ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.2, February 2007, pp.291−302
DOI: 10.1360/jos180291

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

基于图形处理器的数据流快速聚类*

曹 锋+, 周傲英

(复旦大学 计算机科学与工程系,上海 200433)

Fast Clustering of Data Streams Using Graphics Processors

CAO Feng+, ZHOU Ao-Ying

(Department of Computer Science and Engineering, Fudan University, Shanghai 200433, China)

+ Corresponding author: Phn: +86-21-65643024, E-mail: caofeng@fudan.edu.cn, http://www.fudan.edu.cn

Cao F, Zhou AY. Fast clustering of data streams using graphics processors. *Journal of Software*, 2007,18(2): 291−302. http://www.jos.org.cn/1000-9825/18/291.htm

Abstract: Clustering data stream basically requires fast processing speed as well as quality clustering results. In this paper, some novel approaches are presented for such a clustering task using graphics processing units (GPUs), e.g., *K*-means-based method, stream clustering method, and evolving data stream analysis method. The common characteristics of these methods are making use of the strong computational and pipeline power of GPUs. Different from the pervious clustering methods with individual framework, the methods share the same framework with multi-function, which provides a uniform platform for stream clustering. In stream clustering, the core operations are distance computing and comparison. These two operations could be implemented by using capabilities of GPUs on fragment vector processing. Extensive experiments are conducted in a PC with Pentium IV 3.4G CPU and NVIDIA GeForce 6800 GT graphic card. A comprehensive performance study is presented to prove the efficiency of the proposed algorithms. It is shown that these algorithms are about 7 times faster than the previous CPU-based algorithms. Therefore, they well support the applications of high speed data streams.
Key words: data stream; clustering; graphics processor; evolving; window

摘 要: 在数据流环境下,聚类算法不仅需要有较高的聚类质量,同时需要有实时处理速度.因而,提出了一类基于图形处理器(graphics processing unit,简称 GPU)的快速聚类方法,包括基于 *K*-means 的基本聚类方法、基于 GPU 的数据流聚类以及数据流簇进化分析方法.这些方法的共同特点是充分利用了 GPU 强大的处理能力和流水线特性.与以往具有独立框架的数据流聚类算法不同,这些基于 GPU 的聚类算法具有同一框架和多种聚类分析功能,为数据流聚类分析提供了统一的平台.从分析可知,数据流聚类分析的核心操作实际上就是距离计算和比较.基于这一认识,利用 GPU 的子素向量处理功能进行距离计算.性能验证实验是在配有 Pentium IV 3.4G CPU 和 NVIDIA GeForce 6800 GT 显卡的 PC 上进行的.综合分析和实验结果表明,基于 GPU 的数据流聚类算法比传统的 CPU 算法平均快 7 倍,从而为高速数据流应用提供了良好的支持.
关键词: 数据流;聚类;图形处理器;进化;窗口
中图法分类号: TP311 文献标识码: A

* Supported by the National Natural Science Foundation of China under Grant Nos.60496325, 60496327 (国家自然科学基金)
Received 2005-09-01; Accepted 2006-03-29

随着传感器和计算机网络技术的发展,在越来越多的实际应用中,数据是以流的形式出现的.聚类作为一种重要的数据挖掘和分析技术,也越来越多地在数据流环境下展开研究[1−5].与传统的基于磁盘的聚类问题不同,数据流聚类问题除了需要考虑聚类质量以外,聚类处理速度往往也是数据流挖掘算法的一个重要指标.在数据流应用中,由于存储空间相对不足,新到达的数据必须实时、在线进行处理.当数据流出现 Burst 时(即短时间内有大量数据到达),数据流挖掘处理速度问题显得更为突出[3].数据流快速聚类已成为一个尚待解决的重要问题.

现代 GPU(graphics processing unit)中的图形流水线是可编程的,即允许用户编写简单的子素程序在子素处理器上并行运行.子素处理器不但可以直接访问纹理内存(GPU 中的内存),而且能够执行浮点向量操作.此外,普通 GPU 的处理能力已经相当强大.例如,NVIDIA GeForce6800 芯片比 Inter Pentium IV 3.73GHz 极限版处理器集成有更多的晶体管.GPU 的峰值处理能力以每年 2.5~3.0 倍的数据增长,远高于 CPU 的增长速度.由于图形硬件性价比的提升,GPU 成为 CPU 的一个日益重要的图形协处理器出现在普通 PC 上.GPU 最初被设计成用来快速处理图形顶点和图形子素流,GPU 的向量和流式数据处理能力是功能受限的[6,7].然而,由于经济因素和标准统一的问题,复杂的向量和流式处理器广泛出现在大众桌面上的状况目前还不太现实.因此,本文研究如何利用 GPU 日趋强大的处理能力来解决数据流的快速聚类问题.

本文的贡献在于:1) 分析现有的数据流聚类方法,提出了一种基于 GPU 的距离计算技术,用以执行数据流聚类中的核心操作;2) 利用 GPU 强大的处理能力和流水线特性,提出基于 *K*-means 的聚类算法、基于 GPU 的数据流聚类和簇进化分析方法.通过将距离计算操作转移到 GPU,提升了挖掘速度.同时,考虑 CPU 与 GPU 之间的较小总线带宽,将 CPU 与 GPU 之间的数据传输最小化.此外,与以往具有独立框架的数据流聚类算法不同,本文提出的基于 GPU 的聚类算法具有同一框架和多种数据流聚类功能,为数据流聚类提供了统一平台;3) 在一台装有 Pentium IV 3.4G CPU 和 NVIDIA GeForce 6800 GT 显卡的计算机上实现了本文提出的算法,并进行了全面的性能实验.实验结果证明了算法的有效性.基于 GPU 的聚类算法在具有相同聚类质量的同时,比传统 CPU 算法的处理速度提高了近 7 倍.

本文第 1 节回顾相关工作.第 2 节分析现有的数据流聚类算法.第 3 节对图形硬件进行概述.第 4 节介绍基于 GPU 的数据流聚类算法.第 5 节进行性能研究.最后是对全文的总结以及对未来研究工作的展望.

1 相关工作

由于实际应用广泛存在,聚类问题在数据库、数据挖掘和统计领域都得到了大量研究[1,8−11].传统聚类方法可划分为基于划分的方法[7,8]、基于层次的方法[11,12]、基于密度的方法[9]等.近年来,大量研究开始考虑在数据流环境下的聚类问题.数据流上的聚类问题可归结为两类:一类称为单遍扫描的数据流聚类算法[4,5,13,14],这类算法计算整个数据流中界标或滑动窗口中的数据的聚类结果;另一类则侧重于数据流中的簇进化分析[1,2],这类算法认为数据流中的簇是不断进化的,通过对数据流中的数据进行聚类处理,获取数据流中不断进化簇的信息.

随着图形硬件性价比的提高,图形处理领域已提出来很多基于图形硬件功能的新技术.例如:在 RECODE 算法[15]中采用深度缓存和模板缓存加速 3D 物体的碰撞检测;利用多网孔近似技术生成 2D 和 3D 的 Voronoi 图[16]等.区别于这些 2D 或 3D 近似的方法,本文的聚类算法可以对高维数据点进行精确的距离计算.高性能的顶点处理器和光栅化能力,使得 GPU 可以用来处理很多数值计算问题,包括稠密矩阵乘法[17]、向量处理[18]、几何计算[19]等.与这些基于顶点处理器的方法不同,本文利用了图形处理器子素级的向量处理能力.相对于顶点处理器来说,GPU 往往具有更多个子素处理器,因而子素级的向量计算可获得更高的并行处理能力.

在数据库领域中,同样有许多研究利用 GPU 来加速数据库的处理和计算.例如:Sun 等人利用渲染和查找技术来进行空间数据库中的空间选择和连接操作[20];Govindaraju 等人提出了基于 GPU 的数据库查询方法[21],该方法相对于基于 CPU 的查询方法取得了很大的性能提升;此外,Govindaraju 等人还提出了利用 GPU 来进行数据流的分位数和频繁项估计的快速计算[22].然而,至今还很少有人考虑利用 GPU 强大的处理能力去解决数据流聚类问题.

2 数据流聚类算法分析

数据流聚类问题概括来说是将源源不断到达的数据点划分为若干组,使得组内对象的相似性尽可能地要高,而组间对象的相似性尽可能地要低.在单遍扫描的数据流聚类算法中,聚类中间结果往往是一系列的中心点,如整个数据流的聚类算法 STREAM[5]和滑动窗口聚类算法[4].而在数据流簇进化分析的算法中,聚类中间结果往往是一系列 micro-cluster,如整个数据流的簇进化分析 CluStream[1]和滑动窗口簇进化分析等.一个以中心点作为最终结果的经典聚类算法 K-means 包含如下几个步骤[8]:(1) 初始化空间的 K 个点作为最初的中心点;(2) 将其余各个数据点分配到离它最近的中心点,形成 K 组数据点,并根据这 K 组数据点,重新计算各个组的中心点;(3) 反复执行步骤(2)和步骤(3),直到中心点不再变化为止.

在 K-means 算法中,有 4 个主要操作:初始化、分配、中心点计算和结束判断.分配操作(即距离计算和比较)往往是最耗时的部分.在 K-means 算法中,距离计算和比较的次数为 KRN,其中,K 表示中心点的个数,R 表示循环次数,N 表示数据点个数.

由于数据流聚类算法通常基于经典的聚类算法,在数据流聚类算法中往往仍采用距离作为对象间的相似性度量.在基于中心点的聚类算法中,随着数据流中数据块的不断到达,数据块中各个数据点需要计算与块中 K 个临时的中心点间的距离并进行比较,从而决定各个数据点的标号(离该数据点最近的那个中心点的标号).例如:STREAM 算法[5]对数据块大小为 B,长度为 N 的数据流至少需要进行 $\left\lceil \frac{N}{B} \right\rceil KRB$ 次距离计算和比较,其中,K 表示中心点的个数,R 表示块内 K-means 的循环次数.而在基于微簇(micro-cluster)的方法(如 CluStream[1])中,每一个新到达的数据点 p 都需要计算并比较与当前所有 βK(β为大于 0 的整数,通常可以取为 10;K 为簇的个数)个微簇的距离,从而获得数据点 p 与其最近微簇 $c_{\min}$ 间的距离 $d_{\min}$,进而根据 $d_{\min}$ 与 $c_{\min}$ 半径之间的大小关系决定 p 是否被 $c_{\min}$ 所吸收.例如,对于长度为 N 的数据流,CluStream 算法判断新到达数据点能否被吸收需要进行 βKN 次距离计算和比较.

从距离计算和比较的角度,数据流聚类计算的核心在于计算数据点与中心点或微簇的距离.而各聚类算法区别在于数据点、中心点以及微簇在整个算法中的生存期不同.在传统的 K-means 算法中,数据点在整个算法运行过程中是不变的,而中心点随着距离计算的进行不断发生变化.在 STREAM 算法中,数据点在数据块内的距离计算过程中是不变的,但是不断有新的数据块进入算法;中心点在块内距离计算过程中是不断变化的,当有新的数据块进入算法时,也会产生新的中心点.在 CluStream 算法中,数据点不断进入系统,数据点不断发生变化;同时,各个微簇根据新到达数据点的距离计算结果进行创建、合并或删除操作,因而各微簇是不断变化的.

从上述分析可以看出,距离计算和比较是数据流聚类中的核心操作,因而加速该核心操作可显著提升数据流聚类算法的性能.本文基于图形处理器(GPU)来加速数据流聚类过程中的距离计算.下面简要介绍图形处理器.

3 图形处理器简介

3.1 图形处理流水线

进入图形处理器的数据分为两类:一类是描述几何形状的顶点数据;另一类是像素数据,像素数据通常是从纹理中获得的图像或者字体.顶点处理器接收顶点数据并转化为屏幕上的窗口坐标.然后,这些顶点组合成几何形状.在光栅化过程中,形状所覆盖的像素被构造成许多子素.例如,被填充的多边形内部子素可在这一步骤中计算出来.最后,子素传入子素处理器进行处理并进行一系列的测试,如模板测试、深度测试等.子素只有通过这些测试才能被最终写入帧缓存区.通过这一系列过程,这两类数据都被转化为待显示的像素,并存储在帧缓存区中.根据所存储像素的不同属性,帧缓存区在概念上可以划分为 3 个缓存区:1) 颜色缓存,用于存储像素的颜色,像素颜色通常包含红、绿、蓝三色和一个 alpha 通道;2) 深度缓存,用于存储像素的深度值;3) 模板缓存,用于存储像素的模板值.模板值可以看作是屏幕上的一层掩模.图形渲染流水线如图 1 所示.

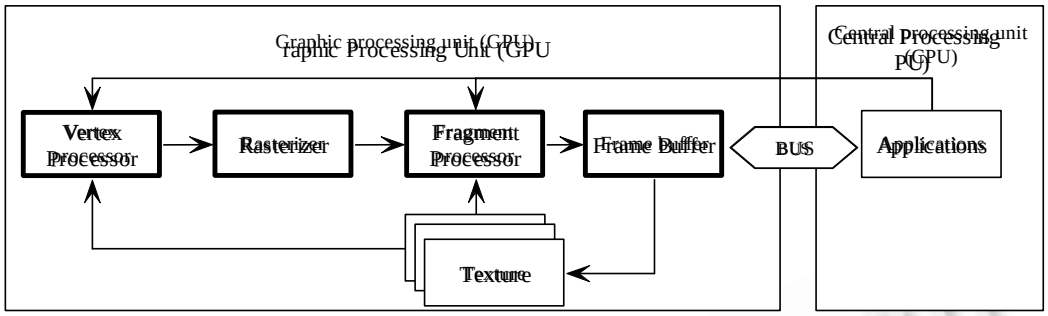

Fig.1 Graphics pipeline overview

图 1 图形渲染流水线简图

3.2 数据表示和子素程序

在本文基于 GPU 的聚类算法中,将要进行聚类的数据点被组织成纹理.纹理[7]在图形编程中通常用来将图像渲染到物体表面上.纹理可以看作是一个 2 维矩阵,矩阵上各元素包含多个通道.例如:一个 RGBA(red green blue alpha)纹理包含红、绿、蓝和 alpha 4 个通道.为了利用 GPU 进行聚类计算,数据点的各个属性存储在一个纹理点的各个通道内.在数据点维度高于 4 维的情况下,数据点各个维度存储在多个纹理的同一位置纹理点的多个通道内.纹理支持多种数据类型,如 8 位的 BYTE、16 位的整数和浮点数等.隶属于 PBuffer(一种屏幕不可见的帧缓存区)的纹理可支持高达 32 位 IEEE 单精度的浮点类型.为了利用存储在多个纹理的多维数据点进行聚类计算,本文将多个纹理渲染到同一个矩形上,并保证各个纹理点与帧缓存区中的像素点一一对应.

GPU 支持可编程的子素处理器.在子素处理器中运行的程序称为子素程序.子素程序可以对纹理进行检索并对子素执行数学计算.其输出结果是子素的新颜色和新深度值.也就是说,子素的新深度值可以是输入子素的颜色和坐标的函数值.本文将利用子素程序进行距离计算.

3.3 模板测试

模板测试用于对帧缓存的计算范围进行限定.一个子素通常对应于帧缓存中的一个像素.当新子素到达时,模板测试比较其对应像素在模板缓存中的模板值与参考值.如果比较结果为“真”,则通过模板测试;如果比较结果为“假”,则去除该子素.GPU 提供了一组模板操作对子素对应像素的模板值进行修改,例如,保持模板值不变,或用参考值替换模板值等.模板测试通常与深度测试相关联.例如:如果模板和深度测试都通过,则执行模板操作 1;如果模板测试通过,深度测试未通过,则执行模板操作 2.

4 基于图形处理器的聚类

4.1 数据转换

在数据流聚类过程中,对象之间的相似性通常是利用对象之间的距离来进行衡量的.欧几里德距离是一种广泛采用的度量方式.d 维向量 $\vec{X}$ 和 $\vec{Y}$ 之间的欧几里德距离定义如下:

$$dist(\vec{X}-\vec{Y})=\sqrt{\sum_{i=1}^{d}(x_i-y_i)^2} \tag{1}$$

内存将源源不断的数据流进行缓存并发送到 GPU 中(如图 2 所示).假设有 N 个 d 维数据点 $P_i(1\le i\le N)$和 K 个中心点 $C_j(1\le j\le K)$.首先,P_i 被组织成图 2 中的矩阵(2)形式(称为数据点矩阵),其中:M 表示矩阵的列数;L 表示矩阵的行数,$N=M\times L$.矩阵中的元素 $a[m][n]$对应于数据点 $P_{(m-1)\times M+n}$,其中,$1\le m\le L$,$1\le n\le M$.

在 GPU 算法实现过程中,一个矩阵对应于一个(或多个)纹理,而 M 和 L 通常都取为 $\lceil\sqrt{N}\rceil$,因为这种矩阵组织形式便于图形硬件更好地发挥子素计算的并行性.对于纹理中空缺的数据点位置,采用模板值加以屏蔽.如前所述,各数据点的各个维度值存储在各纹理点的各个通道内.由于每个纹理点 4 通道的限制,高于 4 维的数据点

按每 4 维进行划分,存储在不同的纹理中,并保证纹理中相同位置的纹理点对应于相同的数据点.对于不足 4 维的数据点,可以采用减少纹理通道的办法进行存储,如采用 RGB 纹理存储 3 维的数据点.数据点矩阵对 K 个中心点 $C_j(1\le j\le K)$分别计算距离矩阵 D_j,如图 2 中的矩阵(3),其中,$dot2(X)$表示对向量 X 和向量 X 做点积.D_j 中的元素 $d[m][n]$对应于从点 $P_{(m-1)\times M+n}$ 到中心点 C_j 的距离.由于欧几里德距离平方值不改变各距离值间的大小关系,本文采用距离平方值作为距离计算结果.

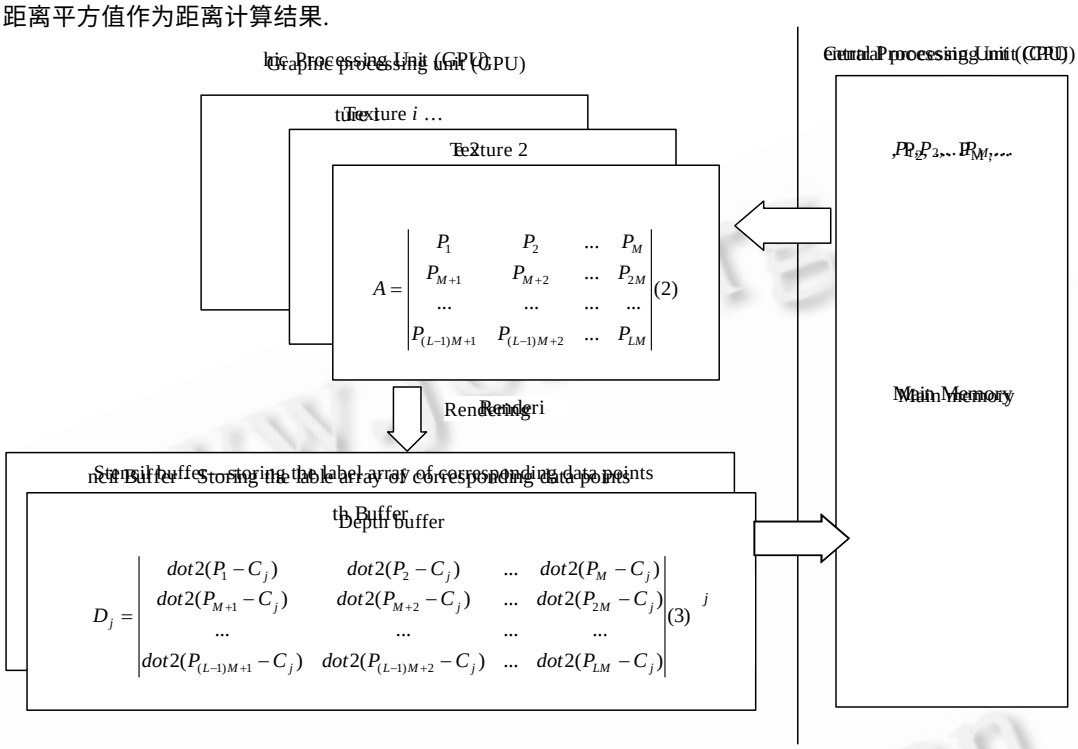

Fig.2 The process of data transition

图 2 数据转换流程

4.1.1 距离计算

GPU 具有硬件优化的向量处理能力,如进行点积计算、向量加法等.如图 3 所示,ComDistance 例程利用 GPU 向量处理能力进行低于 4 维数据点的距离计算.ActivePBuffer 激活 Pbuffer 进行离屏渲染,以获取更高的数据精度.第 2 行允许子素程序 FComDist 进行子素计算.RenderTexturedQuad 渲染一个以 *tex* 为纹理的矩形,保证 *tex* 纹理中各纹理点与 FComDist 产生的子素一一对应.子素程序 FComDist 计算从中心点 v_{cen} 到纹理 *tex* 中各个纹理点 v_{tex} 之间的距离.SUB 向量指令用 v_{tex} 向量减去 v_{cen}.DOT 向量指令对存储在临时变量 *tmpR* 中的向量做点积.最后,将点积所获得的距离值作为各子素的深度值调入深度缓存.

ComDistance(*tex*,v_{cen})
1: ActivePBuffer();
2: Enable fragment program FComDist;
3: *RenderTexturedQuad*(*tex*);
4: Disable fragment program FComDist;
FComDist()
1: v_{tex}=value from *tex*
2: *tmpR=SUB*(v_{tex},v_{cen})
3: result.depth = DOT(tmpR,tmpR);

Fig.3 The routine of distance computation (*dimension*≤4)

图 3 距离计算例程(维度小于或等于 4 维)

当数据点维度 d 高于 4 维时,如前所述,d 维数据点的每 4 维属性构成一个数据点矩阵.为了方便计算,d 维中心点同样按照每 4 维属性的方式划分成不同的段.数据点矩阵与中心点的对应段进行计算,并累计各距离矩阵,从而得到最终的距离矩阵.本文多纹理技术进行高于 4 维数据点的距离计算.当前 GPU 最多可同时支持 8 个纹理单元,也就是说,在一遍计算过程中,允许计算最多 32 维的数据点.然而,在下一代的 GPU 中可同时支持的纹理单元个数会越来越多.目前,当数据点维度超过 32 维时,本文采用多遍渲染的方法进行计算,并将各遍渲染结果进行累加,从而获得最终结果.若某 GPU 可同时支持 u 个纹理单元,则对 d 维数据点距离计算的渲染次数为 $\left\lceil \frac{d}{4u} \right\rceil$.

如图 4 所示,基于 GPU 的数据流聚类具有同一框架,下面我们将分别加以介绍.

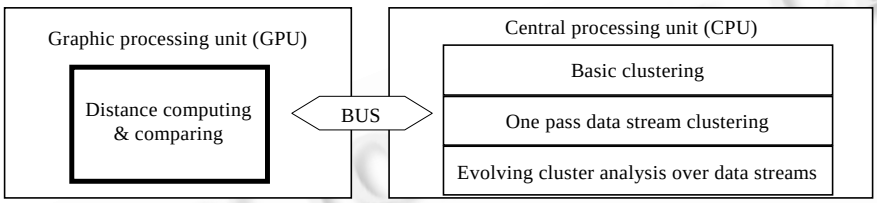

Fig.4 The framework of GPU-based stream clustering

图 4 基于 GPU 的数据流聚类框架

4.2 基于*K*-means的聚类

4.2.1 计算顺序调整

作为一种最基本的聚类方法,*K*-means 有着广泛的实际应用[8].它往往也是数据流聚类计算中的基本模块.因此,本文首先介绍一种基于 GPU 的 *K*-means 聚类方法.该方法重新调整了计算顺序,以更好地利用 GPU 进行并行计算.传统的基于 CPU 的 *K*-means 聚类是以数据点为中心的计算,即一次计算从某个数据点到 K 个中心点的距离并比较这 K 个距离值,从而获得该数据点最近的中心点,如图 5(b)所示.在本文基于 GPU 的 *K*-means 聚类算法中,以中心点为中心进行计算,即一次计算从某个中心点到各个数据点间的距离,然后并行地与上个中心点的计算结果进行比较,如图 5(a)所示.这种计算方式可以通过硬件并行执行.各个数据点的标号当且仅当 K 次以中心点为中心的计算结束后才能最终确定下来.

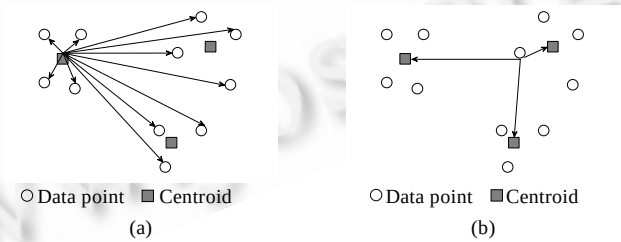

Fig.5 Centroid-Based vs. object-based distance computation

图 5 基于中心点和基于数据点的距离计算

4.2.2 距离比较与类标号计算

距离矩阵中的距离比较通过深度测试来实现.然而,在聚类计算中,仅仅能够进行距离计算和距离比较还不足以提供最终的计算结果.算法需要引入某种机制来存储计算和比较的结果,即与当前数据点距离最近的中心点的标号或微簇的标号(统称为类标号).由于模板缓存与各像素一一对应,并且与用于进行比较操作的深度测试具有内在关联,因而成为存储类标号的良好载体.模板缓存中的模板值记录了对应的数据点的标号,如图 2 所示.

类标号计算过程如下:(1) 通过调用 ComDistance 计算距离矩阵 D_1 并直接存入深度缓存中;(2) 将模板缓存

区中的模板值初始化为1,即将所有的数据点都标定为离中心点1最近;(3) 允许深度测试,保证随后计算出来的子素深度值只有当小于深度缓存中对应的深度时,才可以更新深度缓存,从而实现距离矩阵之间距离的比较;(4) 模板测试设置为一直通过状态.随后,依次计算其他距离矩阵 $D_{i+1}(1\le i<K)$.若 D_i 中新产生的子素通过深度测试,则更新深度缓存中对应像素的深度值,同时将对应像素的模板值更新为 $i+1$;否则,对应像素的深度值和模板值保持不变;(5) 当 K 个距离矩阵都计算完毕后,模板缓存区中保存了与各个数据点最近的中心点标号,而深度缓存区中则保存了各个数据点到最近的中心点间的距离.类标号计算过程利用了GPU的流水线特性,子素程序中的距离计算、深度测试和模板测试对应于图形处理流水线的不同操作,从而减少了整个标号计算过程的处理时间.

4.2.3 中心点计算

CPU通过向GPU输入新的中心点来控制聚类计算中的每一次循环.文献[21]中的实验证明GPU不适合处理累加运算,因而中心点的计算在CPU中进行.类标号计算结果首先从GPU中取回,然后在CPU中将标号相同的数据点累加并生成新的中心点.为了减少GPU与CPU之间的通信代价,本文采用GL_BYTE模式取回标号值,尽管这一模式有8 bit位的限制,即 K 最高可取为256,该模式也将满足绝大多数实际应用的需求.当 K 大于256时,算法将增加部分通信代价,即提高取回数据的数据位数以适应应用需求.

4.3 数据流聚类算法

4.3.1 单遍扫描的数据流聚类

单遍扫描的数据流聚类算法中常采用分而治之的策略.如STREAM算法[5]将数据流分块进行在线数据流聚类.各数据块的聚类可采用多种方法,如文献[5]中提出的LOCAL SEARCH,K-means等.在本文提出的基于GPU的单遍扫描数据流聚类算法中,对数据块的聚类基于 K-means.新到达的 N^{ε} 个点构成数据块,通过纹理传输的方式上载到GPU,而将计算结果(各块的中心点)维护在内存中.当同一级别的中心点个数达到了 N^{ε} 时,同样地,这些中心点组织成纹理并上载到GPU中进行计算.由于数据分块进行处理,在GPU中的显存开销限定为 N^{ε}.对数据块进行聚类可直接采用上一节介绍的基于GPU的 K-means方法.值得注意的是,由于大量的计算集中在GPU上,当GPU进行计算时,CPU几乎处于空闲状态.分配部分距离计算负载给CPU(例如,在GPU处理上次到达数据块的同时,对新到达的数据块采用CPU进行处理),可进一步提高算法的执行效率.

4.3.2 数据流中的簇进化分析

数据流的簇进化分析算法,通常包含在线聚类过程和离线分析过程两部分[1].在线聚类过程根据新到达的数据点,在线维护一组微簇.基于GPU的簇进化分析方法通过GPU来加速这一过程.一种直接运用GPU进行在线聚类的方法是:利用GPU查找与数据点 p 的最近微簇,将微簇的中心点构成纹理上载到GPU,当一个新的数据点 p 到达时,触发一次计算并通过GPU的最大值和最小值缓存获取距离最小值 $d_{\min}$;然后渲染一遍深度为 $d_{\min}$ 的矩形,确定与 p 距离为 $d_{\min}$ 的微簇 $c_{\min}$;最后将模板缓存值取回.模板值不为0的像素所对应的微簇即为与数据点 p 最近的微簇 $c_{\min}$.该算法的一个缺点是,每到达一个新数据点,就要重新传输一次微簇的中心点到GPU.

为了减少GPU和CPU之间的通信代价,本文提出一种批量计算方法,以提高基于GPU的簇进化分析效率.该方法基于如下观察:在绝大多数情况下,新数据点可被现有微簇所吸收,即新簇出现的频率远远小于数据点到达的速度.与基于GPU的 K-means处理过程相似,单位时间内到达的数据点构成纹理上载到GPU,依次比较纹理中的数据点与各个微簇中心的距离.当所有 βK 个微簇与纹理中的数据点都计算完毕后,取回标号矩阵.然后根据各数据点与最近微簇的距离关系决定是否吸收该数据点.若某数据点 p 不能被与其最近的微簇 $c_{\min}$ 所吸收,则根据 p 创建一个新微簇 c_p.后续数据点 p' 若不能被现有微簇所吸收,则尝试能否被根据 p 新建立的微簇 c_p 所吸收.若不能吸收,则根据 p' 创建一个新的微簇 $c_{p'}$.当有新微簇生成时,采用文献[1]中的策略删除或合并现有的微簇.

4.4 算法局限性分析

由于GPU的功能是进行图形图像渲染显示,基于GPU的数据流聚类算法由于受现有硬件条件的限制,存在

如下 3 个方面的局限性[7,22]:(1) 数据精度与算法精度.由于图形渲染对数据精度的要求相对较低,目前较新的 GPU 所支持的最高数据精度为 32 位 IEEE 单精度的浮点类型.由于受数据精度的限制,算法精度也相应受限.当数据流数据精度在单精度浮点范围内时,基于 GPU 的数据流聚类算法与 CPU 算法具有相同的算法精度以及时间和空间复杂度.我们的实验验证了该结论.(2) 纹理内存大小的限制.由于目前 GPU 硬件纹理内存相对较小,当数据量较大时,我们采用 out-of-core 技术将纹理内存的数据换入、换出.(3) 总线带宽.这是 GPU 算法的最大瓶颈.随着 PCI-EXPRESS 技术的逐渐成熟,这一瓶颈的影响将随之减小.总而言之,当前 GPU 硬件在上述 3 个方面的限制有望在下一代硬件中得到改善[22].

5 实验分析

5.1 实验环境

实验在一台装有 3.4GHz Pentrium IV CPU 和 NVIDIA GeForce 6800GT 显卡的 DELL PC 上进行.显卡具有 256M 显存,内存频率为 1000MHz,核心频率为 350MHz.实验真实数据集采用 KDD-CPU'99 网络入侵检测数据集.该数据集包含局域网内原始的 TCP 连接记录[5].由于人造数据集可以很好地控制簇的个数、数据点的个数以及数据点的维度,所以实验采用了人造数据集.在人造数据集中,数据点满足一组高斯分布,数据点的个数为 10K~10M,簇的个数为 8~128,维度是 4 维~28 维.而在簇进化分析实验的人工数据集中,每 1 万个数据点变换一次数据点的高斯分布.在 GPU 聚类过程中,数据点的各维属性为 32 位的单精度浮点数.

5.2 算法实现和评价

基于 GPU 的聚类算法采用 OpenGL API 加以实现,使用 NVIDIA 公司的 CG 编译器编译子素程序.基于 CPU 的聚类算法采用 Intel 公司的 C 编译器,并在编译选项中包含超线程和 SIMD 选项进行优化.GPU 与 CPU 之间的纹理和数据传送采用 AGP 8X 总线接口.算法效果评价采用广泛采用的 SSQ(sum of square distance quality)指标,即各数据点到中心点的距离的平方之和进行度量.效率评价采用算法执行时间进行度量.算法的总代价 *Ac*=磁盘 I/O 代价(I/O cost)+聚类计算代价 *Cc*.基于 GPU 的聚类算法需要在 GPU 和 CPU 之间传送数据.基于 GPU 的聚类计算代价可划分为在 GPU 中进行计算的代价 *Dc* 和从 GPU 传送数据到 CPU 的代价 *Tc*,即 *Cc*=*Dc*+*Tc*.各算法缩写见表 1.

Table 1 Abbreviation of algorithms
表 **1** 算法名称缩写

Algorithm name	Abbreviation	Algorithm name	Abbreviation
GPU-Based *K*-means	GPU-*K*	CPU-Based *K*-means	CPU-*K*
GPU-Based STREAM	STREAM-GPU	Local Search Based STREAM by CPU	STREAM-LS
GPU-Based CluStream (Online)	CluStream-GPU	*K*-means Based STREAM by CPU	STREAM-KM
CPU-Based CluStream (Online)	CluStream-CPU	BIRCH (balanced iterative reducing and clustering using hierarchies) and *K*-means to get final results	BIRCH-KM

5.3 基于*K*-means的聚类比较

5.3.1 聚类代价、总体代价与数据取回代价

实验首先比较了 GPU-*K* 和 CPU-*K* 的聚类代价.如图 6 所示,GPU-*K* 相对于 CPU-*K* 来说有大约 4 倍的性能提升.这种性能提升,首先得益于子素处理器的并行计算能力.现代 GPU 通常具有多个子素处理器.在本文进行实验的 NVIDIA GeForce 6800 中有 16 个子素处理器.其次,距离计算通过 GPU 中硬件优化的向量指令进行.从而进一步加速了聚类计算中的核心计算操作.最后,距离比较基于深度测试通过硬件方式进行,在整个比较过程中没有分支的错误预测.分支的错误预测在基于 CPU 的 *K*-means 程序中会占有大量的 CPU 时间.然后,实验对总体代价进行比较.如图 7 所示,GPU-*K* 和 CPU-*K* 随着数据点个数的增加,总体代价呈线性增长.GPU-*K* 的总体代价约为 CPU-K 的 60%.这是由于在总体代价中,I/O 代价占有较大比例,而在本文的实验中,计算循环次数一般不超过 20 次,当计算循环次数增加时,基于 *K*-means 的 GPU 算法的性能提升将更加显著.

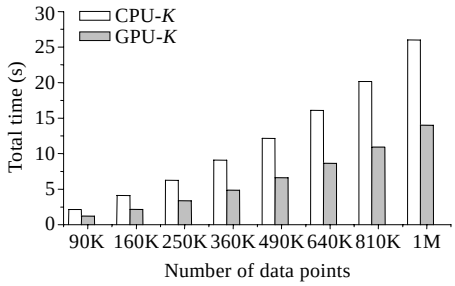

Fig.6 Total cost

图 6 总代价

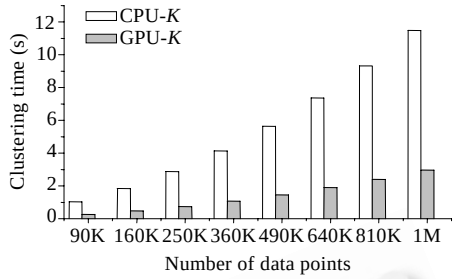

Fig.7 Clustering cost

图 7 聚类代价

由于 GPU 和 CPU 之间的数据传送是基于 GPU 聚类计算中的瓶颈,本文采用 GL_BYTE 紧缩模式传输类标号,从而极大地减少了 GPU 与 CPU 之间的通信量.图 8 显示出通信代价与聚类单次循环代价均随着数据量的增加而增长.然而,通信代价相对于聚类单次循环代价来说并不显著.

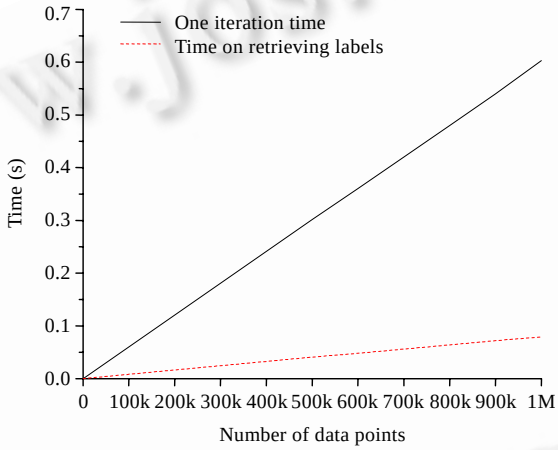

Fig.8 The time on retrieving labels

图 8 标号取回时间

5.3.2 聚类代价与 K 的变化

中心点个数 K 的变化会影响到算法的聚类代价.图 9 显示了聚类代价随 K 而增加,GPU-K 相对于 CPU-K 的性能优势越来越明显.GPU 距离的计算和比较是通过并行方式进行的.K 越大,GPU-K 的并行性能提升效果就越显著.

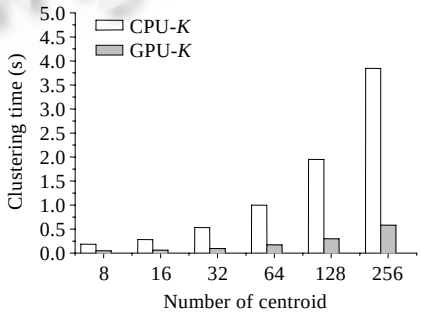

Fig.9 Clustering cost vs. K

图 9 聚类代价与 K 变化

5.4 数据流聚类比较

由于人造数据集可以获得更大的数据集,本文首先在人造数据集上进行数据流聚类的对比实验,其中 K=5.由图 10 和图 11 可见,STREAM-GPU 在获取与 STREAM-KM 相同 SSQ 的前提下,有大约 7 倍的性能提升.这种性能上的提升一方面来自于 GPU 的并行处理能力,另一方面来自于 GPU 的流水线处理方式适合于数据流环境下源源不断到达的流式数据[6].值得注意的是,与 BIRCH-KM 相比,STREAM-GPU 处理速度大约有 20%的提升,而 SSQ 指标有 200%的提升.也就是说,STREAM-GPU 相对于 BIRCH-KM 在获取更高处理速度的同时,具有聚类质量的大幅提升.与之相对应,尽管 STREAM-LS 相对于 STREAM-GPU 有接近 20%的性能提升,但却需要付出高达 15 倍的处理代价,而高速处理能力往往是数据流挖掘算法的一个重要指标.

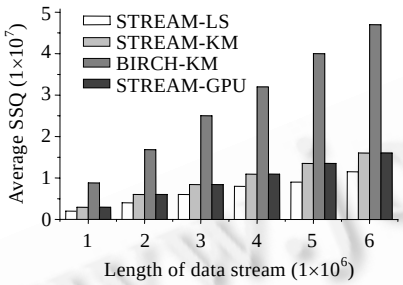

Fig.10 Effectiveness comparison (syn.)

图 10 效果比较(人工数据集)

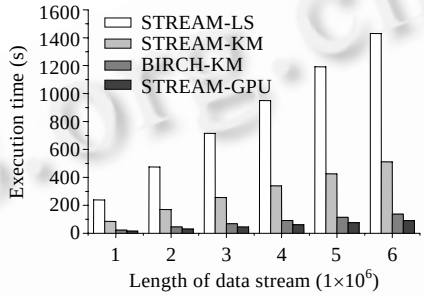

Fig.11 Efficiency comparison (syn.)

图 11 效率比较(人工数据集)

在 KDD CPU'99 真实数据集的实验中,STREAM-LS 的 SSQ 值在大多数情况下与 STREAM-GPU 相同,而在处理时间上却要高出 15 倍左右(如图 12 和图 13 所示).相对于 BIRCH-KM 而言,STREAM-GPU 具有 20%的处理速度提升,聚类质量在最坏情况下有 20%的提升,最好情况下具有高达 800%的质量提升.综上所述,STREAM-GPU 很好地满足了数据流环境下高聚类质量、高数据处理速度的需求.

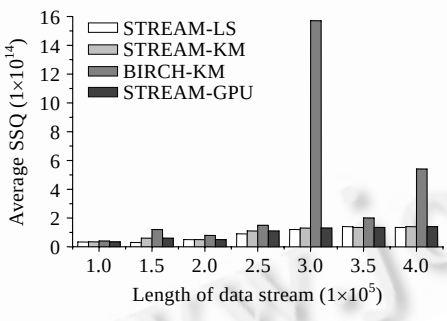

Fig.12 Effectiveness comparison (real)

图 12 效果比较(真实数据集)

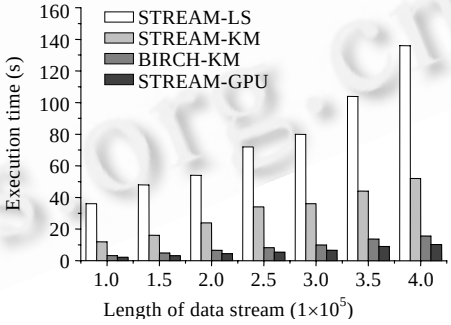

Fig.13 Efficiency comparison (real)

图 13 效率比较(真实数据集)

5.5 簇进化在线聚类性能的比较

实验首先采用 KDD CPU99 真实数据集进行性能比较,数据块的大小为 1 024,K=5,β=10.如图 14 所示,基于 GPU 的簇进化在线聚类算法相对于基于 CPU 在线聚类算法有 3 倍的性能提升,从而大大提高了簇进化在线聚类算法的吞吐能力.簇进化在线聚类算法性能提升相对较小,其原因在于 GPU 和 CPU 之间存在大量的通信开销.

数据块的大小可能会影响到基于 GPU 的聚类算法的性能提升,因此,我们还对不同数据块大小进行了实验.实验中的真实数据为包含 4.5×10^5 条记录的 KDD CPU'99 数据集.人工数据集为包含 2×10^6 条 8 维变换高斯分布的数据记录.如图 15 所示,随着数据块大小的增加,CluStream-GPU 的执行时间逐渐缩短,当数据块大小为

4096 时,执行时间最短.随后,执行时间会随着数据块大小的增加而逐渐增长.这是由于,一方面,随着数据块大小的增加,GPU 的并行性可以更好地得到体现;另一方面,当数据块大小增加到一定程度之后,有更多的工作负载将在 CPU 中处理,GPU 的并行处理能力反而不能得到有效发挥.随着数据块的增大,数据块内新簇出现的概率加大,数据点不能被现有簇所吸收的概率增加,从而有大量数据点不能被现有簇吸收.这些数据点需要在 CPU 中计算一次与新簇之间的距离,因而算法整体执行效率有所降低.

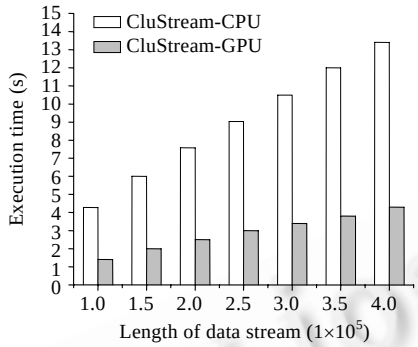

Fig.14 Efficiency comparison of CluStram

图 14 CluStream 的效率比较

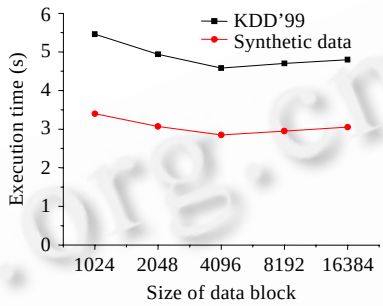

Fig.15 Execution time vs. block size

图 15 执行时间与数据块大小变化

6 结论和展望

本文分析得出,数据流聚类分析的核心操作实质上就是距离计算和比较,并提出了一类基于 GPU 的快速聚类算法,包括基于 GPU 的 *K*-means 聚类方法、单遍扫描的数据流聚类方法和数据流中的簇进化分析方法.这类方法都利用了 GPU 的并行性和流水线特性.聚类算法中的核心操作,即距离计算和比较,利用 GPU 的子素向量处理能力来实现.此外,本文提出的基于 GPU 的聚类算法具有同一框架、多种数据流聚类功能和良好的可扩展性,为数据流聚类分析提供了统一的平台.理论分析和实验结果证明了基于 GPU 的数据流聚类算法的高效性,为高速数据流应用提供了良好的支持.今后的工作包括研究其他基于 GPU 的数据挖掘算法,如离群点检测和分类算法等.

References:

[1] Aggarwal CC, Han J, Wang J, Yu PS. A framework for clustering evolving data streams. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Int'l Conf. on Very Large Data Bases. Berlin: Morgan Kaufmann Publishers, 2003. 81−92.

[2] Aggarwal CC, Han J, Wang J, Yu PS. A framework for projected clustering of high dimensional data streams. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB, eds. Int'l Conf. on Very Large Data Bases. Toronto: Morgan Kaufmann Publishers, 2004. 852−863.

[3] Babcock AK, Babu S, Datar M. Model and issues in data stream systems. In: Popa L, ed. Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. Madison: ACM Press, 2002. 1−16.

[4] Babcock B, Datar M, Motwani R, Callaghan LO. Maintaining variance and *k*-medians over data stream windows. In: Proc. of the 22nd ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems. San Diego: ACM Press, 2003. 234−243.

[5] Guha S, Meyerson A, Mishra N, Motwani R, Callaghan LO. Clustering data streams: Theory and practice. IEEE Trans. on Knowledge and Data Engineering (TKDE), 2003,3(15):515−528.

[6] Venkatasubramanian S. The graphics card as a stream computer. In: SIGMOD-DIMACS Workshop on Management and Processing of Data Streams. 2003. http://citeseer.ist.psu.edu/687799.html

[7] 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

[8] Han J, Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, 2001.

[9] Ester M, Kriegel HP, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Simoudis E, Han JW, Fayyad U, eds. Proc. of the 2nd Int'l Conf. on Knowledge Discovery and Data Mining (KDD'96). Portland: AAAI Press, 1996. 226−231.

[10] Guha S, Rastogi R, Shim K. Cure: An efficient clustering algorithm for large databases. In: Haas LM, Tiwary A, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Seattle: ACM Press, 1998. 73−84.

[11] Zhang T, Ramakrishnan R, Livny M. BIRCH: An efficient data clustering method for very large databases. In: Jagadish HV, Mumick IS, eds. Proc. of the 15th ACM SIGMOD Int'l Conf. on Management of Data. Montreal: ACM Press, 1996. 103−114.

[12] Karypis G, Han EHS, Kumar V. Chameleon: Hierarchical clustering using dynamic modeling. Computer, 1999,32(8):68−75.

[13] Guha S, Mishra N, Motwani R, Callaghan LO. Clustering data streams. In: Proc. of the 41st Annual Symp. on Foundations of Computer Science. Redondo Beach: IEEE Computer Society, 2000. 359−366.

[14] Chalaghan LO, Mishra N, Meyerson A, Guha S. Streaming data algorithms for high-quality clustering. In: Proc. of the 18th Int'l Conf. on Data Engineering. San Jose: IEEE Computer Society, 2002. 685−694.

[15] Baciu G, Wong WSK, Sun H. Recode: An image-based collision detection algorithm. Visualization and Computer Animation, 1999,10(4):181−192.

[16] Hoff III KE, Keyser J, Lin M, Manocha D, Culver T. Fast computation of generalized voronoi diagrams using graphics hardware. In: Proc. of the 26th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press/Addison-Wesley Publishing Co., 1999. 277−286.

[17] Larsen ES, McAllister DK. Fast matrix multiplies using graphics hardware. In: Proc. of the 2001 ACM/IEEE Conf. on Supercomputing. Denver: ACM Press, 2001. 55.

[18] Thompson CJ, Hahn S, Oskin M. Using modern graphics architectures for general-purpose computing: A framework and analysis. In: Proc. of the IEEE/ACM Int'l Symp. on Microarchitectures. Istanbul: ACM/IEEE, 2002. 306−317.

[19] Agarwal P, Krishnan S, Mustafa N, Venkatasubramanian S. Streaming geometric optimization using graphics hardware. In: Battista GD, Zwick U, eds. Proc. of the 11th European Symp. on Algorithms. Budapest: Springer-Verlag, 2003. 554−555.

[20] Sun C, Agrawal D, Abbadi AE. Hardware acceleration for spatial selections and joins. In: Halevy A, Ives ZG, Doan A, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. San Diego: ACM Press, 2003. 455−466.

[21] Govindaraju NK, Lloyd B, Wang W, Lin M, Manocha D. Fast computation of database operations using graphics processors. In: Gerhard W, Arnd Christian K, Stefan D, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Paris: ACM Press, 2004. 215−226.

[22] Govindaraju NK, Raghuvanshi N, Manocha D. Fast and approximate stream mining of quantiles and frequencies using graphics processors. In: Ozcan F, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: ACM Press, 2005. 611−622.

附中文参考文献:

[7] 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

曹锋(1977－),男,上海人,博士,主要研究领域为数据流,数据挖掘.

周傲英(1965－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据流,数据挖掘,XML 数据管理,P2P 计算.