计算机科学2004Vol. 31№. 9

一维可重构流水线总线并行机上平面点集的凸壳算法*)

周世泉　许胤龙　陈国良　赵建勇

(中国科学技术大学计算机系　国家高性能计算中心　合肥230027)

摘　要　确定平面点集的凸壳是计算几何中的一个基本问题。一维可重构流水线总线并行机是近年提出的一种采用光连接的并行计算模型。本文在规模为 n 的可重构流水线总线并行机上提出了一个计算 n 个平面点的凸壳算法，当 n 个点按横坐标递增的顺序存储时，该算法的时间复杂度为 $O(\log n)$。

关键词　凸壳，并行算法，可重构流水线总线

A Convex Hull Algorithm for a Planer Point Set on LARPBS

ZHOU Shi-Quan　XU Yin-Long　CHEN Guo-Liang　ZHAO Jian-Yong

(Department of Computer Science & Technology, University of Science and Technology of China, National High Performance Computing Center Hefei 230027)

Abstract　Computing the convex hull of a given set of planar points is one of the most extensively investigated topics in computational geometry. A linear array with a reconfigurable pipelined bus system (LARPBS) is one of the recently proposed parallel architectures based on optical buses. This paper presents a parallel algorithm for convex hull computation of a given set of n planar points on LARPBS with n processors. In the case of the given n points being stored in LARPBS according to the alphabet order of their coordinates, one per processor, the algorithm runs in time $O(\log n)$.

Keywords　Convex hull, Reconfigurable pipelined bus, Parallel algorithms

1　引言

给定平面点集 S，它的凸壳是指包含 S 的最小凸多边形。在计算机图形学、图像处理、模式识别和运筹学等领域都需要处理凸壳问题。凸壳的计算在计算几何中占有重要位置。F. P. Preparata[1]等人于1985年指出，求 n 个平面点凸壳的串行时间复杂度为 $\Theta(n\log n)$。Tatsuya Hayashi 于1998年在RMESH上提出了成本最优的并行算法。当 n 个点按横坐标递增排序且以邻近序存储在 $\frac{\sqrt{n}}{\log\log n}\times\frac{\sqrt{n}}{\log\log n}$ 的RMESH上时，算法的时间为 $O((\log\log n)^2)$。

在一维可重构流水线总线并行机(A Linear Array with a Reconfigurable Pipelined Bus System, LARPBS)[4]上，各处理器间通过一根基于光交换、可重构的光总线互连，其拓扑结构可以在算法的执行过程中动态地改变。该模型同时具有总线结构广播和非总线结构的点到点通信优势，借此增强并行机的通信能力，达到加速算法的目的。在这种模型上已经研究了不少领域的基本问题，如图论、计算几何、图像处理等[4~7]。但是对于求平面点集的凸壳，在LARPBS结构上还没有任何结果。

F. P. Preparata 等人于1977年在文[8]中首次把分治技术应用于凸壳问题，基本思路如下：把 S 分成两个大小近似相等的子集 S_1 和 S_2，分别递归地计算 S_1 和 S_2 的凸壳，记为 P_1、P_2，最后找 $P_1\cup P_2$ 的凸壳。如何快速合并两个子凸壳是提高分治法求凸壳效率的关键。本文采用类似的思想，通过求两个子凸壳间的切线实现合并。假设点集 S 中任意三点不共线且任意两点的横坐标不相等，按点的横坐标递增的顺序划分集合，所得的两个子集的凸壳是分离的，它们之间的切线能通过并行计算点到凸壳的切线得到。文中先提出在 n 个处理器的LARPBS上常数时间内计算点到凸壳的切线算法，用 n^2 个处理器并行执行该算法，在相同时间内得到两凸壳间的切线。然后采用与LARPBS相适应的采样技术，使得当处理器数减少为 n 时，在O(1)时间内实现合并。凸壳算法分为 $\log n$ 步，第 d 步并行地实现 $n/2^d$ 个凸壳对的合并，在常数时间内完成，故算法时间复杂度为 $O(\log n)$。就我们所知，这是第一个在LARPBS上求平面点集的凸壳算法。

本文第2节介绍LARPBS计算模型和与算法相关的基本概念；第3节介绍切线算法；第4节利用切线算法，得到常数时间的合并算法，由此推出对数时间的凸壳算法；最后总结全文。

2　预备知识

本节介绍LARPBS计算模型及与算法相关的计算几何概念。

2.1　一维可重构流水线总线并行机

在LARPBS中，n 个处理器通过一根单向环绕的光总线相连，相邻处理器间的距离相等。假定相邻处理器间单位长数据的通信延时为 τ，则某个处理器向总线写数据后，延时 τ、2τ、3τ、…后分别到达其后的第一、第二、第三、…个处理器。n 个处理器的LARPBS的循环周期为 $n\tau$，一个循环周期可分为 n 个时间片，通常假定一个处理器在一个循环周期内只能在一个时间片读写数据。LARPBS上的算法通常以循环周期 $n\tau$ 作为时间复杂度的基本度量单位，尽管这会随着处理器数 n 而变化，但就目前的处理器速度，对于上千的处理器数 n，$n\tau$ 与一个CPU操作(如加法与比较)的时间相当，所以这是一个合理的假定。K. Li 等人于1998年在文[5]中对此模型有详细介绍。模型见图1。

*)基金项目：本文工作受国家863项目"SMP机群体系结构上并行算法的研究与实现"(No:2001AA111041)的资助。

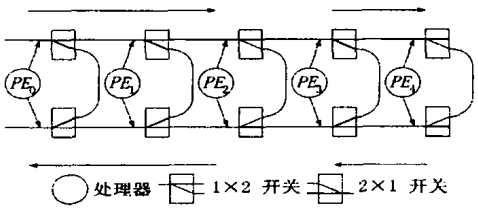

图1 LARPBS模型

本算法中用到了LARPBS上的如下基本操作，详见文[6]。

·分段：相邻处理器间有重构开关，这些开关可动态地置成开或关，将LARPBS分成一些相互独立的子段；

·广播：源处理器 PE_i 向其余 $n-1$ 个处理器发送数据；

·点对点通信：发送处理器 $PE_{i_1}, PE_{i_2}, \cdots, PE_{i_m}$，接收处理器 $PE_{k_1}, PE_{k_2}, \cdots, PE_{k_m}$。$PE_{i_j}$ 发送数据给 $PE_{k_j}(1 \leqslant j \leqslant m)$，且这 m 个数据可同时发送。

·多播：源处理器向 m 个处理器发送数据，$m<n$。

·多重多播：有 g 个不相交的接收组，$G_k=\{PE_{j_{k,1}}, PE_{j_{k,2}}, \cdots\}$，$1 \leqslant k \leqslant g$，对应地，有 g 个发送处理器 $PE_{i_1}, PE_{i_2}, \cdots, PE_{i_g}$。$PE_{i_k}$ 向 G_k 内所有处理器发送数据，这 g 个数据可同时发送。

·逻辑与：处理器 PE_i 存储逻辑位 $b_i(0 \leqslant i \leqslant n-1)$，计算 $b_0 \wedge \cdots \wedge b_{n-1}$ 并存入处理器 PE_0。

K. Li 等人于1998年在文[5]中给出了如下结论：

引理1 分段、点对点通信、广播、多播、多重多播以及逻辑与在LARPBS模型上均在 $O(1)$ 时间内完成。

2.2 相关概念

平面点集的凸壳是一个封闭的并且包含点集内所有点的最小区域。在计算几何中，凸壳理解为上述的域及其边界。

简单多边形 P 是平面上一个封闭且连通的区域，P 的边界由一些线段首尾相连成一条封闭的曲线，且不相邻的线段不相交。如果 P 的内角均小于180度则称为凸多边形。平面点集的凸壳是一个凸多边形。而凸多边形通常用边界上顶点的顺时针(逆时针)顺序来表示。

设凸壳 CH 的最左、最右两点分别为 v_l、v_r，那么 v_l、v_r 将凸壳的边界分为两条凸形链，其中位于 v_l、v_r 连线上方的称为上凸壳，下方的称为下凸壳，见图2。

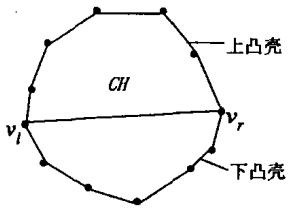

图2 上凸壳与下凸壳

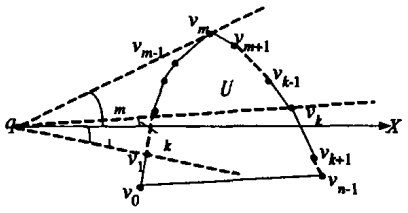

图3 点到上凸壳的切线

若点 q 在凸壳 CH 的最左点的左边或在最右点的右边，则 q 与 CH 左右分离，分别称 q 在 CH 的左侧或右侧。相应地，该点与上凸壳和下凸壳也是分离的。

设点 q 与凸壳 CH 左右分离，V 是 CH 的顶点集，则存在 $v_m \in V$，使得 V 中其余点均在直线 qv_m 的下方，称 qv_m 为 q 到 CH 的上切线，v_m 为上切点。显然，上切点属于上凸壳，故也称上切线为 q 与 CH 的上凸壳间的切线。如图3，切线为直线 qv_m。q 到 CH 下切线的定义类似。

·在图3中，以 q 为原点向右引一条水平射线作为 X 轴，设 v_i 是上凸壳 U 上的任意一个顶点，定义 θ_i 为射线 qv_i 同 X 轴正向的夹角，当 θ_i 的另一条边在 X 轴的上方时，$\theta_i>0$，反之 $\theta_i<0$。在图3中，$\theta_1<0<\theta_k<\theta_m$。显然，上切线所夹的角最大。

·设两个凸壳 CH_1、CH_2，若 CH_1 的最右点在 CH_2 的最左点的左边，那么 CH_1 与 CH_2 左右分离，称 CH_1 在 CH_2 的左侧，CH_2 在 CH_1 的右侧。对应的两个上凸壳和两个下凸壳间也相互分离。

·设两个分离的凸壳为 CH_1、CH_2，V_1、V_2 分别是它们的顶点集，则存在 $u_\alpha \in V_1$、$v_\beta \in V_2$，使得 V_1 和 V_2 中其余点均在直线 $u_\alpha v_\beta$ 的下方，称 $u_\alpha v_\beta$ 为 CH_1 与 CH_2 的上切线，u_α、v_β 为上切点。

显然，两个上切点分别属于两个上凸壳，故也称上切线为两上凸壳间的切线，如图4，上凸壳 U、V 间的切线为直线 $u_\alpha v_\beta$。两个分离凸壳的下切线的定义类似。

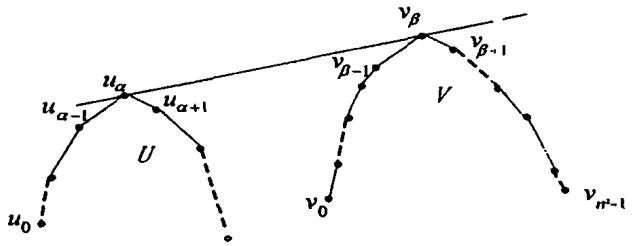

图4 u 两上凸壳间的切线

以上各切线均由两点确定，为了后文的叙述方便，称横坐标较小的那个点为左切点，另一个为右切点。

由上述定义可知：若两上凸壳 U、V 是分离的，则 U 上的每个点与 V 均分离。为求得 U、V 之间的切线，可对 U 上的每个点求到 V 的切线，这些切线中仅有一条与 U 也相切，那么该切线就是 U 与 V 间的切线。用同样的方法可以求得两个分离的下凸壳间的切线，得到第3节的切线算法。

3 切线算法

在介绍两上凸壳间的切线算法之前，先讨论如何计算上凸壳外一点到它的切线。设上凸壳 U 的顶点按横坐标递增的顺序依次为 $v_0, v_1, \cdots, v_{n-1}$，$q$ 是 U 左侧一点，假定集合 $U \cup \{q\}$ 中无三点共线。LARPBS有 n 个处理器，依次记为 PE_0，$PE_1, \cdots, PE_{n-1}$。我们计算每个点对应的夹角 $\theta_0, \theta_1, \cdots, \theta_{n-1}$，按照夹角的定义，这是一个先递增后递减的队列，最大值对应的是切点，即只有切点对应的夹角大于它的左右相邻顶点对应的夹角。据此，我们得到算法1，图3给出了它的示意图。

算法1 计算上凸壳左侧一点到它的切线

输入：处理器 PE_i 存储点 v_i 的坐标 $(x_i, y_i)(0<i<n-1)$，处理器 PE_0 存储 q 点的坐标

输出：处理器 PE_0 存储 q 到 U 的切线及切点

1. 处理器 PE_0 广播 q。

2. 处理器 $PE_i(0 \leqslant i \leqslant n-1)$ 计算 $\sin\theta_i$，处理器 $PE_i(0 \leqslant i < n-1)$ 向处理器 PE_{i+1} 播送结果，处理器 $PE_i(0 < i \leqslant n-1)$ 向处理器 PE_{i-1} 播送结果，则每个处理器存储了相邻夹角的 sin 值。

3. 处理器 $PE_i(0<i<n-1)$ 判断是否 $\sin\theta_{i-1}$ 和 $\sin\theta_{i+1}$ 均小于 $\sin\theta_i$。若处理器 PE_j 满足，则 v_j 为切点，PE_j 向处理器 PE_0 播送 v_j。

由引理1，以上操作均在常数时间内完成，得到以下定理：

定理1 给定 n 个顶点的上凸壳 U，其顶点按横坐标递增顺序依次存放在 n 个处理器的 LARPBS 上，每个处理器存储一个点，q 为 U 外一点，可用常数时间确定 q 到 U 的切线。

给定两个分离的上凸壳 U 和 V，为求得它们之间的切线，可利用算法1对 U 上每个点计算到 V 的切线，将两上凸壳间的切线问题转化成若干个计算点到上凸壳切线的子问题来解决。U 上的每个顶点对应一个子问题，对不同的子问题的计算是相互独立的。算法在 LARPBS 上实现时，我们利用该模型的可重构性，给每个子问题分配一段子 LARPBS，就能并行处理，减少了计算时间。

设 U 在 V 的左边，它们的顶点按横坐标递增的顺序分别为 $u_0,u_1,\cdots,u_{n-1}$ 和 $v_0,v_1,\cdots,v_{n-1}$，依次存储在规模为 n^2 的 LARPBS 的前 $2n$ 个处理器上，处理器 $PE_i(0\leqslant i\leqslant n-1)$ 存储 u_i，处理器 $PE_j(n\leqslant i\leqslant 2n-1)$ 存储 v_j。图4给出了算法2的示意图。下文中出现的 $s_j(0\leqslant j\leqslant n-1)$ 是 $[0,n-1]$ 区间内的整数，它是切线的右切点的下标，该切线的左切点的下标为 j。

算法2 计算两分离的上凸壳间的切线

输入：存储在规模为 n^2 的 LARPBS 的前 $2n$ 个处理器上的两上凸壳的顶点

输出：处理器 PE_0 两上凸壳间的切线及切点

1. 将整个总线分成 n 个独立的子段，每段 n 个连续的处理器，依次存储 V 上的 n 个顶点 $v_0,v_1,\cdots,v_{n-1}$，第 i 段的 n 个处理器上均存储顶点 u_i：

1.1 处理器 $PE_i(0\leqslant i\leqslant n-1)$ 向处理器 $PE_{i\times n+k}(0\leqslant k\leqslant n-1)$ 播送 u_i。

1.2 处理器 $PE_i(n\leqslant i\leqslant 2n-1)$ 向处理器 $PE_{j\times n+i}(0\leqslant j\leqslant n-1)$ 播送 v_i。

1.3 处理器 $PE_i(0\leqslant i\leqslant n^2-1)$ 计算 $j=\lfloor i/n\rfloor$ 和 $k=i \bmod n$。处理器 $PE_{j\times n}(0\leqslant j\leqslant n-1)$ 断开左边的重构开关。

2. 第 j 段 $(0\leqslant j\leqslant n-1)$ 利用算法1计算 u_j 到 V 的切线，处理器 $PE_{j\times n}$ 存储切点 v_{i_j}，合上左边的重构开关，将切点送回处理器 PE_j。

3. 按照与1相同的方法将整个总线分成 n 段，每段 n 个处理器，依次存储 U 上的 n 个顶点，第 j 段的每个处理器均存储切线 $u_jv_{i_j}$。

4. 第 j 段 $(0\leqslant j\leqslant n-1)$ 判断切线 $u_jv_{i_j}$：处理器 $PE_{j\times n+k}(0\leqslant k\leqslant n-1$ 且 $k\neq j)$ 判断 v_k 是否在 $u_jv_{i_j}$ 的下方，是则标记1，否则标记0。段内求逻辑位与，将结果存入处理器 $PE_{j\times n}$。若第 α 段的结果为1，记 $\beta=s_\alpha$，则所求的切线为 $u_\alpha v_\beta(0\leqslant\alpha,\beta\leqslant n-1)$。处理器 $PE_{j\times n}(0\leqslant j\leqslant n-1)$ 合上左边的重构开关，处理器 $PE_{i\times n}$ 将切点 u_α、v_β 向处理器 PE_0 播送。

第1步和第3步中用到了多重多点传送和分段操作，第4步的操作是点对点通信，由引理1，上述各操作在 $O(1)$ 内完成。由定理1，第2步在常数时间内完成。故有：

定理2 给定两个相互分离、顶点数均为 n 的上凸壳，若顶点按横坐标递增的顺序存储在 n^2 个处理器的 LARPBS 上，则在 $O(1)$ 时间内计算切线。

4 凸壳算法

4.1 合并算法

本小节描述上凸壳的合并算法，可用相同的方法进行下凸壳的合并。该算法将用于后文提出的凸壳算法中。为了叙述方便，我们讨论在 n^2 个处理器的 LARPBS 上合并两个大小为 n^2 的上凸壳，用 n 个处理器合并两个大小为 n 的上凸壳是类似的。

由定理2知，用 n^2 个处理器可在 $O(1)$ 内得到两个顶点数为 n 的上凸壳间的切点，为了在相同时间内计算两个顶点数为 n^2 的上凸壳间的切点，我们采用文[9]中的采样方法：给定两个相互分离的、顶点数均为 n^2 的上凸壳 P 和 Q，其顶点按横坐标递增的顺序分别为 $p_0,p_1,\cdots,p_{n^2-1}$ 和 $q_0,q_1,\cdots,q_{n^2-1}$，对上凸壳的顶点集每隔 n 采样，P 的样本点及样本点间的连线构成的仍是一个上凸壳，称为样本上凸壳 A，记 A 的顶点集为 $\{a_i=p_{i\times n}\mid 0\leqslant i\leqslant n-1\}$，样本点分 P 为 n 块 $A_0,A_1,\cdots,A_{n-1}$，每块仍是一个上凸壳，A_i 的顶点依次为 $p_{i\times n},p_{i\times n+1},\cdots,p_{(i+1)\times n-1},p_{(i+1)\times n}$。类似地，记 Q 的样本上凸壳 B 的顶点集为 $\{b_i=q_{i\times n}\mid 0\leqslant i\leqslant n-1\}$，相应的块为 $B_0,B_1,\cdots,B_{n-1}$。算法中出现的 m、t、α、β、γ、c 均为 $[0,n-1]$ 区间中的整数。文[9]中提出如下性质：

引理2 若已知 A 和 B 间的切线是 a_mb_t，设 P 和 Q 间的切线是 $p_\alpha q_\beta$，那么以下至少有一种情况满足：(1) $p_\alpha\in A_m$；(2) $p_\alpha\in A_{m+1}$；(3) $q_\beta\in B_t$；(4) $q_\beta\in B_{t+1}$。

引理3 过 a_m、a_{m+1} 分别向 Q 引切线 $a_mq_{i_m}$、$a_{m+1}q_{i_{m+1}}$，若 a_{m+1} 在 P 中的左邻在切线 $a_{m+1}q_{i_{m+1}}$ 之上且 a_m 在 P 中的右邻在切线 $a_mq_{i_m}$ 之上，那么情况(2)成立，即 $p_\alpha\in A_{m+1}$。

基于引理2，我们可以先计算样本上凸壳间的切线，然后确定切点所在的块，只需在该块中寻找切点，这样减少了计算量。用类似的方法确定另一个切点。

基于引理3，我们得到算法3，用于检测情况(2)以及在该情况下确定切线 $p_\alpha q_\beta$，其余三种情况处理方法类似。设 n^2 个处理器依次为 $PE_0,PE_1,\cdots,PE_{n^2-1}$，处理器 $PE_i(0\leqslant i\leqslant n-1)$ 上存储顶点 p_i，处理器 $PE_j(n\leqslant i\leqslant 2n-1)$ 上存储顶点 q_j，图5给出了该算法的示意图。

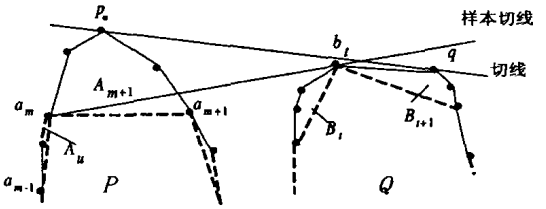

图5 由样本凸壳间的切线确定凸壳的切线

算法3 合并两个分离的上凸壳

输入：两上凸壳的 $2n^2$ 个顶点按横坐标递增的顺序依次存储在 $2n^2$ 个处理器的 LARPBS 上。

输出：合并后的上凸壳的顶点按横坐标递增的顺序依次存储在 LARPBS 左端连续的一些处理器上。

1. 对 P、Q 的顶点每隔 n 个点采样，设所得的样本点按横坐标递增的顺序分别为 $a_0,a_1,\cdots,a_{n-1}$ 和 $b_0,b_1,\cdots,b_{n-1}$。

2. 利用算法2计算样本切线，设为 a_mb_t。

3. 利用算法1求出 a_m 到 Q 的切线 $a_mq_{i_m}$ 和 a_{m+1} 到 Q 的切线 $a_{m+1}q_{i_{m+1}}$。

4. 判断 a_{m+1} 在 P 中的左邻是否在切线 $a_{m+1}q_{i_{m+1}}$ 之上且 a_m 在 P 中的右邻是否在切线 $a_mq_{i_m}$ 之上。若是，那么情况(2)成立，即 $p_\alpha\in A_{m+1}$。否则转向类似的算法检测其余情况是否成立。

5. 利用算法2计算 A_{m+1} 与 B 之间的切线 $p_\gamma b_{i_\gamma}$。

6. 如果 $p_\gamma b_{i_\gamma}$ 是 A_{m+1} 与 Q 的切线，则该切线即为所求，$\alpha=\gamma$，$\beta=n\times s_\gamma$，转向9。

7. 否则，q_β 在 B_{i_γ} 或 $B_{i_\gamma+1}$ 中，利用类似第1步和第2步的检测方法确定 q_β 所在的块 B_c。

8. 计算 A_{m+1} 与 B_c 的切线 $p_\alpha q_\beta$。

9. 合并后的上凸壳顶点依次为 $p_0,p_1,\cdots,p_\alpha,q_\beta,q_{\beta+1},\cdots,q_{n^2-1}$。

算法3在 LARPBS 上的具体实现如下：

1. LARPBS 的左端依次存储样本上凸壳 A、B 的顶点：处理器 $PE_i(0\leqslant i\leqslant n^2-1)$ 计算 $j=\lfloor i/n\rfloor$ 和 $k=i \bmod n$。处理器 $PE_{j\times n}(0\leqslant j\leqslant n-1)$ 上存储样本点 a_j，向处理器 PE_j 播送。处理器 $PE_i(n^2\leqslant i\leqslant 2n^2-1)$ 计算 $j=\lfloor(i-n^2)/n\rfloor$ 和 $k=(i-n^2) \bmod n$。处理器 $PE_{j\times n+n^2}(0\leqslant j\leqslant n-1)$ 上存储样本点 b_j，向处理器 PE_{j+n^2} 播送。

2. 利用算法2计算切线 a_mb_t，切线及切点存储在处理器 PE_0 中。

3. 利用算法1求出 a_m 到 Q 的切线 $a_mb_{i_m}$ 和 a_{m+1} 到 Q 的切线 $a_{m+1}b_{i_{m+1}}$，切线及切点分别存储在处理器 $PE_{m\times n}$ 和处理器 $PE_{(m+1)\times n}$ 中。

4. 处理器 $PE_{(m+1)\times n}$ 向处理器 $PE_{(m+1)\times n-1}$ 播送切线 $a_{m+1}b_{i_{m+1}}$，处理器 $PE_{(m+1)\times n-1}$ 存储的是 a_{m+1} 的左邻，判断它是否

在切线 $a_{m+1}b_{t_{m+1}}$ 之上。处理器 $PE_{m\times n}$ 向处理器 $PE_{m\times n+1}$ 播送切线 $a_mb_{t_m}$，处理器 $PE_{m\times n+1}$ 存储的是 a_m 的右邻，判断它是否在切线 $a_mb_{t_m}$ 之上。若以上均满足，那么情况(2)成立，即 $p_\alpha\in A_{m+1}$。

5. 将 A_{m+1} 与 B 的顶点分布在前 $2n+2$ 个处理器中：处理器 $PE_i(0\leqslant i\leqslant n^2-1)$ 计算 $j=\lfloor i/n\rfloor$ 和 $k=i \bmod n$，处理器 $PE_{m\times n+k}(0\leqslant k\leqslant n)$ 存储的是 A_{m+1} 中的顶点，向处理器 PE_k 播送。处理器 $PE_{j\times n+n^2}(0\leqslant j\leqslant n-1)$ 存储的是 B 中的顶点，向处理器 PE_{j+n+1} 播送。利用算法2计算 A_{m+1} 与 B 之间的切线 $p_rq_{t_r}$，切线及切点存储在处理器 PE_0 中。

6. 处理器 PE_0 向处理器 $PE_i(n\leqslant i\leqslant 2n-1)$ 播送切线 $p_rq_{t_r}$，处理器 PE_i 判断 q_i 是否在切线 $p_rq_{t_r}$ 的下方，是则标记1，反之标记0。处理器 $PE_i(n\leqslant i\leqslant 2n-1)$ 参与计算逻辑位与，若结果为1，则 $\alpha=\gamma,\beta=n\times s_r$，清除所有标记，转向9。

7. 否则，用类似第1步和第2步的检测方法确定 q_β 所在的块 B_ϵ。

8. 类似第5步计算 A_{m+1} 与 B_ϵ 的切线 $p_\alpha q_\beta$，区别是 B_ϵ 的顶点存储在处理器 $PE_{\epsilon\times n+n^2+j}(0\leqslant j\leqslant n-1)$ 中。

9. 将上凸壳的顶点依次分布在 LARPBS 的左端：处理器 PE_0 广播切点 p_α、q_β 的下标 α、β。处理器 $PE_i(n^2+\beta\leqslant i\leqslant 2n^2-1)$ 中存储上凸壳的边界上处在切点 q_β 右侧的顶点，计算 $j=i-(n^2+\beta)+\alpha+1$，向处理器 PE_j 播送。最末顶点所在的处理器 $PE_{n^2-\beta+\alpha}$ 标记0，上凸壳的顶点数为 $\alpha+n^2-\beta$。

由定理1和定理2知，第3步和第5步中计算切线在常数时间内完成。第5步和第9步中的分布顶点用到了点对点通信，第4步采用的是多播和逻辑位与操作，由引理1知，各操作的时间复杂度为 $O(1)$，故有如下结论：

定理3 给定两个相互分离、大小为 n^2 的上凸壳，当顶点按横坐标递增的顺序依次存储在 $2n^2$ 个处理器的 LARPBS 上，可在 $O(1)$ 时间内完成合并。

4.2 凸壳算法的基本原理及具体描述

我们考虑大小为 n 的平面点集 S 的凸壳算法，假设点集 S 中任意三点不共线且任意两点的横坐标不相等。按横坐标将它分成 S_1、S_2 两个子集，使得 S_1 中每个点的横坐标都小于 S_2 每个点的横坐标，分别递归地求 S_1、S_2 的凸壳，直到划分所得的子集的大小为1。然后通过求 S_1 与 S_2 的凸壳间的切线得到 S 的凸壳。

本算法将集合划分成两个大小近似相等的子集，递归求解 $\log n$ 步，直到子集中仅有一个点。算法开始时将每个点看作仅含一个点的凸壳，第 d 步 $(1\leqslant d\leqslant\log n)$ 有 $n/2^{d-1}$ 个分离的凸壳需要合并，分成两个一对，共 $n/2^d$ 对，利用算法3对各对凸壳并行地进行合并。

该算法在 n 个处理器的 LARPBS 上实现时，假设这 n 个点按横坐标递增的顺序依次存储。为便于分段操作，分别计算 S 的上凸壳和下凸壳，各需 $\log n$ 步，计算方法是相同的。

在第 d 步 $(1\leqslant d\leqslant\log n)$ 的开始，总线被分成 $n/2^{d-1}$ 个子段，每段的左端存储一个上凸壳，为了应用定理3，必须先将待合并的两个上凸壳连续存储在子处理器段的左端。进行如下操作：偶数段的首处理器合上左边的重构开关，总线被分成 $n/2^d$ 段，每段的前半段和后半段的左端分别存储一个上凸壳，移动后一个上凸壳的顶点，使两个上凸壳在该段的左端连续存储。

设 S 的上凸壳和下凸壳的顶点数分别为 N_u 和 N_d，当第 $2\log n$ 步结束时，总线的前 N_u 个处理器和 N_d 个处理器分别依次存储上凸壳和下凸壳的顶点。为了便于输出，当 $N_u>N_d$ 时，移动下凸壳的顶点，使凸壳的顶点按顺时针依次存储在总线的左端，反之移动上凸壳的顶点，凸壳的顶点按逆时针依次存储。

算法的具体描述如下：

算法4 平面点集的凸壳算法

输入：按横坐标递增的顺序依次存储在 n 个处理器的 LARPBS 上的 n 个平面点

输出：按顺时针序存储的凸壳顶点

初始时，每个顶点都是上凸壳的最末一个顶点，所有处理器均标记0。

第 d 步，$1\leqslant d\leqslant\log n$：

$d.1$ 有 $n/2^{d-1}$ 个上凸壳，将整个总线分成 $n/2^d$ 段，每段左端连续存储一对相邻的上凸壳：处理器 $PE_i(0\leqslant i\leqslant n-1)$ 计算 $j=\lfloor i/2^d\rfloor$ 和 $k=i \bmod 2^d$。对于 $0\leqslant j\leqslant(n-1)/2^d$：处理器 $PE_{j\times 2^d}$ 断开左边的重构开关。设第 j 段的两个上凸壳为 U、V，对应的顶点数分别为 N_U^j、N_V^j，则第 j 段前半段的左端 N_U^j 个处理器中依次存储 U 的顶点，后半段的左端 N_V^j 个处理器中依次存储 V 的顶点。被标记的处理器在段内广播 N_U^j 和 N_V^j。处理器 $PE_{j\times 2^d+k}(2^{d-1}\leqslant k\leqslant 2^{d-1}+N_V^j-1)$ 向处理器 $PE_{j\times 2^d+k-2^{d-1}-1+N_U^j}$ 播送数据，U 的最末顶点所在的处理器 $PE_{j\times 2^d+N_U^j-1}$ 取消标记。

$d.2$ 利用算法3，各段并行地合并每对上凸壳。

用同样的方法计算下凸壳，下凸壳的最末一个顶点所在的处理器标记1。

第 $2\log n+1$ 步：将凸壳顶点在 LARPBS 的左端按顺时针方向依次存储：被标记0的处理器的序号为上凸壳的顶点数 N_u，被标记1的处理器的序号为下凸壳的顶点数 N_d，被标记的处理器广播 N_u 和 N_d，并比较大小。若 $N_u>N_d$，处理器 PE_0 和处理器 PE_{N_d-1} 中存储的分别是最左点和最右点，不需移动。对 $1\leqslant k\leqslant N_d-2$：处理器 PE_{N_d-1-k} 向处理器 PE_{N_u-1+k} 播送顶点。反之，移动上凸壳的顶点，用同样的方法处理。

由引理1，算法中的顶点移动操作在常数时间内完成，应用定理3，每步的合并操作也在常数时间内完成，故有结论：

定理4 给定大小为 n 的平面点集 S，当各点按横坐标递增的顺序存储在 n 个处理器的 LARPBS 上，可在 $O(\log n)$ 内确定 S 的凸壳。

总结 求平面点集的凸壳是计算几何中的一个基本问题，本文提出对数时间的凸壳算法，充分体现了 LARPBS 上的通信灵活性，提高并行算法的执行效率。但算法处理的点在输入时已按横坐标递增的顺序存储，对于任意存储的平面点集，算法的时间复杂度将受到排序时间的影响。据目前所知，对于大小为 n 的输入，在规模为 n 的 LARPBS 上最快的排序时间为 $O(\log n\log\log n)$[10]，因此 n 个平面点的凸壳在 $O(\log n\log\log n)$ 内确定。若想得到更好的求解平面点集的凸壳算法，必须加快排序或者对本算法做进一步的改进。

参 考 文 献

1 Preparata F P, Shamos M I. Computational Geometry: an Introduction. Springer-Verlag, New York, NY, 1985

2 Bokka V, Gurla H, Olariu S, Schwing J. Time- and VLSI-optimal convex hull computation on meshes with multiple broadcasting. Frontiers of Massively Parallel Computation, Arlington, Virginia, 1995, 2:506～513

3 Hayashi T, Nakano K, Olariu S. An $O((\log\log n)^2)$ Time Algorithm to Compute the Convex Hull of Sorted Points on Reconfigurable Meshes. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(12)

4 Li K, Pan Y, Hamdi M. Solving Graph theory problems using reconfigurable pipelined optical buses. Parallel Computing, 2000, 26:723～735

5 Li K, Zheng S Q, Pan Y. Fast and processor efficient parallel matrix multiplication algorithms on a linear array with a reconfigurable pipelined bus system. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(8):705～720

6 Pan Y, Li K. Linear array with a reconfigurable pipelined bus system: concepts and applications. Journal of Information Sciences 1998, 106:237～258

REESSE 2公开密钥密码体制

苏盛辉[1] 杨炳儒[2]

(北京石油化工学院信息管理系 北京102617)[1] (北京科技大学信息工程学院 北京100083)[2]

摘 要 作者综合利用超递增序列、杠杆函数和HASH函数的特性提出了能有效地抵御"极小点"攻击和"LOB-L3归约基"攻击的REESSE2公开密钥密码体制,详细描述了该体制的数学基础、密钥生成算法、加密算法和解密算法。文章对REESSE2体制的安全性和优越性做了分析,并归纳了提高公钥密码体制安全性的两条途径。

关键词 杠杆函数,超递增数列,冗余加密,公开密钥密码体制

The REESSE 2 Public Key Cryptosystem

SU Sheng-Hui[1] YANG Bing-Ru[2]

(Beijing Institute of Petrochemical Technology, Beijing 102617)[1] (Beijing University of Science and Technology, Beijing 100083)[2]

Abstract On the basis of the super increasing sequence, lever function and HASH function, the authors put forward the REESSE2 public key cryptosystem which can defend efficiently attacks by the extreme small dot and LOB-L[3] reduced base, expound the math foundation and three algorithms for generating-key, encryption and decryption of the crytosystem, analyze algorithmic security and advantage, and induce the two approaches to enhancing securities of public key crytosystems.

Keywords Lever function, Super increasing sequence, Redundancy encryption, Public key crytosystem

1 引言

1978年,在Whitfield Diffie和Martin Hellman提出公开密钥密码思想两年后,首先由Ralph Merkle和Martin Hellman设计了一个基于背包问题的公钥密码体制。尽管背包问题被证明属于NP完全类,但是,由于MH变换(模乘变换)本身的缺陷,该体制先后被以"极小点"方法和"LOB-L[3]归约基"方法破译[1~3]。

MH背包体制可以被扩展到伽罗瓦域的多项式环上。此时,虽然超递增序列的元素由整数变成了多项式,但是从私钥到公钥的变换仍是模乘变换。由于多项式模乘变换与整数模乘变换在本质上是一致的,因此这种改变只是原有体制的扩展,而不是新体制的发明[1,4])。显然,多项式环上的MH公钥体制与椭圆曲线上的ElGamal公钥体制有异曲同工之妙:它们都提高了体制的安全性。这是因为,在多项式环上或椭圆曲线上,对基本运算符进行了组合,形成了一种新的更复杂的群的运算符。

本文介绍的REESSE 2公钥密码体制综合利用了超递增序列、杠杆函数和HASH单向散列函数的特点,对从私钥到公钥的变换进行了创新,即密钥的变换从单群扩展到了双群,同时在域的两个群上实施变换运算。注意,这与代数系统升级了但密钥变换仍在单群上进行有本质的不同。经分析,REESSE 2体制在安全性和速度方面有其优势。

2 REESSE2体制的数学基础

2.1 三个有关函数

2.1.1 杠杆函数 在REESSE1公钥体制中[5],私有密钥为互素序列$\{A_1,A_2,\cdots,A_n\}$,公开密钥$\{C_1,C_2,\cdots,C_n\}$经公式$C_i=(A_i*W^{f(i)})\bmod M$变换后得到。其中,$f(i)$为整数到整数的单射函数,且$5\leqslant f(i)\leqslant(n+4),(i=1,2,\cdots,n)$。

在上述变换中,$f(i)$具有这样的特点:从公开密钥破译私有密钥时,需考虑$f(i)$的全排列数$n!$,因此当n足够大时,$f(i)$的排列在有效时间内不可被穷举;但从私有密钥解开密文时只需考虑$f(i)$的累加和,时间复杂度与n多项式相关,故解密是可行的。若把密文当作支点,则$f(i)$是"公开"一头计算量大,"私有"一头计算量小,因此,称具有这种特征的$f(i)$为杠杆函数。

2.1.2 超递增序列 设$A_1,A_2,\cdots,A_n$为n个互不相同的正整数,且满足:$A_i>\Sigma A_j(i=2,3,\cdots,n;j=1,2,\cdots,i-1)$,则称这样的正整数序列为超递增序列,记为$\{A_1,A_2,\cdots,A_n\}$或$\{A_i\}$。

超递增序列有如下性质:对于任意的正整数$m(1\leqslant m\leqslant$

苏盛辉 副教授,博士生,主要从事信息安全、柔性建模与集成技术的研究和开发。**杨炳儒** 教授,博士生导师,主要从事知识发现与智能系统、柔性建模与集成技术的研究。

7 Pan Y, Li K, Zheng S Q. Fast Nearest Neighbor Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System: [Technical Report # 97-002]. Department of computer Science, Louisiana State Unibersity, Baton Rouge, LA 70803(1997)

8 Preparata F P, Hong S J. Convex hulls of finite sets of points in two and three dimensions. Comm. ACM, 1977, 2(20): 87~93

9 Atallah M J, Goodrich M T. Parallel algorithms for some functions of two convex polygons. algorithmica, 1988, 3: 535~548

10 Datta A, Owens R, Soundaralakshmi S. Fast Sorting Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System. IEEE Transactions on Parallel and Distributed Systems 2002, 13(3): 202~222