数据流窗口连接查询处理器研究

钱江波[1], 王永利[2], 陈 征[1], 陈华辉[1], 金 光[1]
(1. 宁波大学信息科学与工程学院, 浙江宁波 315211; 2. 南京理工大学计算机科学与技术学院, 江苏南京 210094)

摘 要: 高速数据流处理是数据流管理系统的一个关键问题. 目前已有系统一般采用查询优化、系统调度、降载等方法来提高速度, 在高速数据流环境下存在明显的不足. 为了最大程度地提高数据流连接操作的速度, 提出专用硬件处理器 WJSP 及设计. 针对千变万化连接条件所共同的基本操作, 提出在 WJSP 可执行的指令系统 WJSI, 并提出多核处理器以提高 WJSP 的扩展性和并行性. 实验结果显示 WJSP 原型比 STREAM 的速度提高 10 多倍, 具有相当高的处理性能. WJSP 可嵌入到路由器、交换机、传感器等设备中, 提供高速数据流连接处理.

关键词: 连续查询; 窗口连接; 查询处理; 指令集设计
中图分类号: TP332, TP311 文献标识码: A 文章编号: 0372-2112 (2009) 02-0404-06

Hardware Processor for Window Joins over Multiple Data Streams

QIAN Jiang-bo[1], WANG Yong-li[2], CHEN Zheng[1], CHEN Hua-hui[1], JIN Guang[1]
(1. *School of Information Science and Engineering, Ningbo University, Ningbo, Zhejiang 315211, China;*
2. *School of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing, Jiangsu 210094, China*)

Abstract: Accelerating data processing is one of the key problems in DSMS(Data Streams Management System) research. Many researchers promote processing speed by query optimization. It potentially results in saturating the CPU. Load shedding is a candidate choice when a DSMS is executing aggregate operations and it becomes overload. However, load shedding can not be applied on join operation for it will potentially lose many results. In order to greatly accelerate join processing, we present a novel hardware processor implementation for window join evaluation over multiple data streams. To address variations in join conditions, we propose a set of instructions that can be executed in the processor. In addition, we introduce a multi-core processor to improve scalability and flexibility of the processor. Experimental results show that the processor outperforms software by more than ten times.

Key words: continuous query; window joins; query processing; instruction-set design

1 引言

近年来数据流在越来越多的实际应用中出现, 传统数据库在处理该类数据时表现出极大局限性[1~4]. 而数据流管理系统[1]适应流的特点, 从底层开始就与传统数据库有较大的不同. 目前国外已有一些原型系统, 如 TelegrahpCQ[2], Aurora[3], STREAM[4]等.

为方便表述, 先说明以下名词含义.

数据流(data stream)[5]. 数据流 S 由元组$< s, t >$ 组成, s 是该数据流模式的元素, t 是时间戳.

基于元组数(count-based or sequence-based sliding window)的滑动窗口[5]. 数据流 S 上滑动窗口 n 定义为S 的一个子集, 在任意时刻 t, 该子集由最近的 n 个元素构成. 如果窗口已满, 新元组的到来替换最旧的元组.

连续查询(continuous queries)[6]. 连续查询是长期运行的、连续不间断的、常设的、持久的查询(long-running, continuous, standing, and persistent queries). 在一段时间内, 连续查询是对数据流不停地、连续地执行查询, 对新到来的元组执行操作, 增量式产生新的查询结果.

窗口连接[7]. 设有 n 条数据流连接, 对于每个输入元组, 必须与其它所有数据流窗口内元组执行连接操作, 然后将结果以数据流形式输出.

窗口连接是数据流管理系统的关键操作之一, 提高该操作的速度也是目前该领域的一个重要研究内容. 本文的主要贡献包括: (1) 综合机器指令设计、处理器设计等多项技术, 设计实现用于高速数据流的多核处理器; (2) 设计实现相应的指令集; (3) 通过对比实验, 显示该处理器比斯坦佛大学的 STREAM 的速度提高 10 多倍.

收稿日期: 2007-08-01; 修回日期: 2008-10-15
基金项目: 国家自然科学基金(No. 60803021, No. 60803001); 宁波市自然科学基金(No. 2007A610007); 宁波大学人才工程(No. XR0710004); 江苏省博士后基金(No. AD41190); 南京理工大学科研发展基金(No. XKF07030); 浙江省教育厅科研项目(No. 20070978)

2　相关工作

尽管数据流的连接操作被限制在滑动窗口内, 但当窗口内元组数量较多时, 连接操作还是非常耗时间的. 如果处理一个元组的时间大于该流元组到来时间的间隔, 输出将被滞后, 同时输入的元组将被积累等待处理. 目前国内外对于高速数据流处理方式一般为: (1) 在软件系统上做一些优化以期提高速度, 但提高性能有限; (2) 设计降载操作, 丢弃一些来不及处理的数据, 但对连接等操作存在较大误差[8~10]. 因此国内外有专家采用硬件系统加速处理.

文献[11] 认识到低延迟、高吞吐量是数据流管理系统需要解决的关键问题, 利用 chip multiprocessor 本身具有的并行性来提高数据流连接操作的处理速度. 文献指出由移动指针完成基本窗口的划分, 将元组按属性存储能够减少连接时数据的移动, 双缓冲区技术能够提高流水线操作性能, 与数据流速率相关的批处理技术能够平衡连接时吞吐量和元组延迟这两个矛盾. 实验结果表明片内多重处理器能够有效用于数据流连接操作, 其处理速度约为 3.2Ghz Intel Xeon 速度的 8.3 倍.

在数据库操作中, GPU 在某些操作上能比 CPU 处理速度快, 其加速主要来自于几个方面: (1) 子素处理器的多像素并行处理; (2) GPU 的高度流水线结构使得多个元素可以在流水线上同时得到处理; (3) GPU 的深度剔除功能使得部分子素可以在流水线中较早地被排除在外; (4) 在 GPU 实现的这些查询操作中不存在分支, 如误预期(misprediction) 操作. 利用 GPU 加速技术, 文献[12] 讨论数据库的范围查询(Range Query), 最值查询(K-th Largest) 等, 文献[13] 讨论空间数据库查询, 文献[14] 讨论数据库排序.

网络处理器的结构与通用处理器不同, 一般拥有多个微处理引擎(micro engine) 和多层的存储结构. 文献[15] 讨论了三种基本的数据库操作: 顺序查找、簇集和非簇集索引查找、哈希连接. 因网络处理器多核心、多线程的特殊结构, 使其能够并行存取存储器, 效率比常规的单线程通用处理器大为提高. 文献[16] 整合 NPU 与三重内容寻址存储器(Ternary Content Addressable Memories, TCAMs), 利用固定时间的内容寻址及高并行性, 提供高速求数据流上频繁项集的方法.

在国内, 也有专家采用硬件来提高数据流的处理速度. 文献[17] 注意到数据流聚集查询处理很耗时间, 提出一种基于硬件加速的高速数据流聚集查询方法, 充分发挥硬件在处理速度上的优势和软件在灵活性方面的长处. 文献[18] 提出基于 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.

3　WJSP 设计

尽管数据流的窗口连接操作复杂并且需要比较多的资源, 但其最基本的是比较和逻辑操作, 这些操作都可以采用硬件来加速处理. 但是, 单纯的硬件系统灵活性不足, 不能满足千变万化的不同连续查询条件的需要. 为此, 我们设计并实现了用于处理数据流并发窗口连接的处理器 WJSP(Window Join Special Processor) 及其指令系统 WJSI(Window Join Specific Instruction set), 以提供高性能的连接处理并满足不同的连续查询的需要.

3.1　WJSP 原型逻辑设计

图 1 是能处理 3 条数据流连接的 WJSP 原型的逻辑设计图, 也是 M3Join[19] 多流并发连接方法的一个硬件实现. WJSP 中元组的连接操作以类似网络数据包查找路由表的方式进行, 以控制探测序列. 图 1 中 WJSP 由 4 部分处理单元组成, 一个 RIOU(Routing and IO Unit) 和三个 JoinUnit(JU). RIOU 完成数据流的输入输出、查找连接路由表以及一些控制功能. 对于每条参与连接的数据流, 均有一个独立的处理单元 JU 处理与该流相关的比较、逻辑和连接等操作. 每个 JU 中均设置了指令存储器、指令地址寄存器、比较器、逻辑运算器、连接操作器、数据寄存器等执行和控制单元. 数据流元组通过 RIOU 的 DataIn 端输入, 在 Control Unit 控制下, 根据元组扩展的路由标记找到该元组对应的连接路由表(Join Routing Table, JRT) 记录, 然后将该元组插入对应的 JU (路由记录的 Insert 位为 1, 作为插入使能) 中的窗口缓冲区, 同时进入 JU(路由记录的 Probe 位为 1) 处理. JU 执行完相关的比较、逻辑和连接等操作后, 更改结果元组的路由标记, 将结果元组返回 JRT, 根据新路由标记查找 JRT 后决定将该结果元组返回查询用户(路由记录 Queries 位为 1) 或继续探测, 进入相应 JU 处理(路由记录 Probe 位为 1). 连接后数据流元组(图 2) 总共有 152

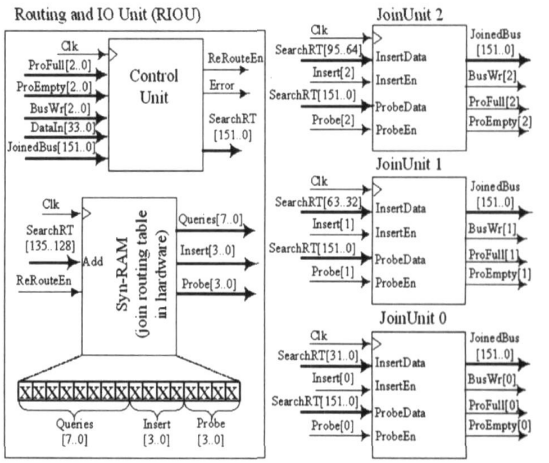

图1　WJSP原型逻辑设计

位, 其中包括3 个元组、一个路由标记、一个最大时间戳和一个最小时间戳. 最大最小时间戳的差可以判断元组是否符合查询的窗口条件.

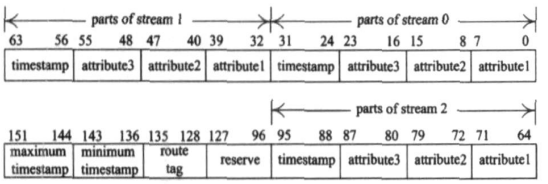

图2 WJSP原型中数据流格式

3.2 WJSI 指令系统

单纯的硬件系统灵活性不足, 不能满足千变万化的不同连续查询的需要. 为此, 在 JoinUnit 中采用指令系统 WJSI, 以提供高性能的连接处理并满足不同的连续查询的需要.

图 3 是 WJSI 指令格式, 这些指令的组合(WJSI 程序) 可以完成窗口连接的所有操作.

下面以例 1 说明.

例 1 数据流管理系统注册了下面 9 条查询, 设图 1 的 JoinUnit0 处理 R 流相关操作, 包括; R. a= S. d, R. b= S. e, R. a= S. d or R. c= S. e, R. a= S. d and R. c= S. e 等比较和逻辑操作, 经过编译, 这些操作可以编译成表 1 的指令.

Query1: select * from R, S where R. a= S. d window 500;

Query2: select* from R, S, T where R. a= S. d and S. e = T. f window 1000;

Query3: select* from S, T where S. e= T. f window 500;

Query4: select * from R, S where R. b= S. e window 1000;

Query5: select* from R, S where R. a= S. d or R. c= S. e window 500;

Query6: select * from S, T where S. e= T. g window 1000;

Query7: select* from R, S where R. a= S. d and R. c= S. e window 1000;

Query8: select * from S, T where S. d= T. f window 2000;

Query9: select* from R, S, T where(R. c= S. e or S. e= T. f) and R. a= S. d window 2000;

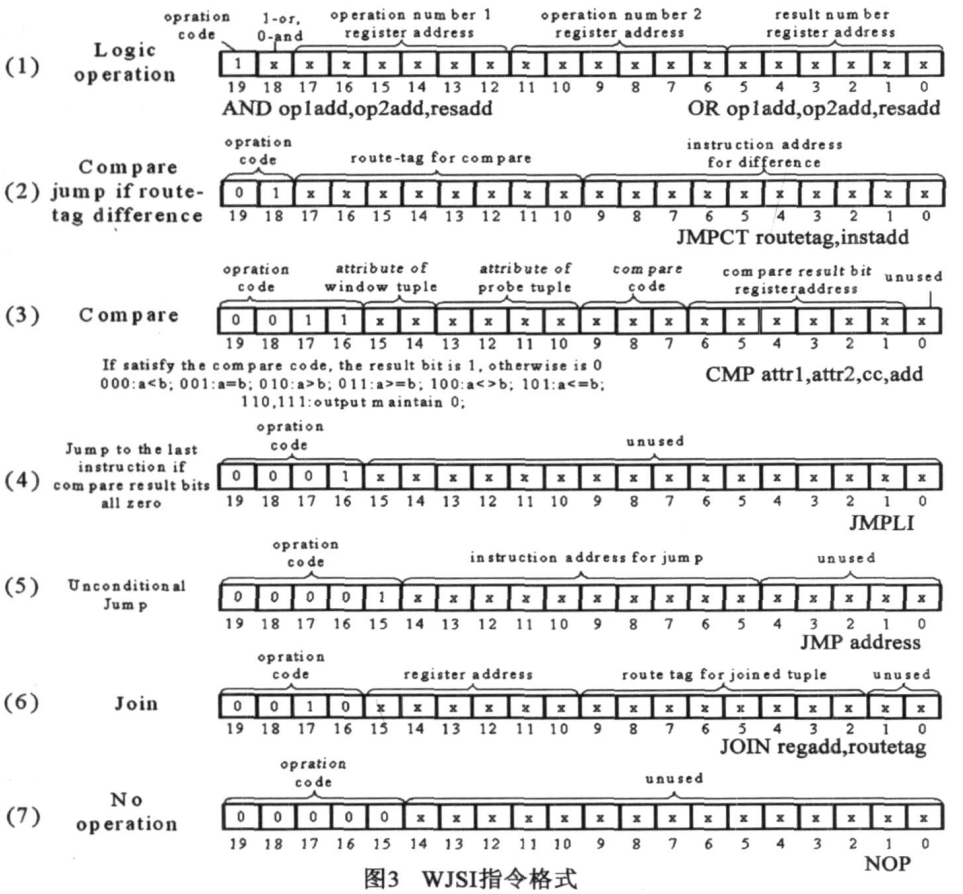

图3 WJSI指令格式

表1　WJSI 组成的程序完成 R 流相关操作(JoinUnit0 内程序)

指令地址	指令	机器指令	完成功能
0	CMP 00,0100,001,000000	0011,0001,0000,1000,0000	R.a= S.d 的比较结果存到数据寄存器地址 010000
1	CMP 01,0101,001,000001	0011,0101,0100,1000,0010	R.b= S.e 的比较结果存到数据寄存器地址 000001
2	CMP 10,0101,001,000010	0011,1001,0100,1000,0100	R.c= S.e 的比较结果存到数据寄存器地址 000010
3	JMPLI	0001,0000,0000,0000,0000	如果全零不必再往下计算,提高效率
4	OR 000000,000010,0100001	100,0000,0000,1001,0000	000000∪000010→010000, R.a= S.d or R.c= S.e 存到数据寄存器地址 010000
5	AND 000000,000010,010001	1000,0000,0000,1001,0001	000000∩000010→010001, R.a= S.d and R.c= S.e 存到数据寄存器地址 010001
6	JMPCT 00000001,0000001100	0100,0000,0100,0000,1100	如果输入元组路由标记不是 1,则跳到指令地址 12
7	JOIN 000000,00000100	0010,0000,0000,0001,0000	输入元组路由标记是 1,如果有连接结果(数据寄存器地址 000000 内容为 1),则连接后元组标记为 4
8	JOIN 000001,00000101	0010,0000,0100,0001,0100	输入元组路由标记是 1,如果有连接结果(数据寄存器地址 000001 内容为 1),则连接后元组标记为 5
9	JOIN 010000,00000110	0010,0100,0000,0001,1000	输入元组路由标记是 1,如果有连接结果(数据寄存器地址 010000 内容为 1),则连接后元组标记为 6
10	JOIN 010001,00000111	0010,0100,0100,0001,1100	输入元组路由标记是 1,如果有连接结果(数据寄存器地址 010001 内容为 1),则连接后元组标记为 7
11	JMP 0000001110	0000,1000,0001,1100,0000	跳过下面操作至地址 14
12	JMPCT 00001100,0000001110	0100,0011,0000,0000,1110	如果输入元组路由标记不是 12,则跳到指令地址 14
13	JOIN 000000,00010000	0010,0000,0000,0100,0000	输入元组路由标记是 12,如果有连接结果(数据寄存器地址 000000 内容为 1),则连接后元组标记为 16
14	NOP	0000,0000,0000,0000,0000	空操作,指示结束

4　可扩展性和局限性

由于一片芯片只有有限的资源,所以必须考虑 WJSP 的可扩展性,使 WJSP 能够处理不同数量的数据流和不同大小的窗口. 我们采用三类组件集成一个大容量多核 WJSP. 前两类组件能在图 1 找到,分别是 JU 和 RIOU. 第三类组件是图 4 中的 WinCtrlUnit(WCU). 在 JU 中的比较和连接操作是串行的,因此当窗口较大时执行时间会比较大. 另外,如果窗口较大的话,由于容量的限制,可能无法放入一片芯片中. 这时,我们可以用几个 JU 和一个 WCU 组成一个多核 JU(Multi-core JU, MJU),并且也提高了操作的并行性. WCU 其实就是一个移位寄存器,其中仅有一位是 1,用于插入使能,而其它位都是 0.

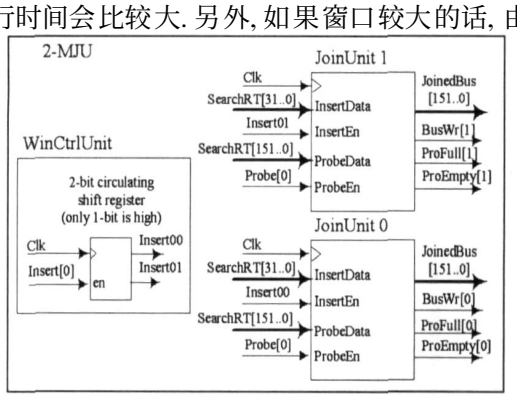

图4　2个JU组成一个MJU

图 4 中 WJSP 的 MJU 都是由 2 个基本 JU 组成. 与图 1 比较,主要的不同是 insert 引脚. 为了防止产生重复元组,我们将需插入窗口的元组轮流插入到 2 个 JU 中,而探测元组必须都插入到这 2 个 JU 中以正确执行连接. 其实上述方法就是将一个窗口分成两个窗口来并行执行连接. 当然,我们也可将一个窗口分为更多个窗口进一步提高处理的能力.

5　实验结果

我们采用 Altera 公司的 FPGA 开发工具 Quartus II 6.1① 来模拟实现 WJSP. 为了测试 WJSP 的性能,我们将 WJSP 的处理速度与 STREAM② 作了比较. STREAM 是斯坦佛大学研究的典型数据流管理原型系统. 我们将

① http://www.altera.com
② http://www-ab.stanford.edu/stream/

STREAM 安装在 Intel PIV 2.4G,512M 内存的 Redhat Linux 8.0(内核 2.4.18-14)上,注册例 1 的 9 条连接查询(改变查询的窗口谓词分别为 500,1000,2000,4000),注册的数据流为 R(a,b,c),S(d,e)和 T(f,g).元组的属性值在[0,255]平均随机分布(因原型 WJSP 处理元组为 8 位),速率一样.

用 STREAM 命令"gen_client -l[log-file] -c[config-file][script-file]",记录其运行日志文件中的查询处理时间.在"streamr-0.6.0/config"中修改了 2 项参数:(1)将 MEMORY_SIZE 从 33554432(32M)修改为 134217728(128M);(2)将 RUN_TIME 从 1000 修改为 1000000. STREAM 的 gen_client 命令是从磁盘中读取数据流元组,并将处理结果写回磁盘,所以其日志文件中记录的处理时间包括较多的 I/O 时间.为测量精确的处理速度,我们作了 2 处修改以去除 I/O 的影响:(1)为去除 Output 时间,在"gen_client/gen_output.cc"中注释结果元组输出语句;(2)为去除 Input 时间,我们用 Linux 命令"mke2fs /dev/ram0"创建了一个 RamDisk,将原始数据流文件存到 RamDisk,并配置使 STREAM 系统从 RamDisk 中读数据.当用 STREAM 原型执行查询时,我们仅注册了 Query1~4 和 6~8,因 STREAM 不支持 OR 谓词.尽管没有 Query5,9,我们认为不影响系统性能分析,因为执行三流连接的 Query2 占用了主要处理时间.

当不同个数的 JU 组成 MJU 时,WJSP 的处理速度是不同的.我们测试了 7 种 WJSP 方案,它们每个 MJU 的窗口缓冲数都是 4096 元组,但组成的 JU 数不同.如 WJSP 的处理 R 流 MJU 由 64 个 JU 组成,而每个 JU 包含 64 个窗口缓冲元组.我们用 Stratix ⅢEP3SL200F1517C2 实现 WJSP.表 2 是这 7 种 WJSP 方案,及 WJSP 运行的最高频率和所占的资源.

表 2 WJSP 频率及消耗的硬件资源

组成 MJU 的基本 JU 个数	基本 JU 窗口元组数	MJU 窗口元组数	最高频率	逻辑单元消耗	内存消耗
1	4096	4096	115.97M	1%	17%
2	2048	4096	115.75M	2%	17%
4	1024	4096	103.53M	6%	17%
8	512	4096	92.58M	12%	17%
16	256	4096	84.60M	28%	17%
32	128	4096	69.30M	53%	17%
64	64	4096	51.87M	89%	17%

为了评估 WJSP 的执行效率,我们将元组存于 FPGA 的 RAM 中,然后模拟执行这些元组.一个元组执行完后立即执行下一个,从而达到 WJSP 最大的执行速度.从模拟执行的每个元组的平均执行周期数,可以得出处理一个元组的平均时间.图 5 是 WJSP 与 STREAM 处理速度的比较.当 WJSP 的窗口缓冲区由 1 个 JU 组成时,由于主频的优势(2.4G 对 115.97M),STREAM 处理速度比 WJSP 略快.随着组成 MJU 的 JU 数增加,WJSP 处理的并行度提高,处理速度也明显增加.当 WJSP 的窗口缓冲区由 64 个 JU 组成时,其处理速度是 STREAM 的 10 多倍.

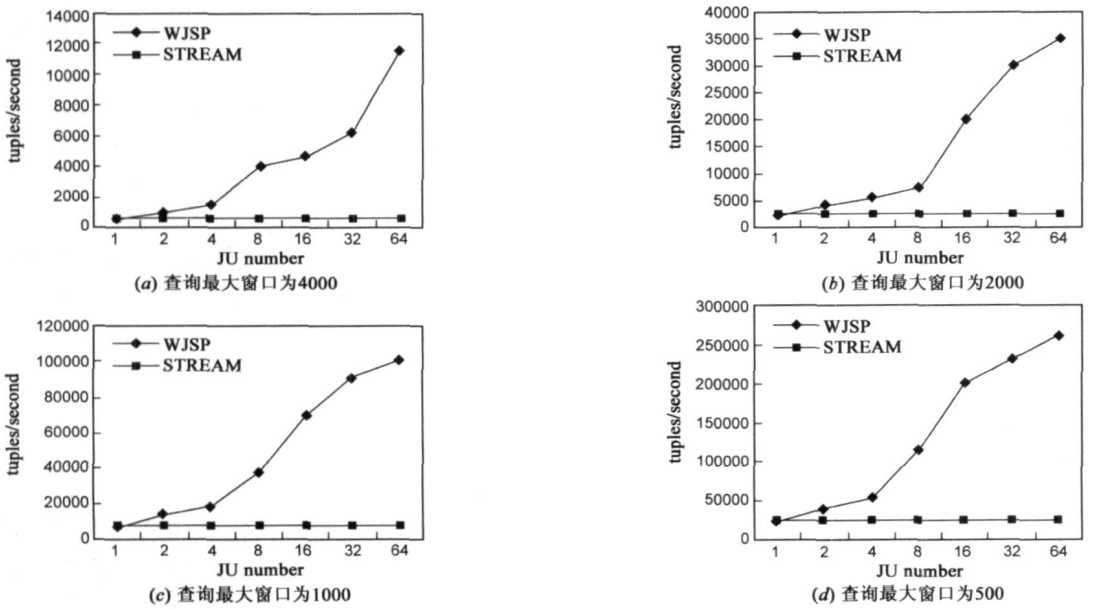

图5 WJSP 和STREAM性能比较

6 结束语

绝大多数连续查询处理算法是用软件实现的, 我们提出用硬件加速数据流连接处理的方法, 极大地提高了处理的速度. 为了处理不同的连接条件, 我们还设计了指令系统 WJSI. 同时也提出了多核集成方法以提高WJSP 的扩展性, 进一步提高处理的速度. 实验显示 WJSP 比 STREAM 的软件实现提高 10 多倍. WJSP 可嵌入到路由器、交换机、传感器等设备中, 提供高速的数据流连接处理. 廉价且高性能的连接操作协处理器能够增强 DSMS 的处理能力, 尤其是在类似网络监控等性能要求特别高的的领域.

参考文献:

[1] Babcock B, Babu S, Datar M, et al. Models and issues in data streams[A]. Proc ACM Symp on Principles of Database Systems[C]. New York: ACM Press, 2002. 1- 16.

[2] Raman V, Deshpande A, Hellerstein J M. Using state modules for adaptive query processing[A]. ICDE 2003[C]. Los Alamitos: IEEE Computer Society, 2003. 353- 364.

[3] Carney D, Cetinternel U, Cherniack M, et al. Monitoring streams a new class of data management applications[A]. VLDB 2002[C]. San Fransisco: Morgan Kaufmann, 2002. 215 - 226.

[4] Babcock B, Datar M, Motwani R. Load shedding for aggregation queries over data streams[A]. ICDE 2004[C]. Los Alamitos: IEEE Computer Society, 2004. 350- 361.

[5] Arasu A, Babu S, Widom J. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations[EB/OL]. http://dbpubs.stanford.edu: 8090/pub/2002 - 57.

[6] Golab L, Özsu M T. Issues in Data Stream Management[J]. SIGMOD Record, 2003, 32(2): 5- 14.

[7] Ayad Ahmed, Naughton Jeffrey F. Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams [A]. Sigmod 2004[C]. New York: ACM Press, 2004. 419- 430.

[8] Abadi D J, Carney D, Cetintemel U, et al, Aurora: a new model and architecture for data stream management[J]. The VLDB Journal, 2003, 12(2): 120- 139.

[9] The STREAM Group. STREAM: the stanford stream data manager[J]. IEEE Data Engineering Bulletin, 2003, 26(1): 19- 26.

[10] Tatbul N, Çetintemel U, Zdonik S, et al. Load shedding in a data stream manager[A]. VLDB 2003[C]. San Fransisco: Morgan Kaufmann, 2003. 309- 320.

[11] Gedik B, Yu P, Bordawekar R. Executing stream joins on the cell processor[A]. VLDB 2007[C]. San Fransisco: Morgan Kaufmann, 2007. 363- 374.

[12] Govindaraju N, Lloyd B, Wang W, et al. Fast computation of database operations using graphics processors[A]. SIGMOD 2004[C]. New York: ACM Press, 2004. 215- 226.

[13] Bandi N, Sun C, Abbadi A, et al. Hardware acceleration in commercial databases: a Case study of spatial operations[A]. VLDB 2004[C]. San Fransisco: Morgan Kaufmann, 2004. 1021- 1032.

[14] Govindaraju N, Gray J, Kumar R, et al. GPUTeraSort: high performance graphics co-processor sorting for large database management[A]. ACM SIGMOD 2006[C]. New York: ACM Press, 2006. 325- 336.

[15] Brian G, Anastassia A, Larry H, et al. Accelerating database operators using a network processor[A]. First International Workshop on Data Management on New Hardware[C]. New York: ACM Press, 2005. 32- 36.

[16] Bandi N, Metwally A, Agrawal D, et al. Fast data stream algorithms using associative memories[A]. ACM SIGMOD 2007 [C]. New York: ACM Press, 2007. 247- 256.

[17] 刘学军, 胡平, 等. 基于硬件加速的高速数据流连续实时聚集查询[J]. 电子学报. 2007, 35, (2). 228- 233.
Liu Xuejun, Hu Ping, et al. Continual aggregation queries over high rate data streams based on hardware acceleration[J]. Acta Electronica Sinica, 2007, 35(2): 228- 233. (in Chinese)

[18] 曹锋, 周傲英. 基于图形处理器的数据流快速聚类[J]. 软件学报. 2007, 18(2): 291- 302.
Cao F, Zhou A. Fast clustering of data streams using graphics processors[J]. Journal of Software. 2007, 18(2): 291 - 302. (in Chinese)

[19] 钱江波, 徐宏炳, 等. 多数据流滑动窗口并发连接方法. 计算机研究与发展[J]. 2005, 42(10): 1771- 1778.
Qian J, Xu H, et al. Simultaneous sliding window join approach over multiple data streams. Journal of Computer Research and Development[J]. 2005, 42(10): 1771- 1778. (in Chinese)

作者简介:

钱江波　男, 1974 年出生, 副教授, 博士, 主要研究方向为数据库和数据流管理技术, 数据挖掘, 逻辑电路设计等.
Email: qianjiangbo@nbu.edu.cn.

王永利　男, 1974 年出生, 副教授, 博士, 主要研究方向为数据流管理, 数据挖掘, 模式识别等.
Email: wyl_seu@126.com.