示.

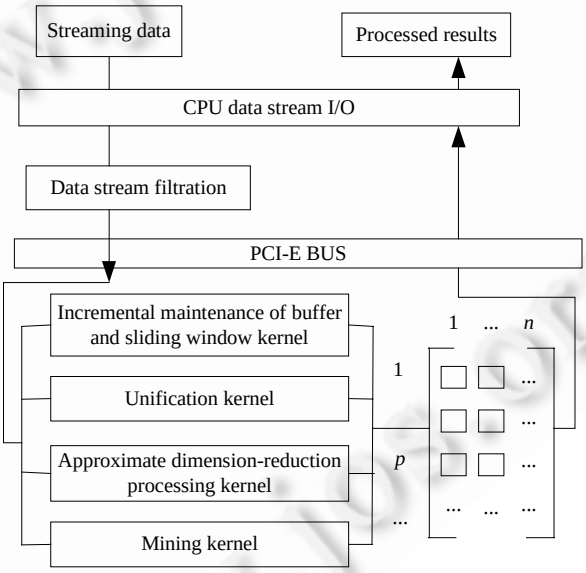

Fig.3 Specific and feasible architecture of high dimensions data streams processing based on GPU

图 3 GPU 协处理高维数据流的具体可行框架

在本框架中,增量维护内核特别适合高维流的处理,与高维数据流模式的定义十分吻合,负责生成和增量维护缓冲窗口和滑动窗口摘要信息;统一化内核负责执行调整高维数据流格式等任务;近似降维处理内核是在滑动窗口的模式下进行概要结构生成,完成低阶近似降维处理.挖掘类内核以摘要二维概要矩阵为输入,执行多数据流挖掘算法.由于该框架使用二维概要数据矩阵对多条高维数据流进行密集组织和统一索引,使得该框架同

时适合对多数据流其他的并行计算要求,具有较强的通用性.

下面对本文研究的重点,即多条高维度数据流相关系数并行算法的具体实施方案进行说明.

本文将 n 条高维数据流组织成一个数据流立方,其结构与奥运场馆水立方的结构类似,也可以称为切片面包结构.为了分析 n 条数据流中任意两条中任意维度上的相关性,认为每层代表一条高维数据流,每一层上的分块代表各维度上的属性,若某条数据流的某维属性为空,则数据立方中对应的分块置为 0.在计算时,像单维数据流那样组织成 n 阶矩阵,而实际上只需要计算该矩阵的上三角或者下三角即可,从而造成了存储器空间的浪费.而本数据流立方可以每层代表不同的数据流,所以节省了存储空间.图 4 给出了数据流立方的示例.

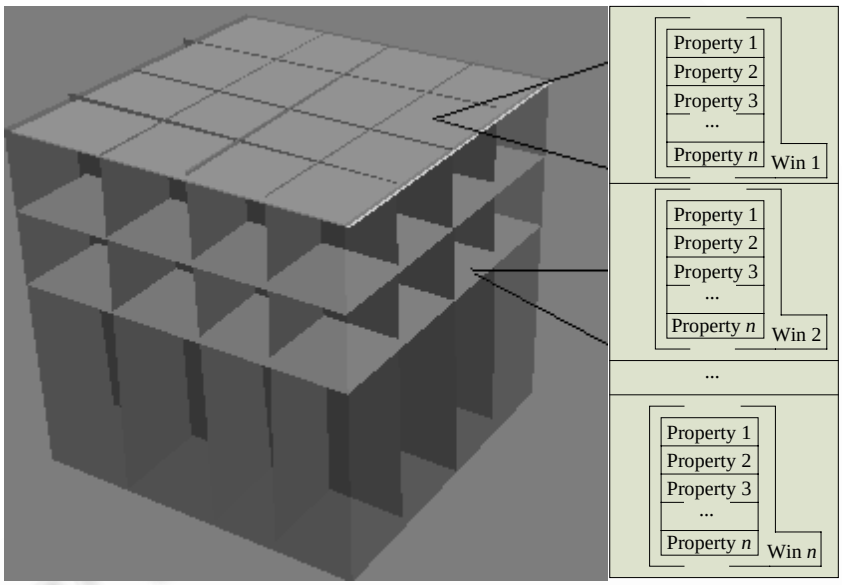

Fig.4 Data streams cube chart

图 4 数据流立方示意图

我们将这个数据流立方抽象到一个 CUDA 计算网格上,并通过线程块索引(blockIdx)访问滑动窗口中的二维概要数据矩阵.每条数据流对应一个窗口,每个窗口上映射为一个二维矩阵,在该矩阵上进行不等概行向兼列向采样,得到标有矩阵行和列的重要测度.例如在图 3 中,利用线程块计算 S_3 和 S_1 的某一维度属性的相关性,通过线程块索引到滑动窗口 3 和滑动窗口 1 上概要矩阵上,进行处理.在 CUDA 计算网格中的每一个线程块中,利用块内的 512 个线程,每次以并行增量更新的方式维护两个数据流样本矩阵的协差阵 S_{21},S_{22} 和各自的方差阵 S_{11},S_{12},然后对高维的矩阵进行列向和行向上的采样实现维数约简,最后并行计算特征值及其特征向量,得到最终结果.如果线程块的行索引小于或等于列索引,则说明该线程块不在下数据流立方内,直接返回.

2.4 使用CUDA的并行计算模型

本文的研究课题是在 CUDA[18]统一计算设备架构上进行的,是一种并行编程模型和软件环境,CUDA 模型将 CPU 作为主机(host),GPU 作为协处理器(co-processor)或设备(device),两者协同工作.CUDA 的核心有 3 个重要抽象概念:线程组层次结构、共享存储器、屏蔽同步.这些抽象提供了细粒度的数据并行化和线程并行化,嵌套于粗粒度的数据并行化和任务并行化之中.

CUDA 的一个内核程序可由多个线程块(block)组成,这些线程块的集合就是一个网格(grid).每个块内可以使用最多 512 个线程,同一个块内的线程可以在没有共享内存 Bank 冲突的情况下高速访问块内共享内存,并可以使用一个内部命令 *_syncthreads*()_进行同步.CUDA 定义了 4 个内置变量 *threadIdx*,*threadDim*,*blockIdx*,*blockDim*,分别表示线程索引、线程维度、块索引、块维度,通过这些内置变量可以索引到网格内具体块的具体

某一个线程.图 5 描述了本课题在 CUDA 模型下的线程块网格.

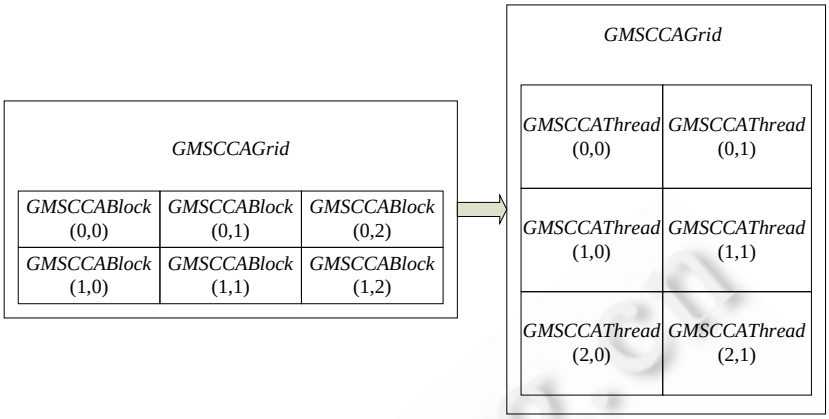

Fig.5 Gird and blocks of CUDA

图 5 CUDA 的线程块网格

GMSCCA(GPU multi-dimensional stream canonical correlation analysis)是指高维数据流相关性分析的并行算法.一个 Kernel 函数中存在两个层次的并行,即 *GMSCCAGrid* 中的 *GMSCCABlock* 间并行和 *GMSCCABlock* 中的 *GMSCCAthread* 间并行,实现两层并行模型.

3 高维数据流相关性分析的并行算法设计

3.1 算法相关基础及定义

定义 3.1. 皮尔逊积差系数(product-moment correlation coefficient)及其数学特征.

在概率论和统计学中,相关系数(correlation,或称关联系数)显示两个随机变量之间线性关系的强度和方向.在统计学中,相关的意义是用来衡量两个变量相对于其相互独立的距离.在这个广义的定义下,有许多根据数据特点而定义的用来衡量数据相关的系数.对于不同数据特点,可以使用不同的系数.最常用的是皮尔逊积差相关系数.其定义是两个变量协方差除以两个变量的标准差.本文采用皮尔逊积差相关系数进行研究.

定义 3.2. 统计意义上两组数据的相关性[21].设 x_i 为 n 维($n\geqslant 1$)元素,$F(x_i),G(x_i)$为 n 维元素映射的数据函数:

$$F'(x_i)=F(x_i)-\frac{1}{N}\sum_{j=1}^{N}F(x_j),\ \xi_F=\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}F'^2(x_i)}.$$

那么,$D(\tau)=\frac{1}{\xi_F\zeta_G}\left[\frac{1}{N-1}\sum_{i=1}^{N}F'(x_i)G'(x_i+\tau)\right]$可以帮助应用获知两组 n 维变化数据的相关性.

从柯西-施瓦茨不等式可知,相关系数的绝对值不超过 1.进行实时比较时,即当τ=0 时,当两个变量的线性关系增强时,相关系数趋于 1 或–1;当一个变量增加而另一变量也增加时,相关系数大于 0;当一个变量的增加而另一变量减少时,相关系数小于 0;当两个变量独立时,相关系数为 0.但反之并不成立.如果两个函数 F=G 时,通过 C(0)的值可以分析单条数据流自身的属相相关性及其变化的周期性.

模式定义 1. 高维数据流模式[20].

本文是在数据流十字转门模型的基础上,定义适合高维数据流分析的滑动数据流窗口模式.高维数据流 $\alpha_1\ldots\alpha_i\ldots$可以定义为高维信号 X 到实数集上的一个映射:$X[1\ldots N]\rightarrow R^p$,即将一条高维数据流的 n 个数据映射到一个列向量里.每个α_i是对不同时刻的数据流中某一个属于值 $X[j]$的更新值,$\alpha_i=(j,\Delta_i)$表示一个更新元组,其含义是 $X_i[j]=X_{i-1}[j]+\Delta_i$,$\Delta_i$可能为正也可能为负,表示在时刻 t 的 p 维更新向量,完全符合十字转门模型.向量Δ_i只能读取 1 次,按照时间戳 i 的增加流入.高维数据流的滑动窗口模式定义为包含最近 n 项元素的序列$\alpha_{t-n-1}\ldots\alpha_i$.可将

数据流视为矩阵,这里只是概念意义上的矩阵,不需要真正物化这个矩阵.

令 X_{n*p} 表示时间窗为 n 的具有 p 个属性的高维流矩阵,Y_{n*q} 表示时间窗为 n 的具有 q 个属性的高维流矩阵(不失一般性,设 $p\leqslant q$),$X_{(i,t)}$表示 X 中第 t 个时刻的第 i 个属性值;$X(i)$表示 X 的第 i 行;$Y(i)$表示 X 的第 t 列.若在时刻 t,对 X 和 Y 两个元组元组没有限制,那么两条数据流是异步的.若在时刻 t,元组 X 有 p 个值,元组 Y 有 q 个值,每个值对应于一个属性值的流入,那么我们称这种情况下的 X,Y 是同步的.元组按时间 t 有序,如果高维数据流在某时刻未接收到元组,则以稀疏数据流矩阵,即全 0 的元组代替.

3.2 算法思想与设计

高维数据流 X 与 Y 之间典型相关性分析的基本思路为:根据矩阵论[22],以最大限度地提取 X 与 Y 之间相关关系的主要特征为准则,从 X 中提取组合变量 U,从 Y 中提取组合变量 V,如公式(1)所示.

$$U_{n+1}=X_{p+n}A_{n+1},V_{n+1}=Y_{q+n}A_{n+1} \tag{1}$$

其中,A,B 为线性变换,又称为空间特征向量.按公式(1)把具有较多个变量的数据流矩阵 X 与 Y 之间的相关化为较少组合变量 U 与 V 之间的相关.

在实际的数据流应用中,一般使用样本进行估计.使用长度为 n 的滑动时间窗口中的样本进行典型相关性分析的过程如下:

设 $\begin{bmatrix}X_{(i)}\\Y_{(i)}\end{bmatrix}(i=1,2,\ldots,n)$ 为来自总体 $\begin{bmatrix}X\\Y\end{bmatrix}$ 的一个样本,其中,$X_i=(X_{1i},X_{2i},\ldots,X_{pi})^T,Y_i=(Y_{1i},Y_{2i},\ldots,Y_{qi})^T(i=1,2,...,n)$,计算 $X_{(i)}$的方差阵 S_{11},$Y_{(i)}$的方差阵 S_{22},协方差阵 S_{21},则 X,Y 的第 k 对典型变量为 $U_k=e_k^TS_{11}^{-\frac{1}{2}}X_{(i)},V_k=f_k^TS_{22}^{-\frac{1}{2}}Y_{(i)}$,其典型相关系数为 $\rho_{U_k,V_k}=\rho_k$ (k=1,2,…,p).其中,

$$S_{11}=\frac{1}{n-1}\sum_{i=1}^{n}(X_{(i)}-\bar{X})(X_{(i)}-\bar{X})^T,\ S_{22}=\frac{1}{n-1}\sum_{i=1}^{n}(Y_{(i)}-\bar{Y})(Y_{(i)}-\bar{Y})^T,$$

$$S_{21}=\frac{1}{n-1}\sum_{i=1}^{n}(X_{(i)}-\bar{X})(Y_{(i)}-\bar{Y})^T=S_{21}^T,\ \bar{X}=\frac{1}{n}\sum_{i=1}^{n}X_{(i)},\ \bar{Y}=\frac{1}{n}\sum_{i=1}^{n}Y_{(i)}.$$

$\rho_1^2\geqslant\rho_2^2\geqslant\ldots\geqslant\rho_p^2$ 为 p 阶矩阵 $M=S_{11}^{-\frac{1}{2}}S_{12}S_{22}^{-1}S_{21}S_{11}^{-\frac{1}{2}}$ 的特征值,$e_1,e_2,\ldots,e_p$ 为相应的正交单位化特征向量,$f_1,f_2,\ldots,f_p$ 为 q 阶矩阵 $N=S_{22}^{-\frac{1}{2}}S_{21}S_{11}^{-1}S_{12}S_{22}^{-\frac{1}{2}}$ 的对应于前 p 个最大特征值(按由大到小的次序排序)的正交单位化特征向量.

3.3 基于CUDA的并行高维数据流相关性算法分析

(1) 算法流程

基于前面的 GPU-n 维数据流处理通用模型,本文结合文献[21]的高维数据流处理的方法给出基于 CUDA 架构的高维数据流相关性分析的并行算法 GMSCCA.

本文中的高维数据流的相关性分析采用的算法是一种基于近似技术在 GPU 上进行的快速高维数据流相关性并行分析算法 GMSCCA,即以增量更新的方式维护两个数据流样本矩阵的协差阵 S_{21},S_{22} 和各自的方差阵 S_{11},S_{12};然后对高维的乘积矩阵 M 进行列向和行向上的采样实现维数约简,降低生成典型相关系数的代价.算法流程如图 6 所示.其中,$M=S_{11}^{-\frac{1}{2}}S_{12}S_{22}^{-1}S_{21}S_{11}^{-\frac{1}{2}}$,$W$ 是采样后的矩阵.

Step 1. 启动 CUDA,使用多卡时加上设备号,使用 *cudaSetDevice*()设置 GPU 设备.

Step 2. 为输入数据分配主机内存空间以及缓冲区.

Step 3. 从流数据源获取输入数据进行初始化.

Step 4. 为 GPU 分配显存以及缓冲区,用于存放输入数据.

Step 5. 将内存中的输入数据交换到显存.

Step 6. 为 GPU 分配显存以及缓冲区,用于存放输出结果.

Step 7. 根据滑动窗口数据流模式,同步流的当前输入到达时,时间窗前滚一个时刻,接受一个新的元组.

Step 8. 启动设备内核,执行高维数据流相关性分析的并行算法子算法

GPU_Producing_Matrix_of_Variance_and_Covariance_Matrix().

Step 9. 计算相邻时刻元组更新$\Delta(j)$的累加和 G 是否大于跳跃因子δ(精度系数):若大于,则重新计算 CCA;否则,不需要重新计算.

Step 10. 生成概要数据结构,实施不等概行采样兼列采样对矩阵进行维数约简.执行子算法:

GPU_unequal_probability_sampling_in_Row_And_Column_kernel().

Step 11. 并行计算典型特征值和典型特征向量.

Step 12. 为 CPU 分配内存,用于存放 GPU 传回来的输出数据.

Step 13. 将显存中的结果回读到内存.

Step 14. 使用 CPU 对数据进行其他处理.

Step 15. 释放内存和显存空间.

Step 16. 退出 CUDA.

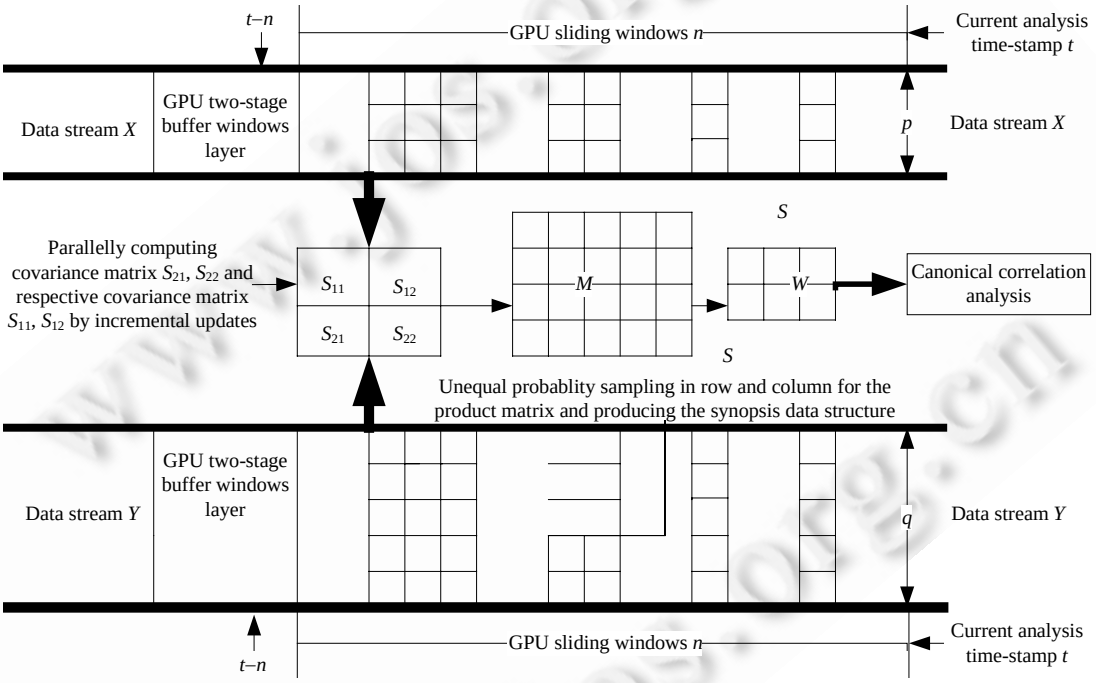

Fig.6 Algorithm analysis process diagram

图 6 算法分析流程图

(2) 并行算法 GMSCCA 流程的正确性论证

① 宏观粗粒度分析

高维数据流的维数通常很高,对其进行高性能运算频繁计算矩阵的乘、转置等是避免不了的,这无疑是非常耗时的.由于在实际应用中数据流量是连续庞大的,所以物化所有的流数据是不实际的,而且对用户实时地对高维数据流进行统计性的连续查询和分析提出了更高的要求和挑战.由于计算能力受限,牺牲部分准确性来换取速度的解决方案是解决此类问题的关键,结合 GPU 的高性能运算能力是一个很好的办法.

② 微观细粒度分析

本文给出的并行方法是结合了文献[21]的高维数据流的处理方法,所以算法的思想和流程得到了保证,本文是在此基础之上进行 GPU 的并行设计;算法中对数据流的处理采用的是基于基本窗口的滚动窗口类型的滑动窗口模型,是一种适合于高维数据流的机制;并行增量的算法、方差以及协方差的并行计算都是 GPU 应用中比较成熟的方法,所以在滑动窗口数据流模式下,并行增量生成 X,Y 各自的方差阵及 X 与 Y 协差阵的正确性可以得到保证;采用不等概行采样兼列采样进行概要数据结构构造的方法中涉及矩阵中行和列的重要程度的测度,是由第 2 节提出的高维数据流 Froenius 范数和 2-范数的定义来规范的.

3.3.1 设备内存上的滑动窗口更新、删除

对于滑动窗口数据流模式,当同步流的当前输入 $T=(t,\Delta_x,\Delta_y)$到达时,$X_{(t-n+1)}=X_{(t-n+2)},\ldots,X_{(t-1)}=X_{(t)},X_{(t)}=X_{(new)}$,时间窗前滚一个时刻,接受新的元组 $X_{(new)}$.n 步的上述赋值操作归结为 $X=X+\Delta_x$,同理有 $Y=Y+\Delta_y$,一般情况下可认为 $\Delta_x=X_{(new)}-X_{(t-n+1)},\Delta_y=Y_{(new)}-Y_{(t-n+1)}$.同步高维数据流增量归一化算法如下:

3.3.2 数据流矩阵的循环队列实现方式

更新矩阵时(即时间窗前滚),用新元组替换掉最旧的元组,只需简单地使偏移索引指向下一个相邻的窗口.

3.3.3 增量生成算法

在滑动窗口数据流模式下,并行增量生成 X,Y 各自的方差阵及 X 与 Y 协差阵.

算法. *GPU_Producing_Matrix_of_Variance_and_Covariance_Matrix*(X,Y,Δ_x,Δ_y,n).

输入:$X\in R^{p*n},Y\in R^{q*n},X,Y$ 的更新值Δ_x,Δ_y,滑动窗口 n 的长度.

输出:$S_{11}\in R^{p*p},S_{12}\in R^{p*q},S_{21}\in R^{q*p},S_{22}\in R^{q*q}$.

Step 1. 设备内存上的滑动窗口更新、删除

Step 2. or all non-zero items in column t of $A,B\{j|A(j,t)\neq 0,B(j,t)\neq 0\}$ do begin

Step 3. if ($j\neq i$) begin

Step 4. 并行地增量生成 X,Y 各自的方差阵及 X 与 Y 协差阵

$$S_{11(i,j)}+=\Delta_a A_{(t)};S_{22(i,j)}+=\Delta_b B_{(t)};S_{12(i,j)}+=\Delta_a B_{(t)}+\Delta_b A_{(t)}$$

Step 5. end if

Step 6. if ($j=i$) begin

Step 7. 并行地增量生成 X,Y 各自的方差阵及 X 与 Y 协差阵

$$S_{11(i,j)}+=2\Delta_a A_{(t)}+(\Delta_a)^2;S_{22(i,j)}+=2\Delta_b B_{(t)}+(\Delta_b)^2;S_{12(i,j)}+=\Delta_a B_{(t)}+\Delta_b A_{(t)}+\Delta_{ab}$$

Step 8. end if

Step 9. $A_{(t)}+=\Delta_a;B_{(t)}+=\Delta_b$;

Step 10. end do

图 7 为子算法并行增量的思想及详细过程.

如图 7 所示的增量算法对每个表示数据流的大数组进行划分,让每个 block 对一个子块进行 scan,scan 就是对一个数组求 all-prefix-sums(前缀和)的过程,在每个 block 的计算完成后,要将该子块的总和写入辅助数组(auxiliary array);接下来对这个辅助数组进行 scan,得到一个块增量数组 block sums,其中的每个元素代表一个 block 相对于上一个块的增量.

假设输入数组 *INPUT*[ ]中有 N 个元素,由每个 block 负责处理 B 个元素,这样就需要 N/B 个 block,每个 block 有 $B/2$ 个线程(每个线程处理 2 个元素).首先,由各 block 独立计算 block 中的 scan,并将数组输出到 *OUTPUT*[ ]中.但此时得到的还不是最终结果,还需要另外一个 kernel 将上一步中每块的最后一个元素写入 *SUMS*[ ],再对 *SUMS*[ ]进行 scan 操作,求出每一块相对上一块的增量,并写入 *INCR*[ ].最后,由第 3 个 kernel 函数为 *OUTPUT*[ ]中的每个 block 加上对应的偏移量(uniform add)*INCR*[j],求出最终结果.

对于数组元素个数不是 2 的幂次方的情况,将数组分为两部分来处理:第 1 部分拥有 $floor(1.0\times n/B)\times B$ 个元素,而其余元素(记为 $rest,rest<B$)存放在第 2 部分.对第 1 部分的处理按照图 7 所示处理,对第 2 部分则需要另行

处理(用一个 block 进行处理),这时分配的共享存储器空间的大小应该是大于 *rest* 的最小的 2 的幂次方.

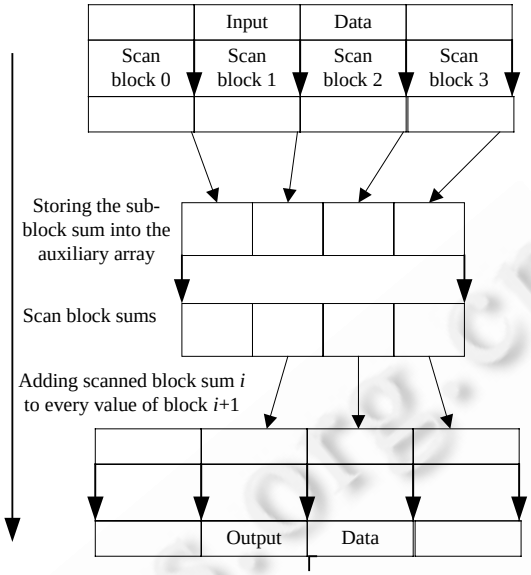

Fig.7 Parallel incremental sub-algorithm analysis diagram

图 7 并行增量子算法思想

scan 算法的伪码

Step 1. 动态分配共享存储器空间 *extern_shared_float temp*[ ];

Step 2. 全局存储器向共享存储器的复制,复制时右移一个位置,最左边填 0

Step 3. for (int *offset*=1;*offset*<*n*;*offset**=2)

{

pout=1−*pout*;

pin=1−*pout*;

_syncthreads();

temp[*pout*×*n*+*thid*]=*temp*[*pin*×*n*+*thid*];

if (*thid*>=*offset*)

temp[*pout*×*n*+*thid*]+=*temp*[*pin*×*n*+*thid*−*offset*];

}

Step 4. *_syncthreads*();

Step 5. *g_odata*[*thid*]=*temp*[*pout*×*n*+*thid*]; //数据拷回

3.3.4 方差及协方差的并行算法

对于高维数据流 *X*,*Y* 的方差阵,*X* 与 *Y* 之间的协方差阵也是非常耗时的计算.以 *X* 的方差阵为例对并行计算方差阵和协方差阵进行算法说明.

(1) 利用 GPU 归约求和计算 $\bar{X}$.

算法.

Input: *T*g_idata*, *T*g_odata*.

Output: $\bar{X}$.

//加载共享存储器

```
Step 1.   SharedMemory⟨T⟩ smem;
Step 2.   T *sdata=smem.getPointer();
          //设置线程索引
Step 3.   unsigned int tid=threadIdx.x;
          unsigned int i=blockIdx.x*blockDim.x+threadIdx.x;
Step 4.     sdata[tid]=g_idata[i];
          _syncthreads();
          //在共享存储器里进行归约
Step 5.   for (unsigned int s=1; s<blockDim.x; s*=2)
          {
          if ((tid %(2*s))==0) {
               sdata[tid]+=sdata[tid+s];
          }
          _syncthreads();
        }
          //将 block 的计算结果写回全局存储器
Step 6.   if (tid==0) g_odata[blockIdx.x]=sdata[0];
Step 7. end;
```

(2) 按照实时性的要求将分成如图 8 所示的线程层次.一共有 $N\times P$ 个线程分配时:

- *dim*3 *dimBlock*(512);
- *dim*3 *dimGrid*((*N*+*dimBlock.x*–1)/*dimBlock.x*, (*N*+*dimBlock.y*–1)/*dimBlock.y*).

每个线程负责处理数据流矩阵中的一个元素:

- if (*threadIdx.x*=0),该索引下对应的元素减去 $\bar{X}$ 列向量中第 1 个元素;
- if (*threadIdx.x*=1),该索引下对应的元素减去 $\bar{X}$ 列向量中第 2 个元素.

从而得到 $X(i)-\bar{X}$ 的左右元素组成的矩阵,然后对该矩阵进行转置处理.在本算法中,通过引入 shared memory 实现线程间通信,就可以让一个 half-warp 中的线程按照合并访问方式输入一行数据后,再以合并访问方式输出数据.

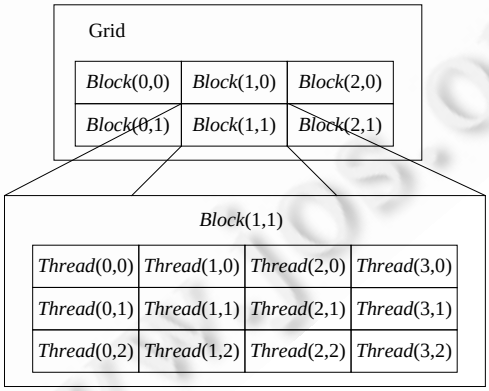

Fig.8 Thread level

图 8 线程层次

采用 CUDA 的两层并行,用合并访问方式将数据从显存读入共享存储器中.经过同步后,每个线程与和它关于对角线对称的线程交换操作的数据,再按照合并访问方式将结果写到显存中.在同一个block中实现需要进行

数据交换和通信的细粒度并行,而在各个block间实现不需要进行数据交换的粗粒度并行.对全局内存器的所有读写均是合并读写,共享存储器大小是(*BLOCK_DIM*+1)×*BLOCK_DIM*,保证 half-warp 按列访问数组的时候不发生 bank conflict.

(3) 计算矩阵及其转置矩阵的乘法

在这个实现中,每个 block 负责结果数组 *C* 中的一个方块 Csub 的计算,其中每个线程负责计算 Csub 的一个元素.分块计算如图 9 所示,*A* 的 sub-matrix 维度是(*A.width*,*block_size*),与 Csub 有相同的行索引;*B* 的 sub-matrix 维度是(*block_size*,*A.width*),与 Csub 有相同的列索引.为了适应设备资源,两个矩形矩阵被分成大小为 *block_size* 的正方形矩阵,Csub 的计算就是这些小正方形矩阵的乘积之和.这些乘积计算如下:首先从 global memory 加载两个相应的正方形矩阵到 shared memory,一个线程负责加载一个元素;接下来,每个线程负责计算乘积中的一个元素,并且除以 *n*–1.每个线程累积每个这样乘积的结果到一个寄存器中,一旦结束就将结果写回 global memory.算法思想如图 9 所示.

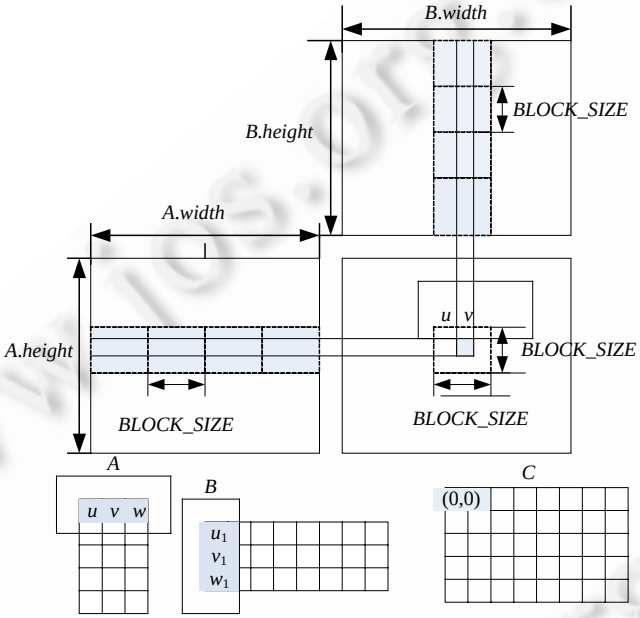

Fig.9 Matrix multiplication algorithm

图 9 矩阵乘法算法

3.3.5 概要数据结构的构造

高维数据流处理的关键在于设计一个远小于数据集规模的结构,从而可以在内存中高效处理数据,需要保存长时间跨度的滑动窗口或者数据流到来的高速,解决这个问题的一种方法是对滑动窗口内的数据进行不等概采样.

真实数据流通常具有大量属性维(即 *p*,*q* 的值很大),计算复杂度很高,而且流矩阵中存在稀疏矩阵以及在每个流矩阵中只是少数几个属性维起决定作用.为了实现实时的相关性分析,需要使用少量属性维形成的简单矩阵代替原来的数据流矩阵.因此,本文采用在欧几里得空间中不等概行采样兼列采样的约简近似技术来提高效率.这样得到的采样阵,我们称作高维数据流的概要矩阵.

根据矩阵论[22],Froenius 范数和 2-范数作为矩阵中行和列的重要程度的测度,这样在采样时可以为每一行(列)赋上反映重要程度的“标志值”.

为了在确定的精度内保证约简的质量,本文采用 Johnson-Lindenstrauss 引理[23],这是对高维矩阵进行维数约简的有效技术.引理表明,如果利用矩阵对一个 *n* 维向量进行变换,根据高斯分布选取矩阵的元素,则可以高概

率在 $s(s<n)$维的结果空间中保持向量间的相对距离.

概要数据结构生成的理论依据如下:

先对 C 实施行采样兼列采样,根据 JL 引理确定采样行(或列)的数目,并结合随机高斯概率和每一行(或列)的测度选取行(或列).采样的概率由文献[23]中的引理确定:

基于以上分析给出并行算法如下:

GPU¬_unequal_probability_sampling_in_Row_And_Column_kernel(*C*,*p*,α,ε,*W*)

//*p* 为矩阵的维度

Step 1. int *k*=0, *r*=o; //确定采样的行数与列数

Step 2. for (int *i*=0; *i*<*p*; *i*++)

Step 3. *CUDA_SAFE_CALL* 生成介于 0~1 之间的随机数 G_i 的内核函数 *random_kernel*

Step 4. 申请 *p* 个线程,在每个线程中并行计算 $\alpha\|C^{(i)}\|_2^2/\|C\|_F^2$,并且进行以下运算

Step 5. if ($G_i<\alpha\|C^{(i)}\|_2^2/\|C\|_F^2$)

Step 6. then $D^{(k)}=C^{(i)}$;并行增量生成;$\|D\|_F^2=\|D\|_F^2+C^{(i)}$ //$\|D\|_F^2$ 的初值为 0

Step 7. if (*k*<*s*) then 退出 for 循环;

end for

Step 8. for (*j*=0; *j*<*p*; *j*++) begin

Step 9. *CUDA_SAFE_CALL*)生成介于 0~1 之间的随机数 G_i 的内核函数 *random_kernel*

Step 10. 申请 *p* 个线程,在每个线程中并行计算 $\frac{\alpha}{2}\|D_{(j)}\|_2^2/\|D\|_F^2$,并且进行以下运算

Step 11. if ($G_i<\frac{\alpha}{2}\|D_{(j)}\|_2^2/\|D\|_F^2$)

Step 12. then $W_{(r)}=D_{(j)}$

Step 13. if (*k*<*s*) then 退出 for 循环;

Step 14. end for

Step 15. 将结果 *W* 交换到 global memory

其中,使用 GPU 技术生成随机数算法在这里进行简单说明.

Step 1. //设置随机数种子

Step 2. *atomicExch*(&*next*,*seed*);

Step 3. //分配线程索引

unsigned int tid=threadIdx.x;

*unsigned int tid_in_block=threadIdx.x+threadIdx.y*blockDim.x*;

*unsigned int tid_in_grid_x=blockDim.x*blockIdx.x+threadIdx.x*;

*unsigned int tid_in_grid_y=blockDim.y*blockIdx.y+threadIdx.y*;

*unsigned int tid_in_grid=tid_in_grid_y*blockDim.x*gridDim.x+tid_in_grid_x*;

//计算线程的线性编号

Step 4. *unsigned int temp*=(*next*+*tid*+*tid_in_grid*)*1103515245+*tid_in_block*;

Step 5. *atomicExch*(&*next*,*temp*);

//采用线性同余的方式

Step 6. *temp*=((*next**314159629)+453806245)%(*int*)(*powf*(2.0,31));

Step 7. *atomicExch*(&*next*,*temp*);

Step 8. *atomicAdd*(&*next*,12345);

Step 9. *unsigned int result*=(*unsigned int*)(*next*/65536)%32768;

Step 10. return (57434*tid+tid_in_grid*145133+result*9837+tid_in_block*545346)%32768;

限于篇幅,在 GPU 上生成满足 $N(0,1)$分布的随机数算法略.同时,矩阵 C 的列采样兼列采样矩阵 W 与矩阵 C 在 Froenius 范数的意义上是好的近似.证明略.

3.3.6 特征值以及特征向量的计算

在计算 CCA 时,采用跳跃式的计算.因为若相邻两个元组(或元组的更新)到达的时间间隔较长,在每次流更新时重新计算 CCA.但是在一般情况下,在某个特定时刻,计算 CCA 之后并没有明显的变化,那么就没有必要频繁地重新计算 CCA.其思想如下:

若在时刻 t_1 计算 CCA,此时设$\Delta(i)$表示$\begin{bmatrix}\Delta_{(i)}^X\\ \Delta_{(i)}^Y\end{bmatrix}$,记 G 为从时刻 t_1 到时刻 t 之间$\Delta(i)$的累加和,设定阈值 M:

- 如果 $G\geqslant M$,则当前流矩阵仍然是合理的,在 t_1 和 t 之间不必重新计算 CCA;
- 如果 $G\leqslant M$,则重新计算 CCA.

这样,阈值 G 的确定由下面的方法进行确定:

设 $Z=\begin{bmatrix}X\\ Y\end{bmatrix}$的最大典型特征值为$\lambda_1$,由于$\|Z-Z_1\|_F$可以作为$\lambda_1$的估计,衡量 G 变化的阈值可以由λ_1来确定.如果要求保留多个典型特征值,则 G 应当与$\delta(\lambda_1+\lambda_2+\ldots+\lambda_k)$相比较.计算矩阵特征值以及特征向量也是非常耗时的,也需要并行处理,限于篇幅,这里不作说明.

在本文中求矩阵 W 的特征值和特征向量的意义:矩阵 W 的特征值表示相关性的强度,如果一个特征值比其他特征值大许多,那么其对应的特征向量就代表在前 k 个最大相关向量张成子空间中更强的线性相关.

4 实验结果及分析

本文的目的在于使用图形处理器提高高维数据流的实时处理能力,实现 GMSCCA 算法和纯 CPU 版本高维数据流典型相关性分析算法,用于结果验证和带宽对比.对比实验使用 Matlab 生成的、选择随机生成含 1 000 万个数据的数据集作为种子,在内存中反复生成满足特定分布的数据集供实验使用.我们构造和选取不同相关度的数据集 X,Y 进行测试:

数据集:

(1) 线性数据集 DS_L:每个属性值取自于线性数据,然后叠加符正态分布 $N(10,10)$的样本,期望得到大于 0.5 的相关系数.

(2) 真实气象数据集 *DSw*:涉及天气变化的 5 年历史数据,分析两条气象指数的相关性,用于指导天气预报.事先无法预计其相关系数的大小,由实际情况决定.

测试环境:根据数据的规模,选取合适的 CPU 和 GPU.这里的测试采用 4 核 Intel i7 920 CPU 和 Tesla C1060 的 GPU,详细参数见表 1.

Table 1 Test environment

表 1 测试环境

CPU	GPU	运行环境
Intel i7 Quad CPU 920 2.66GHz, 4MB L2, DDR3 1 033MHz 8GB	GDDR3 4GB, 30SM, 8SPs/SM, 16 384 registers/SM, Max. 1 024 threads/SM, 16KB shared memory per SM, 64KB const./text. memory	Fedora Linux 10, CUDA 3.0

实验 1. GMSCCA 与 CPU 版的高维数据流典型相关性分析算法运行时间的对比.

假设有足够大的缓存空间可以容纳窗口中所有的元组,图 10 中显示了在 p=256,q=512,n=2048,δ=1.2,α=1.1 条件下,CPU 串行与 GPU 并行处理两种算法在 DS_L 数据集上每元组平均处理时间的比较.

图中同时给出了选择不同的行(列)数 s 时对运行时间的影响,s 对应了不同的近似精度参数ε(ε=0.5,ε=0.4,

ε=0.3).显而易见,GMSCCA 算法极大地降低了运行时间,CPU 串行算法的运行时间是 GPU 并行算法的 36 倍左右,所以并行算法有效地提高了高维数据流挖掘算法的实时性.在实际应用中,数据流矩阵的最小维数大于 128 并且参数ε大于 0.5 时对 C 进行采样才有意义.

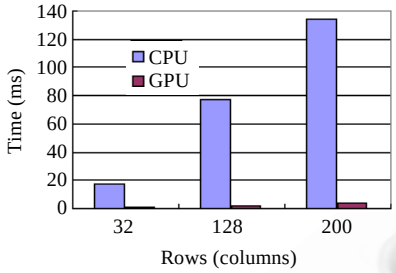

Fig.10 Comparison to linear data set DS_L with noise

图 10 在带有噪声的线性数据集 DS_L 的比较

实验 2. 高维数据流滑动窗口长度 n 的变化对近似精度的影响.

由于本文中的并行算法涉及到了不等概采样,因此数据集潜在的相关性以及它的数据分部情况对于算法的性能有至关重要的影响.首先在 p=102,q=102,δ=1.2,ε=0.14,n={204,408,816,1632,3264}的条件下测试该并行方法计算不同数据集的相关系数的近似精度与滑动窗口长度 n 的关系.测试结果如图 11 所示,DS_L 的相关系数近似精度最高;DSw 呈现出一种比较强相关性,相关系数的近似精度次之.总体上,不同的数据集的平均近似精度保持在 0.88 左右,性能稳定而且良好,能够满足大多数数据流的挖掘应用.

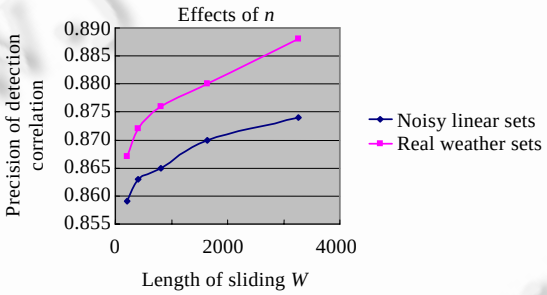

Fig.11 Impact of change about length of sliding window on the approximation accuracy

图 11 滑动窗口长度的变化对近似精度的影响

由实验结果可知,如果高维数据流滑动窗口的大小 n 发生变化,而概要数据结构的规模没有发生变化,则高维数据流每个元组的平均时间变化也并不明显.n 的变化只影响概要矩阵的更新代价,对其他步骤的速度影响不大.这就证明了 GMSCCA 算法对每个元组处理的时间相对恒定,与窗口的大小 n 无关,适合处理无限的高维流数据.

5 总结和展望

本文应用矩阵论和统计理论进行非规则流中高维度数据流的典型相关性处理,创新性地提出了基于 GPU 的高维数据流处理模型及实施架构,并在 GPU 的 CUDA 平台上进行仿真实验.实验结果表明,该算法在处理速度上可以提高 30 倍左右,而且可以在计算资源受限情况下快速、精确地分析高维数据流之间的相关性,并可以在近似精度和性能之间折中.同时,该算法可以应用于更为广泛的多个高维数据流之间的相关性分析,对气象预报、生命与医药信息挖掘、战场传感网络、水资源等实时的趋势分析应用具有重要意义.

但是,鉴于 GPU 并行处理数据流的 I/O 开销很大,将对整个处理过程的加速产生影响,这是今后有待进一步研究和提高的方面.同时,后续研究工作也将集中于非规则流中其他数据流的实时并行处理算法的研究.

References:

[1] Sakurai Y, Papadimitriou S, Faloutsos C. BRAID: Stream mining through group lag correlations. In: Proc. of the 2005 ACM SIGMOD Int'l Conf. on Management of Data. 2005. 599−610. [doi: 10.1145/1066157.1066226]

[2] Zhu YY, Dennis SS. StatStream: Statistical monitoring of thousands of data streams in real time. In: Proc. of the 28th VLDB Conf. 2002. 358−369.

[3] Papadimitriou S, Sun JM, Faloutsos C. Streaming pattern discovery in multiple time-series. In: Proc. of the 31st VLDB Conf. 2005. 697−708.

[4] Guha S, Gunopulos D, Koudas N. Correlating synchronous and asynchronous data streams. In: Proc. of the 9th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2003. 529−534. [doi: 10.1145/956750.956814]

[5] Yang J. Dynamic clustering of evolving streams with a single pass. In: Proc. of the 19th IEEE Int'l Conf. on Data Engineering (ICDE 2003). 2003. 695−697. [doi: 10.1109/ICDE.2003.1260838]

[6] Dai BR, Huang JW, Yeh MY, Chen MS. Clustering on demand for multiple data streams. In: Proc. of the 4th IEEE Int'l Conf. on Data Mining (ICDM 2004). 2004. 367−370. [doi: 10.1109/ICDM.2004.10060]

[7] Babcock B, Babu S, Datar M, Motwani R, Thomas D. Operator scheduling in data stream systems. The VLDB Journal, 2004,13(4): 333−353. [doi: 10.1007/s00778-004-0132-6]

[8] Wu EH. State of the art and future challenge on general purpose computation by graphics processing unit. Journal of Software, 2004,15(10):1493−1504 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1493.htm

[9] Wang YL, Xu HB, Dong YS, Qian JB, Liu XJ. A correlation analysis algorithm based on low-rank approximation for multiple dimension data streams. ACTA ELECTRONICA SINICA, 2006,34(2):293−300 (in Chinese with English abstract).

[10] Bulut A, Singh AK. A unified frame work for monitoring data streams in real time. In: Proc. of the 21st Int'l Conf. on Data Engineering (ICDE 2005). Tokyo: IEEE Computer Society, 2005. 44−55. [doi: 10.1109/ICDE.2005.13]

[11] Guha S, Gunopulos D, Koudas N. Correlating synchronous and asynchronous data streams. In: Proc. of the 9th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (SIGKDD 2003). Washington: ACM Press, 2003. 529−534. [doi: 10.1145/956750.956814]

[12] Mehmet S. Detecting time correlations in time-series data streams. Technical Report, Intelligent Enterprise Technologies Laboratory, 2004. www.hpl.hp.com/techreports/2004/HPL-2004-103.html

[13] Yi BK, Sidiropoulos ND, Johnson T, Jagadish HV, Faloutsos C, Biliris A. Online data mining for coevolving time sequences. In: Proc. of the 16th Int'l Conf. on Data Engineering (ICDE 2000). San Diego: IEEE Computer Society, 2000. [doi: 10.1109/ICDE.2000.839383]

[14] Wang L, Zhang CY. Computing model of general2purpose computation on GPU. Application Research of Computers, 2009,26(6): 2356−2358 (in Chinese with English abstract).

[15] Shu C, Fu ZL, Tan YC. Fast operation of large-scale high-precision matrix based on GPU. Journal of Computer Applications, 2009, 29(4):1177−1179,1192 (in Chinese with English abstract). [doi: 10.3724/SP.J.1087.2009.01177]

[16] Li JM, Chi ZX, Wan DL. Parallel genetic algorithm based on fine-grained model with GPU-accelerated. Control and Decision, 2008,23(6):697−704 (in Chinese with English abstract).

[17] Cao F, Zhou AY. Fast clustering of data streams using graphics processors. Journal of Software, 2007,18(2):291−302 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/291.htm [doi: 10.1360/jos180291]

[18] Nvidia. NVIDIA CUDA programming guide. 2008. http://developer.download.nvidia.com/compute/cuda/2_0/NVIDIA_CUDA_Programming_Guide_2.0.pdf

[19] Wu N. Key techniques research of high productivity stream architecture [Ph.D. Thesis]. Changsha: Graduate School of National University of Defense Technology, 2009 (in Chinese with English abstract).

[20] Wen M, Wu N, Zhang CY, Li HY, Li L. Research of the abstract stream architecture. Computer Engineering and Science, 2006, 28(7):123−126 (in Chinese with English abstract).

[21] Yang XM, Dong YS, Xu HB, Liu XJ, Qian JB, Wang YL. Online correlation analysis for multiple dimensions data streams. Journal of Computer Research and Development, 2006,43(10):1744−1750 (in Chinese with English abstract). [doi: 10.1360/crad20061011]

[22] Yang M, Liu XZ. Matric Theory. Wuhan: Huazhong University of Science and Technology Press, 2005 (in Chinese).

[23] Johnson WB, Lindenstrauss J. Extensions of lipschitz mapping into Hilbert space. Contemporary Mathematics, 1984,26(5): 189−206.

附中文参考文献:

[8] 吴恩华.图形处理器用于通用计算的技术、现状及其挑战.软件学报,2004,15(10):1493−1504. http://www.jos.org.cn/1000-9825/15/1493.htm

[9] 王永利,徐宏炳,董逸生,钱江波,刘学军.基于低阶近似的多维数据流相关性分析.电子学报,2006,34(2):293−300.

[14] 王磊,张春燕.基于图形处理器的通用计算模式.计算机应用研究,2009,26(6):2356−2358.

[15] 苏畅,付忠良,谭雨辰.一种在 GPU 上高精度大型矩阵快速运算的实现.计算机应用,2009,29(4):1177−1179,1192. [doi: 10.3724/SP.J.1087.2009.01177]

[16] 李建明,迟忠先,万单领.一种基于 GPU 加速细粒度并行遗传算法的实现方法.控制与决策,2008,23(6):698−704.

[17] 曹锋,周傲英.基于图形处理器的数据流快速聚类.软件学报,2007,18(2):291−302. http://www.jos.org.cn/1000-9825/18/291.htm [doi: 10.1360/jos180291]

[19] 伍楠.高效能流体系结构关键技术研究[博士学位论文].长沙:国防科学技术大学,2008.

[20] 文梅,李海燕,伍楠,张春元,李海燕,李礼.流体系结构抽象模型研究.计算机工程与科学,2006,28(7):123−126.

[21] 杨雪梅,董逸生,徐宏炳,刘学军,钱江波,王永利.高维数据流的在线相关性分析.计算机研究与发展,2006,43(10):1744−1750. [doi: 10.1360/crad20061011]

[22] 杨明,刘先忠.矩阵论.武汉:华中科技大学出版社,2005.

周勇(1971－),男,山东安丘人,博士生,副教授,主要研究领域为高性能计算与数据流,软件体系结构可信与演化.

卢晓伟(1984－),男,硕士生,主要研究领域为高性能计算.

程春田(1965－),男,博士,教授,博士生导师,主要研究领域为水火电优化调度,电网经济运行,电网节能算法,智能算法,并行计算.