# 基于硬件加速的高速数据流连续实时聚集查询

刘学军[1,2], 胡 平[1], 徐宏炳[2], 董逸生[2], 钱江波[2], 王永利[2]
(1. 南京工业大学信息科学与工程学院, 江苏南京 210009; 2. 东南大学计算机科学与工程学院, 江苏南京 210096)

摘 要: 近年来, 动态数据流环境下的聚集查询正成为一个热点研究问题. 目前的相关算法主要是采用近似技术, 以牺牲精度来换取处理速度的提高. 然而, 在高速数据流环境下, 处理速度仍然难以满足需求. 软硬件协同的高速数据流处理技术逐渐引起人们的关注. 提出了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. 算法是增量的, 也实现了多窗口资源共享. 最后, 给出了算法的复杂度分析并实验验证了方法的有效性.

关键词: 数据流; 聚集查询; 软硬件协同; 增量计算
中图分类号: TP311.13 文献标识码: A 文章编号: 0372-2112 (2007) 02-0228-06

# Continual Aggregation Queries over High Rate Data Streams Based on Hardware Acceleration

LIU Xue-jun[1,2], HU Ping[1], XU Hong-bing[2], DONG Yi-sheng[2], QIAN Jiang-bo[2], WANG Yong-li[2]
(1. *College of Information Science and Engineering, Nanjing University of Technology, Nanjing, Jiangsu* 210009, *China*;
2. *College of Computer Science and Engineering, Southeast University, Nanjing, Jiangsu* 210096, *China*)

**Abstract:** Recently there has been a growing interest in aggregation queries for scenarios in which data streams arrive at very high rates and a data stream system is registered with many simultaneous queries. In order to dealing with the huge amounts of data and increasingly stringent response-time requirements, Most existing work in this area has to adopt approximate technology which sacrifice aggregate veracity. But in the environment of high rate data streams, the processing rate still cannot satisfy requirements. So query processing based on hardware-software codesign has recently emerged as a viable solution for dealing with high rate data streams. In this paper, We propose a kind of novel aggregate query algorithms based on hardware-software codesign, which incorporate hardware advantage in processing rate and software long suit in agility. Many incremental computation approaches and resource sharing techniques in sliding-window aggregations are introduced. Lastly, time cost of the algorithm is analyzed and the experiment show the feasibility and effectiveness of the approach.

**Key words:** data streams; aggregation queries; hardware-software codesign; incremental computation

## 1 引言

近年来, 数据流模型的出现对传统的数据管理技术提出了巨大的挑战, 数据流管理正在成为一个热点研究领域. 聚集是数据流系统中的一种重要查询类型, 也常常是一些高级分析的基础. 面对大量、快速数据流, 目前已有的聚集算法普遍是采用近似技术, 减少数据处理量, 以牺牲精度来换取处理速度的提高. 文献[1,2]研究了传感器网络中的聚集查询问题, 重点是如何动态地建立路由树, 实现流水线聚集操作. 文献[3]利用随机草图技术, 提取数据流的概要, 减少数据的处理量来加快数据处理速度, 并提出了一种草图分割技术来提高算法的性能, 该算法是一种近似聚集查询算法. 文献[4]则采用了小波技术对数据流进行压缩提高数据处理速度, 实现了近似聚集查询. 文献[5]研究了数据流的时态聚集查询, 方法是将时间空间分割为若干段, 采用分层的时间粒度保存聚集结果, 每个时间段采用两棵 SB-Tree, 一棵保存起始时间小于给定时间的记录的聚集值, 另一棵保存截止时间小于给定时间的记录的聚集值, 通过两棵 SB-Tree 来实现数据流的时态聚集查询. 文献[6]则研究了数据流的相关聚集.

但是, 在高速数据流环境下, 上述方法难以应用. 软硬件协同的处理技术已经引起人们的关注. 文献[7]讨论了多媒体数据流处理器的调度问题, 通过适当的流调度策略来提高并发度, 降低通信带宽. Khailany 等人提出了一个图像流处理器

收稿日期: 2006-02-13; 修回日期: 2006-08-22
基金项目: 江苏省高技术项目(No. BG2004034); 江苏省 2004 年度研究生创新计划项目(No. xm04-36)

的体系结构,给出了若干关键实现技术[8].我们采用 *FPGA* 实现了高速数据流的连接操作[9].

本文结合数据流的特点,提出了一种软硬件协同的数据流聚集查询方法,可以显著地提高系统的处理速度,特别适合高速数据流环境,目前尚未见到有关研究报道.

## 2　相关定义

定义 **1**　设 *DS* 是数据流序列$\langle s_1, s_2, \cdots, s_k, \cdots\rangle$,其中 $k = 1, 2, \cdots$,将数据流 *DS* 分段,每一个分段对应一个数据流子序列,这样的一个数据分段称为一个基本窗口,记作 *BW*.

定义 **2**　一个滑动窗口 *SW* 对应一个连续的基本窗口序列$\langle BW_1, BW_2, \cdots, BW_k\rangle$,它所容纳的基本窗口的数目是一个定值 $k$.

随着新数据的到来,滑动窗口以基本窗口为单位不断更新.每进入一个新的基本窗口,最旧的一个基本窗口被删除,滑动窗口随之更新一次.因此,滑动窗口中包含的数据不断变化和更新.

## 3　软硬件协同的数据流实时聚集查询

### 3.1　求和(Sum)和平均值(Avg)的实现

设滑动窗口的长度为 $w$,包含了 $k$ 个基本窗口,每个基本窗口的长度为 $b$, $S[i] = s[(t - w) + (i - 1)b, (t - w) + ib - 1]$ 表示第 $i$ 个基本窗口(其中,$t$ 为当前时刻),$S[i;j]$ 表示 $S[i]$ 的第 $j$ 个值,则有:

对于每个基本窗口,$\Sigma(S[i]) = \Sigma_{j=1\text{ to }b} S[i;j]$,其增量计算式为:$\Sigma_{new}(S[i]) = \Sigma_{old}(S[i]) + S[i;b] - S[i;0]$.

$b$ 个数据点(或时间点)后,滑动窗口 Sum 聚集值 $\Sigma_{new}(s) = \Sigma_{old}(s) + \Sigma(S[k]) - \Sigma(S[0])$,初始 $\Sigma_{new}(s) = \Sigma_{i=1\text{ to }k} S[i]$.

由此可见,滑动窗口的 Sum 聚集值可以由基本窗口的 Sum 聚集值经过简单的运算得到.在高速数据流环境下,我们采用基于 FPGA 的硬件结构计算各个基本窗口的 Sum 聚集值,这无疑将大大提高聚集值的计算速度.对于基于元组的基本窗口,其实现原理如图 1 所示,图中,fifozlsum、fifozlsumclock 和 fifozlsumwr 分别为数据输入、数据输入控制时钟和时钟使能控制端.

计数器 inst5 和比较器 inst9 起计时功能,决定了基本窗口的大小,利用比较器的输入 *slidingcount* 可以动态地调整基本窗口的大小.例如:*slidingcount* 的值为 20,则基本窗口的大小为可以容纳 20 个元组.累加器 inst2 对初始基本窗口内的数据求和,此时数据只入不出.一旦基本窗口已满,累加器 inst2 停止工作,先进先出队列 inst1 的读使能变为 1,数据从队列 $q$ 端输出并被 D 触发器 inst8 存储,同时,触发器 inst6 的时钟有效,开始存储进入队列的数据.进入队列的数据和离开队列的数据相减,再利用累加器 inst13 累加求和,然后,与累加器 inst2 的数值相加求和就得到了基本窗口内的 Sum 值.上述过程是一个利用硬件的增量计算过程.输出端口 result 连续输出基本窗口内的 Sum 值,而 sumresult 则每间隔一个基本窗口输出一次 Sum 值.对于基于时间的基本窗口,其实现过程类似,但是,有 3 点重要不同:(1) 计数器 inst5 和比较器 inst9 起计时作用,而不是起计数作用;(2) 计数器 inst5 的时钟使能端不再受

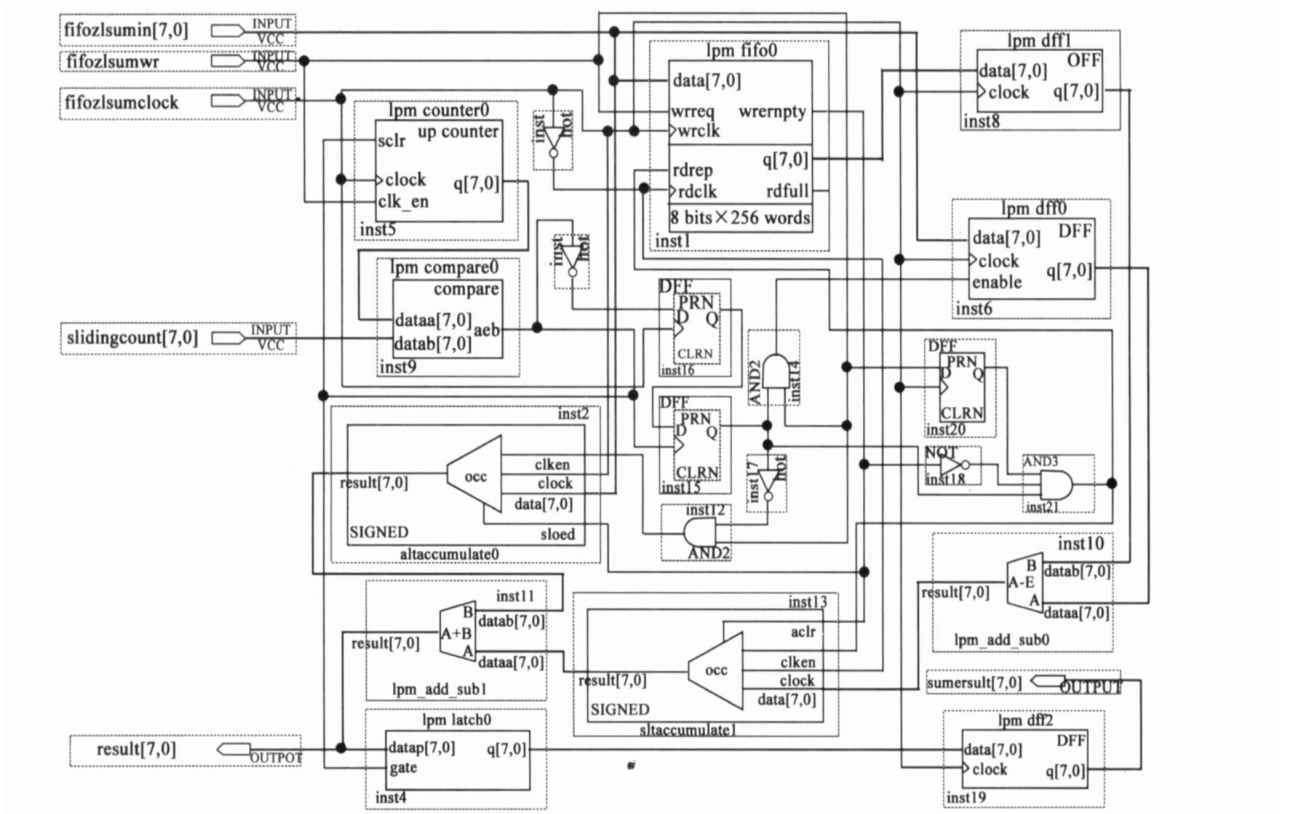



图 1　基本窗口的 Sum 实现

到队列输入使能控制,即便基本窗口没有数据输入,计数器也照常工作并计时;(3)经过一个基本窗口时间后,即使没有新数据进入,旧数据也不断从基本窗口内滑出,不必控制数据按照出去一个、进入一个顺序进行.

实际上,我们也可以根据需要将基于时间的窗口转化为基于元组的窗口后,再利用图 1 所示的结构来计算基于时间的基本窗口的聚集值.

采用上述结构,每经过一个基本窗口时间,按照如下算法更新各个基本窗口和滑动窗口的 Sum 聚集值.

算法 **1** *UpdateSum*,更新各个基本窗口和滑动窗口的 Sum 聚集值.

输入: *sumresult*, *k*. *sumresult* 表示图 1 输出端口 *sumresult* 的输出值, *k* 表示滑动窗口包含的基本窗口数量.

输出:更新后的各个基本窗口和滑动窗口的 Sum 聚集值.

*UpdateSum*(*sumresult*, *k*)

(1) for( $i=0; i<k; i++$ )

(2) $\Sigma(S[i])=\Sigma(S[i+1]);$

(3) $\Sigma(S[k])=$ *sumresult*;

(4) $\Sigma(s)=\Sigma(s)+\Sigma(S[k])-\Sigma(S[0])$

为了计算平均值 Avg,需要保存 Sum 和 Count 两种聚集操作的值,即数据对(Sum, Count). 滑动窗口的 Avg 聚集值可以由基本窗口的 Sum 聚集值和 Count 聚集值经过简单的运算获得. 基于元组的基本窗口由于窗口内元组数不变,只需在采用图 1 的结构计算 Sum 聚集值的同时输出窗口内元组数,就可以得到数据对(Sum, Count). 而基于时间的基本窗口由于窗口内元组数可能发生变化,因此,需要专门的电路来统计窗口内元组的数量,可以采用两个计数器分别对进入和离开窗口的数据进行计数,两者之差就是窗口内的元组计数.

### 3.2 最大值(Max)的实现

若一个滑动窗口 $s$ 由 $k$ 个基本窗口组成, $\mathrm{Max}(S[i])$ 表示第 $i$ 个基本窗口的最大值, $\mathrm{Max}(s)$ 表示滑动窗口的最大值,则 $\mathrm{Max}(s)=\mathrm{Max}(\mathrm{Max}(S[1]),\mathrm{Max}(S[1]),\cdots,\mathrm{Max}(S[k]))$,即 $k$ 个基本窗口的最大值. 我们采用如图 2 所示的硬件结构来计算基本窗口的 Max 聚集值.

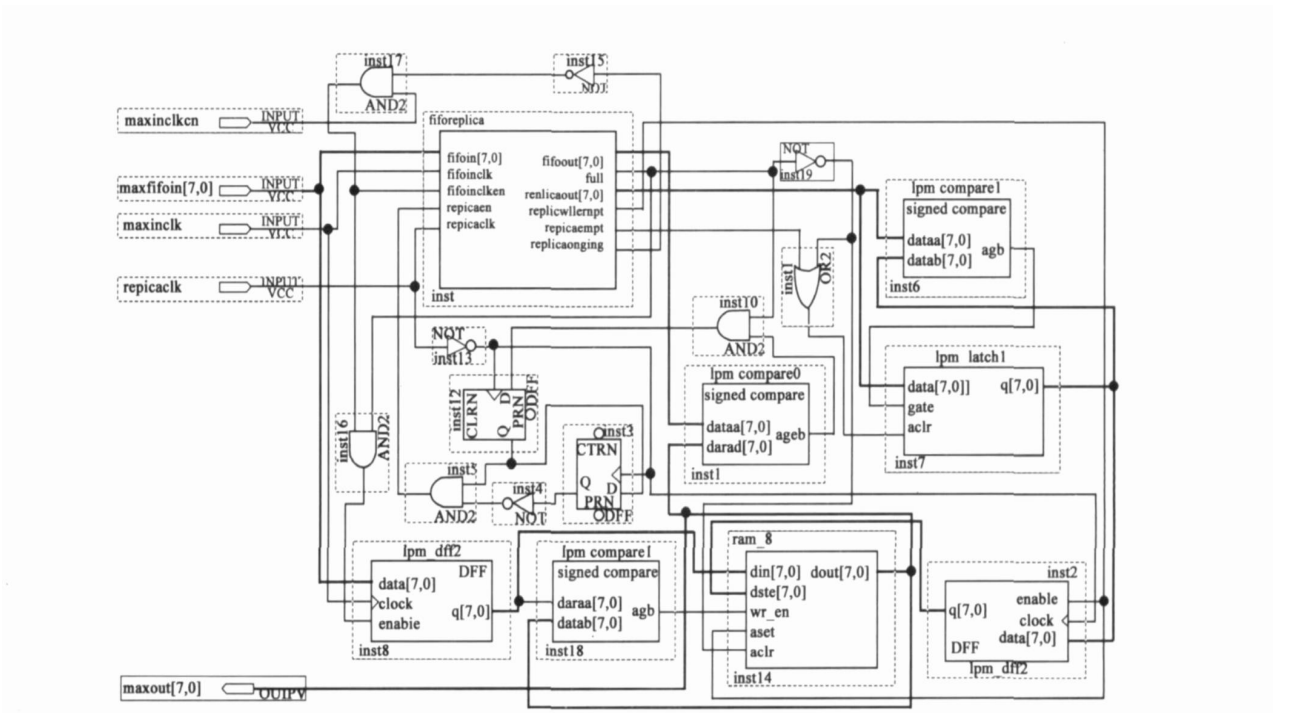



图 2 基本窗口的 Max 实现

Inst 是一个具有复制功能的先进先出队列,由两组寄存器组构成,为表述方便,分别记为 fifowindow 和 fiforeplication. fifowindow 相当于一个基本窗口,复制窗口 fiforeplication 存储由 fifowindow 复制来的数据. Inst14 保存了基本窗口的最大值. 每进入窗口一个新数据,被 D 触发器 inst8 存储,并利用比较器 inst18 与保存在 inst14 中的最大值比较,如果大于后者,则更新最大值. 窗口由 fifoout 端口输出数据,每输出一个数据,利用比较器 inst1 和最大值比较,如果等于最大值,说明 inst14 中保存的最大值离开了窗口,需要重新计算窗口的最大值. 此时, inst 的 replicaen 端口为 1, fifowindow 中不再进入新数据,也不再有旧数据离开,启动复制功能,将 fifowindow 中保存的数据复制到 fiforeplication,一个时钟(replicaclk)内完成复制. fiforeplication 内的数据在时钟 replicaclk 控制下,由 inst 的 replicaout 端输出,通过比较器 inst6 和锁存器 inst7 得到了 fiforeplication 中的最大值,即基本窗口中的最大值. 输出端 replicwillempt 为 1 时将锁存器 inst7 锁存的数据保存到 inst14 中. 接下来, replicaen 端口变为 0, fifowindow 再次允许新数据进入和旧数据离开. 当输出端 replicaempt 为 1,对锁存器 inst7 进

行清零, 而 replicaongoing 为 1 时, 表明正在对复制窗口 fr foreplication 内的数据进行比较, 此时阻止新数据进入基本窗口. 图 2 中, maxfifoin 为数据输入端, maxinclk 为控制数据输入的时钟(或数据采样周期), maxinclken 为 maxinclk 时钟使能控制端, replicaclk 是控制复制和 inst 的 replicaout 端输出的时钟. 通常, replicaclk 的频率应远远高于 maxinclk 的频率. 输出端 maxout 连续输出了滑动窗口的最大值.

采用上述结构, 每经过一个基本窗口间隔, 按照如下算法更新各个基本窗口和滑动窗口的 Max 聚集值.

算法 **2**　*UpdateMax*, 更新各个基本窗口和滑动窗口的 Max 聚集值.

输入: *Maxout*, *k*. *Maxout* 表示图 2 输出端口 *Maxout* 的输出值, *k* 表示滑动窗口包含的基本窗口数量.

输出: 更新后的各个基本窗口和滑动窗口的 Max 聚集值.

*UpdateMax*(*Maxout*, *k*)

(1) for( *i*= 0; *i*< *k*; *i*++ )

(2)　Max(*S*[*i*])= Max(*S*[*i*+ 1]);

(3) Max(*S*[*k*])= *Maxout*;

(4) if(Max(*S*[*k*])> Max(*s*))

(5)　Max(*s*)= Max(*S*[*k*]);

(6) if(Max(*S*[0])⩾Max(*s*))

Max(*s*)= Max(Max(*S*[1]), Max(*S*[2]), …, Max(*S*[*k*]))

// 重新计算滑动窗口的最大值.

滑动窗口的最小值计算类似最大值的计算, 由于篇幅限制, 这里不再多述.

### 3.3　聚集的多窗口共享

在一些数据流应用系统中, 往往包含着对同一数据源的大量并发查询, 独立地处理各个查询往往是低效率的, 有时甚至难以满足实时要求. 因此, 利用查询的相似性进行批量处理, 提高资源的共享就变得尤为重要. 这里, 我们主要研究基于基本窗口的共享. 多个查询共用同一种基本窗口也可以减少硬件资源的消耗. 设有 *n* 个查询可以窗口共享, 它们的有效基本窗口数为 *m*, 按照时间由近及远的次序, 有效基本窗口分别为 *S*[*m*]、*S*[*m* - 1]、…、*S*[0], *S*[0] 表示刚由有效基本窗口变为无效基本窗口的窗口. 可以采用算法 3 更新各个可共享的滑动窗口聚集值.

算法 **3**　*UpdateAgg*, 更新各个基本窗口和各个滑动窗口的聚集值.

输入: *Hardwarevalue*, *m*. *Hardwarevalue* 表示每隔一个基本窗口时间上述相关硬件的输出值, *m* 表示有效的基本窗口数量.

输出: 更新后的各个基本窗口和各个滑动窗口的聚集值.

*UpdateAgg*(*Hardwarevalue*, *m*)

(1) for( *i*= 0; *i*< *m*; *i*++ )

(2)　*f*(*S*[*i*])= *f*(*S*[*i*+ 1]);　//*f*(*S*[*i*]) 表示对第 *i* 个基本窗口求聚集值.

(3) *f*(*S*[*k*])= *Hardwarevalue*;

(4) 根据各个滑动窗口所包含的基本窗口, 分别更新各个滑动窗口聚集值.

当并发查询数很多时, 分别更新各个滑动窗口将花费较多时间, 针对这种情况, 我们提出了如下改进的算法.

*UpdateAggtree*(*Hardwarevalue*, *m*)

(1) if(*T* is *null*) {

(2)　以 *S*[*i*]为叶结点, 建立一棵满二叉聚集树 *T*;}

(3) else {

(4)　for( *i*= 0; *i*< *m*; *i*++ )

(5)　　*f*(*S*[*i*])= *f*(*S*[*i*+ 1]);

(6)　*f*(*S*[*k*])= *Hardwarevalue*;

(7)　调用 *accumulate*(*Lchildval*, *Rchildval*), 更新树 *T* 中的各个非叶结点聚集值; }　// *Lchildval* 和 *Rchildval* 分别表示左孩子的聚集值和右孩子的聚集值.

(8) 各个滑动窗口查询聚集树 *T*, 得到各自的聚集值;

对于不同的聚集操作, *accumulate* 函数有如下不同的实现方式:

(*a*) 如果聚集操作是 Sum 或者 Count, 则 *accumulate*(*x*, *y*) 的实现方式为 *accumulate*(*x*, *y*)= *x*+ *y*.

(*b*) 如果聚集操作为 Max, 则 *accumulate*(*x*, *y*) 的实现方式为 *accumulate*(*x*, *y*)= *Max*(*x*, *y*).

(*c*) 如果聚集操作为 Min, 则 *accumulate*(*x*, *y*) 的实现方式为 *accumulate*(*x*, *y*)= Min(*x*, *y*).

(*d*) 如果聚集操作为 Avg, 则需要存储 Sum 与 Count 两种聚集操作的聚集值, *accumulate*((*xsum*, *xcount*), (*ysum*, *ycount*)) 的实现方式为 *accumulate*((*xsum*, *xcount*), (*ysum*, *ycount*)) = ((*xsum*+ *ysum*), (*xcount*+ *ycount*)).

(*e*) 如果 *x*= *NULL*, 对于任何 *y*, 都有 *accumulate*(*x*, *y*)= *y*.

性质 **1**　在满二叉聚集树 *T* 的第 *i* 层上有 $2^{i-1}$ 个节点(*i* ⩾1).

证明　利用归纳法证明此性质.

当 *i*= 1 时, 只有一个根节点, 显然, $2^{i-1}=2^0=1$ 是成立的.

由归纳假设, 第 *i*- 1 层上有 $2^{i-2}$ 个节点成立. 由于满二叉树的每个节点的度为 2, 所以在第 *i* 层上的节点数为第 *i*- 1 层上的节点数的 2 倍, 即 $2*2^{i-2}=2^{i-1}$ 成立, 命题得证.

性质 **2**　最大层有 *m* 个节点的聚集树 *T* 有 2* *m*- 1 个节点.

证明　设 *m*、$n_2$ 分别为叶节点数、度为 2 的节点数, 由于满二叉聚集树 *T* 中除叶节点外, 其余节点的度均等于 2, 所以, 节点总数为 *n*= *m*+ $n_2$(式 1). 除根节点外, 其余节点都有一个分支进入, 设 *B* 为分支总数, 则 *n*= *B*+ 1, 又由于这些分支是由度为 2 的节点射出的, 所以有 *B*= 2* $n_2$, 于是得到 *n*= 2* $n_2$+ 1(式 2). 根据式 1 和式 2, $n_2$= *m*- 1, 所以 *n*= 2* *m*- 1, 命题得证.

性质 **3**　最大层有 *m* 个节点的聚集树 *T* 的深度为 $\log_2 m$ + 1.

证明　聚集树 *T* 的深度等于最大层的层次数, 设最大层所处的层次为 *h*, 根据性质 1 有: *m*= $2^{h-1}$, 所以 $\log_2 m$= *h*- 1, 即 *h*= $\log_2 m$+ 1, 命题得证.

对于一棵满二叉聚集树 $T$, 其叶节点数需要满足性质 1 的要求, 如果有效基本窗口数 $m$ 不满足上述要求, 可以通过补充一个或多个节点来满足性质 1 的要求, 这些节点的聚集值为 *NULL*.

根据上述算法, 如果一个滑动窗口包含了从 $S[1]$ 到 $S[m]$ 的 $m$ 个基本窗口, 只要检索聚集树 $T$ 的根结点就得到了滑动窗口的聚集值, 计算效率大大提高.

带有连接操作的聚集计算, 首先进行连接操作, 再进行聚集操作. 连接的硬件实现见我们在文献[9]中提出的基于硬件加速的数据流连接算法, 这里不再多述.

## 4 算法分析和实验验证

### 4.1 算法的时间复杂性分析

算法 UpdateSum 的时间主要花费在 $k$ 个基本窗口的更新方面, 因此, 时间复杂度为 $O(k)$. UpdateMax 算法更新 $k$ 个基本窗口的语句频度为 $k$, 当离开滑动窗口的数据等于最大值时, 则重新计算滑动窗口的最大值, 需要 $k$ 次比较, 因此, 时间复杂度也为 $O(k)$. 算法 UpdateAgg 更新 $m$ 个基本窗口的语句频度为 $m$, 时间复杂度为 $O(m)$, $n$ 个查询(设每个滑动窗口平均包含 $k$ 个基本窗口) 分别更新各自的聚集值, 最坏时间复杂度为 $O(k*n)$, 因此, 当 $k*n>m$ 时, 算法时间复杂度为 $O(k*n)$, 否则, 可认为算法时间复杂度为 $O(m)$. 根据性质 2, 对于一棵有 $m$ 个叶结点的满二叉聚集树, 其总结点数为 $2*m-1$, 因此, UpdateAggtree 算法更新聚集树的时间复杂度为 $O(m)$, 每个查询检索聚集树得到它的聚集值, 根据性质 3, 最坏的时间复杂度为 $O(\log_2 m)$, 最好的时间复杂度为 $O(1)$.

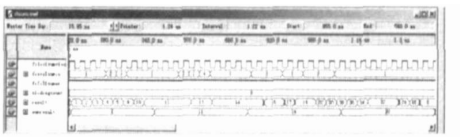
设硬件的时钟周期为 $T$, 每个基本窗口内有 $w$ 个元组, 则基本窗口内一个元组平均执行时间如下: (1) Sum 聚集: 在一个时钟周期内完成了元组的写入和累计计算, 因此, 一个元组平均执行时间为 $T$; (2) Avg 聚集: 同 Sum 聚集类似, 一个元组平均执行时间也为 $T$; (3) Max 聚集: 在计算 Max 时, 如果离开窗口的元组属性值小于保存的最大值, 只需丢掉该元组并读入一个新元组即可, 若元组输入时钟周期为 $t$, 则一个元组平均执行时间为 $t$; 否则, 需要重新计算基本窗口的最大值, 需要时间为 $w*(T+2)$, 此过程中, 新元组不可以进入基本窗口. 通常取 $t$ 远远大于 $T$.

### 4.2 实验仿真

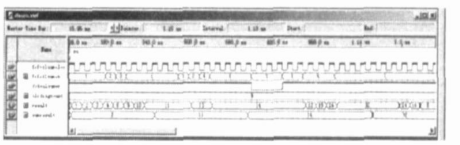
我们采用 FPGA 实现了上述聚集计算的硬件结构, 在 Altera 公司的 QuartusII Web Edition 集成环境中进行了设计和系统仿真. 图 3 是求和(Sum) 的仿真波形图, 窗口的大小受 slidingcount 值控制, 本例中含有 8 个元组. 我们也分别对数据输入写使能始终为 1 和某段时间不为 1 两种情况进行了仿真. 图 4 是最大值(Max) 的仿真波形, 窗口包含了 4 个元组. 从图中可以看出满足设计要求, 仿真结果是完全正确的.

### 4.3 实验对比分析

STREAM1 系统是一个典型的数据流管理系统, 我们将本文提出的聚集方法和 STREAM 系统的处理性能进行了比较. 实验注册了 50 条查询, 最大查询窗口容纳 1600 个元组, 最小查询窗口容纳 200 个元组, 基本窗口的大小分别取 100 个元组和 200 个元组. 元组属性值在[0, 255]之间平均随机分布.



(a) 写使能都为 1 时的 Sum 仿真波形



(b) 写使能不全为 1 时的 Sum 仿真波形

图 3 Sum 仿真波形图

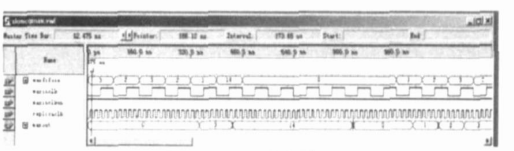


图 4 Max 仿真波形

STREAM 系统执行查询时, 用 STREAM 命令"*gen_client-l[logfile]-c[configfile][script-file]*" 记录其运行日志文件中的查询处理时间. STREAM 的参数都是缺省的, 我们仅将"*stream-0.6.0/config*" 最后一行的 *RUN_TIME* 从 1000 到 100000. 我们用 FPGA 芯片 Stratix EP1S10F780C5 实现本文提出的硬件结构, 最高运行频率 78MHz. 我们将 50000 个元组存于 FPGA 的 RAM 中, 循环读取和处理这些元组, 每次循环的起始元组是不同的. 软件实验平台是 Intel 赛扬 1GHz CPU, 内存 128MB, Redhat9.0 操作系统.

图 5 是不同方法处理速度的比较, 其中 AGGW1 和 AGGW2 分别表示采用软硬件协同聚集且基本窗口分别是 200 和 100 的情况, 从图中可以看出本文提出的聚集方法比 STREAM 系统的执行速度快得多. 这是由于软硬件协同聚集的基本窗口采用硬件结构计算聚集, 而软件部分不仅考虑了查询的共享, 还体现了增量计算的思想. 在一些重要应用领域, 处理速度的提高是有重要意义的. 从图 5 也可以看出, 随着基本窗口的增大, 软硬件协同聚集方法表现出更好的性能.

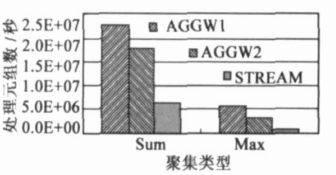



图 5 不同方法处理速度的比较

## 5 结束语

在高速数据流环境下, 提高数据的处理速度成为系统的关键. 已有的聚集算法基本上都是通过采用近似技术, 以牺牲精度来换取速度的提高. 随着硬件技术的快速发展和硬件成本的迅速下降, 软硬件协同技术逐渐引起了人们的关注. 本文提出了一种基于软硬件协同的增量聚集算法, 即发挥了硬件在处理速度上的优势, 又发挥了软件在灵活性方面的长处. 由硬件实现基本窗口的聚集值, 多个查询窗口共享同一种基本窗口, 即可以减少硬件资源的消耗, 又可以提高查询的资源共

享.本文提出的数据流聚集方法将应用于我们的基于硬件预处理的数据流管理系统中,一些基本查询操作,如连接[9]、选择、投影和聚集等将在硬件前端部分实现或全部实现,这将大大提高数据流的处理速度.

参考文献:


[1] S R Madden, M J Franklin, J M Hellerstein, W Hong. TAG: a tiny AGgregation service for ad-hoc sensor networks[A]. In Fifth Symposium on Operating Systems Design and Implementation[C]. Boston, MA, 2002. 131- 146.

[2] Madden SR, Szewczyk R, Franklin MJ, Culler D. Supporting aggregate queries over ad-hoc wireless sensor networks[A]. In: Kindberg T, ed. Proceedings of the Fourth Workshop on Mobile Computing and Systems Applications[C]. Los Alamitos: IEEE Computer Press, 2002. 49- 58.

[3] Alin Dobra, Minos Garofalakis, Johannes Gehrke, Rajeev Rastogi. Processing complex aggregate queries over data streams [A]. In Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data[C]. Madison, Wisconsin, 2002. 61- 72.

[4] A C Gilbert, Y Kotidis, S Muthukrishnan, M Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries[A]. In Proceedings of the 27th Conference on Very Large Data Bases[C]. Roma, Italy, 2001. 79- 88.

[5] D Zhang, D Gunopulos, V J Tsotras B Seeger. Temporal aggregation over data streams using multiple granularities[A]. In Proceeding of International Conference on Extending Database Technology[C]. Prague, Czech Republic, 2002. 646- 663.

[6] J Gehrke, F Korn, D Srivastava. On computing correlated aggregates over continual data streams[A]. In Proceedings of the ACM SIGMOD Conference[C]. Santa Barbara, CA, USA, 2001. 13- 24.

[7] Ujval J Kapasi, Peter Mattson, William J Dally, John D Owens, Brian Towles. Stream scheduling[A]. EE482C Advanced Computer Organization: Stream Processor Architecture, Spring 2001/2002. In Proceedings of the 3rd Workshop on Media and Streaming Processors[C]. Austin, Texas, 2001. 101- 106.

[8] B Khailany, WJ Dally, UJ Kapasi, P Mattson, J Namkoong, JD Owens, B Towles, A Chang, S Rixner. Imagine: media processing with streams[J]. IEEE Micro, 2001, 21(2): 35- 46.

[9] J-B Qian, H-B Xu, Y-S Dong, X-J Liu, Y-L Wang. FPGA acceleration window joins over multiple data streams[J]. Journal of Circuits, Systems, and Computers, 2005, 14(4): 813- 830.


作者简介:

刘学军　男,1971 年生,博士研究生,主要研究兴趣为数据流管理、软硬件协同设计、数据挖掘等. E-mail: kj-njgd@163.com

胡　平　男,1962 年生,副教授,研究领域包括嵌入式系统、智能诊断和信息处理等.

徐宏炳　男,1947 年生,教授,研究领域包括数据库、ASIC 的开发与应用、实时数据采集与信息处理等.

董逸生　男,1940 年生,教授,博士生导师,研究领域包括数据库、信息系统和软件工程等.