# 多核架构下计算凸壳的并行算法

颜 坚[1] 毕硕本[2] 汪 大[1] 郭 忆[2]
(南京信息工程大学计算机与软件学院 南京 210044)[1] (南京信息工程大学遥感学院 南京 210044)[2]

**摘 要** 改进了周培德的 $Z_{3\text{-}2}$ 算法,提出一种在多核架构下计算平面点集凸壳的并行算法。用"颜氏距离"来数字化平面上点与有向线段的位置关系,减少了计算次数和时间。进一步将原算法中比较耗时的两个过程分别在O(1)的时间复杂度内进行迭代分解,即当原问题规模大于给定阈值时,将原问题分解为若干个独立的子问题,若所得子问题的规模仍大于给定阈值,则再对子问题进行分解;所有子问题被加入并行任务组进行并行求解以充分利用多核处理器的并行计算资源。给出了算法的正确性说明,实验结果也表明本算法稳定高效。

**关键词** 凸壳,并行计算,多核,颜氏距离
**中图法分类号** TP301.6 **文献标识码** A

## Parallel Algorithm for Computing Convex Hulls in Multi-processor Architecture

YAN Jian[1] BI Shuo-ben[2] WANG Da[1] GUO Yi[2]
(School of Computer & Software, Nanjing University of Information Science & Technology, Nanjing 210044, China)[1]
(School of Remote Sensing, Nanjing University of Information Science & Technology, Nanjing 210044, China)[2]

**Abstract** This paper Improved the $Z_{3\text{-}2}$ algorithm proposed by ZHOU Pei-de, and proposed a parallel algorithm for computing the convex hulls of planar point set in multi-processor architecture. The times and duration of calculation were reduced by digitizing the positional relationship between a point and a directed line segment on plane with "Yan's distance". Further, the two progresses were decomposed iteratively which are the foremost time-consuming parts in the algorithm within the complexity of O(1). That is, the original task is decomposed into several sub-tasks when its scale is greater than a given threshold and then decompose any of the sub-tasks if its scale is still greater than the threshold. All sub-tasks will be executed in parallel from the parallel task group to take full advantage of the parallel computation resources of the multi-processor. The correctness of the algorithm was discussed. The experiment results show that the algorithm is efficient and stable.

**Keywords** Convex hull, Parallel computing, Multi-core, Yan's distance

## 1 引言

凸壳是计算几何中最普遍、最基本的一种结构[1]。平面点集的凸壳问题在计算机图形学、图像处理与模式识别、地理信息系统等众多领域中应用广泛。

计算凸壳的串行算法国内外的早期研究成果主要有卷包裹法、格雷厄姆算法、分治算法、增量算法、实时算法、快速算法等。其中,格雷厄姆算法[2]是 Graham 于 1972 提出的求解平面点集凸壳问题的早期最佳算法,其时间复杂度为 O($n$log$n$)。金文华等对初始点集合进行初步筛选后,将点集按照一定顺序串联成有序简单多边形,再利用前瞻回溯方法搜索多边形而得到凸壳[3]。赵军和曲仕茹利用巧妙的设计在其算法中避免了乘法运算[4],节省了计算时间。

随着并行计算技术的不断发展,国内外的许多研究人员尝试将并行技术应用到凸壳的计算[5-8]中。这些并行算法中,绝大多数使用了分治策略,将原问题分解为若干子问题,并行独立求解这些子问题,然后合并所有子问题的解得到原问题的解。如 Reif 和 Sen 利用随机抽样执行划分[5];张三元等先对点集按 $x$ 坐标排序,再将点集按顺序分成与处理器数目相同数目的子集合,各子集合大小基本相同[6];郝小柱等通过排序将点集串联形成逆时针方向封闭的有序简单多边形,再对该多边形进行分割[7];周启海等根据点集的 $x$ 坐标进行等间距带状划分[8]。目前,这些并行算法还存在两个主要问题:第一,并行粒度不受控制,负载平衡问题考虑不足,有可能出现瓶颈;第二,为实现并行计算引入了大量额外计算任务,如对原问题进行分解时,通常要对初始点集合进行排序等。

2005 年,随着 Intel 第一款双核处理器 Pentium D/XE 的出现,处理器进入多核时代[9]。并行计算正成为主要的程序运行模式[10]。周培德在文献[1]中提出用分治法求解平面点集凸壳的 $Z_{3\text{-}2}$ 算法,其基本思想是先求出点集中 $x$、$y$ 坐标最

到稿日期:2012-04-05 返修日期:2012-07-16 本文受国家自然科学基金项目(41071253),江苏省六大人才高峰项目(20080249)资助。
颜 坚(1988-),男,硕士生,主要研究方向为计算几何、时空信息融合、并行计算等,E-mail:407037359@qq.com;毕硕本(1965-),男,博士,教授,主要研究方向为地理信息系统、数据挖掘等;汪 大(1988-),男,硕士生,主要研究方向为地理信息系统、遥感图像融合等;郭 忆(1989-),男,硕士生,主要研究方向为地理信息系统、气象信息融合等。

大值、最小值，然后顺序连接最大值、最小值所对应的点成四边形，该四边形划分点集为5个子集，不考虑位于四边形内的子集，对其他4个子集迭代地删除不是凸壳顶点的点。可见，$Z_{3\text{-}2}$算法符合动态快速优化算法[11]的应用条件。本文首先提出了用颜氏距离来判断点与有向线段的位置关系，而在原算法中仅利用了颜氏距离的符号来判断点位于有向线段的哪一侧(即正负划分)，再使用欧氏距离找出离有向线段最远的点。本文将这两个过程合二为一，摒弃了计算欧氏距离时的除法和开方运算。其次，为充分利用多核处理器的并行计算资源，对原算法中点集分类过程和迭代过程进行了并行扩展，将耗时较多的任务分解为若干可以并行执行的任务，各并行任务之间相互独立，任务分解的过程可以在O(1)的时间复杂度内完成，并行开销小。最后，通过并行、串行的自适应选择来控制并行任务粒度以控制并行加速比，并消除了可能产生的瓶颈。

## 2 算法的理论基础

本文约定简单多边形的顶点序列为该多边形的各顶点按逆时针方向排列而成的点列，其有向边链为顺次连接顶点序列中的点所成的逆时针方向的闭合的有向环。

**定义1** 设$S:=\{P_1,\cdots,P_n\}$，$n\geqslant 3$，为平面上任意非空点集，则

(1)点集$S$的凸壳$CH(S)$为包含$S$的最小凸集；

(2)点集$S$的凸壳边界$BCH(S)$是一凸多边形，其顶点为$S$中的点；

(3)计算平面点集$S$的凸壳即从$S$中找出若干点，它们沿逆时针方向对应于$BCH(S)$的各个顶点，称该点列为$S$的凸壳顶点序列，记为$LVBCH(S)$；

(4)集合$Q=\{P|P\in S$且$P$在$BCH(S)$上$\}$中的点为$S$的凸壳点，由若干凸壳点围成的凸多边形为$S$的一个不完全凸壳边界$NBCH(S)$，称$NBCH(S)$的顶点序列为$S$的不完全凸壳点列$NCP(S)$，$NBCH(S)$的有向边链为对应的不完全凸壳边链$NCE(S)$。

**定义2(颜氏距离)** 设$P_1P_2$为平面上一向量，其起止点坐标分别为$P_1(x_1,y_1)$和$P_2(x_2,y_2)$，点$P(x,y)$为平面上任意一点，定义行列式

$$D=\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} \tag{1}$$

的值为点$P$到向量$P_1P_2$的颜氏距离，记为$YD(P,P_1P_2)$。

颜氏距离用来表征点和有向线段的位置及距离关系，本文主要利用它的如下性质：

**性质1(几何意义)** 三角形$\triangle PP_1P_2$的面积$S(\triangle PP_1P_2)=|YD(P,P_1P_2)|/2$；

**性质2(正负划分性)** 若点$P$在直线$P_1P_2$上，则$YD(P,P_1P_2)=0$；否则，若点$P$在向量$P_1P_2$的右(左)侧，则$YD(P,P_1P_2)>0$($YD(P,P_1P_2)<0$)；

**性质3(线性性)** 若$P_3=(1-\lambda)P_1+\lambda P_2$，则$YD(P,P_1P_2)=\lambda YD(P,P_1P_3)$；

**性质4(轮换性)** $YD(P_3,P_1P_2)=-YD(P_2,P_1P_3)$；

**性质5(自反性)** $YD(P,P_1P_2)=-YD(P,P_2P_1)$。

上述性质均可通过代数运算得出证明，本文不再赘述其证明过程。下面提出3个基于颜氏距离的定理，它们是本文算法的理论支撑。

**定理1** 设$S:=\{P_1,\cdots,P_n\}$，$n\geqslant 3$，为平面上任意非空点集，$Q_1$、$Q_2$为平面上任意两点，若存在$P\in S$使得$YD(P,Q_1Q_2)=\max\{YD(P_i,Q_1Q_2)\mid i=1:n\}>0$，则$P$在$BCH(S)$上。

证明(反证法)：假设$P$不在$BCH(S)$上，则：

(1)由定义1可知，点$P$不可能在$BCH(S)$外；

(2)假设$P$在$BCH(S)$内，则在$BCH(S)$上必能找到一点$Q$，使得点$P$在$\triangle QQ_1Q_2$内，则有$S(\triangle PQ_1Q_2)<S(\triangle QQ_1Q_2)$；又因为$YD(P,Q_1Q_2)>0$，根据性质1、性质2有$YD(Q,Q_1Q_2)>YD(P,Q_1Q_2)$，这与$YD(P,Q_1Q_2)=\max\{YD(P_i,Q_1Q_2)\mid i=1:n\}$相矛盾，所以$P$不在$BCH(S)$内。

综合(1)、(2)可知，$P$在$BCH(S)$上。

定理1说明了点集中满足其条件的点一定在该点集的凸壳边界上，算法求解过程中找到的所有凸壳点都应用了定理1的结论。

**定理2** 设$S:=\{P_1,\cdots,P_n\}$，$n\geqslant 3$，为平面上任意非空点集，则任意$P\in S$，在$NCE(S)$中至多存在一条有向边$t$，使得$YD(P,t)>0$。

证明(反证法)：假设在$NCE(S)$中存在两条以上的有向边满足条件，设$t_1$、$t_2$为其中任意两条，则存在$P\in S$使得$YD(P,t_1)>0$且$YD(P,t_2)>0$。如图1所示，连接$t_1$终点$t_2$起点、$t_2$终点$t_1$起点得向量$QR$、$UW$。因为$NCE(S)$围成的多边形$NBCH(S)$是非凹的，所以向量$QR$和$UW$不相交，不妨设$QR$为其中长度较长的一条，则$t_1$和$t_2$的延长线的交点$O$在$QR$的左侧，即$YD(O,QR)<0$。根据性质3可知，$YD(P,OQ)>0$；又由性质4、性质5得$YD(O,PQ)<0$。同理$YD(O,RP)<0$。因此，点$O$在$\triangle PQR$内，所以点$W$、$U$在$\triangle PQR$内。从而找不到过点$W$或$U$的直线$l$使得$P$、$Q$、$R$3点在$l$的同侧闭半平面内，所以点$W$和点$U$在$BCH(S)$内部。这与$t_1$、$t_2$为$NCE(S)$中的有向边相矛盾。证毕。

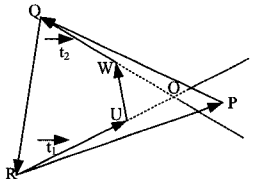


图1 定理2示例图

定理2说明对于当前的不完全凸壳边链，点集中的每一个点至多只能在其中一条有向边的外侧，这是对点集进行分类的依据。

**定理3** 设$S:=\{P_1,\cdots,P_n\}$，$n\geqslant 3$，为平面上任意非空点集，任意$R_1R_2\in NCE(S)$，定义点集合$Q:=\{P|P\in S$且$YD(P,R_1R_2)>0\}$，当$Q\neq\varnothing$时，存在$P_0\in Q$，使得$YD(P_0,R_1R_2)=\max\{YD(P,R_1R_2)\mid P\in Q\}$；那么对于任意$P\in Q$，若$YD(P,R_1P_0)\leqslant 0$且$YD(P,P_0R_2)\leqslant 0$，则$P$为非凸壳顶点。

证明：因为$P\in Q$，所以$YD(P,R_1R_2)\leqslant 0$；又$YD(P,R_1P_0)\leqslant 0$且$YD(P,Q_0P_2)\leqslant 0$，所以$P$在$\triangle R_1R_2P_0$内部或边界上。

当$P$在$\triangle R_1R_2P_0$内部时，$P$在$BCH(S)$内，故$P$为非凸壳顶点；

当 $P$ 在$\triangle R_1R_2P_0$ 边界上时，假设 $P\in LVBCH(S)$，则在 $LVBCH(S)$中有 3 点共线，这与 $BCH(S)$是凸多边形相矛盾。证毕。

定理 3 则说明了算法迭代过程中可以删除满足其条件的所有点，因为它们不是凸壳顶点。

定理 1—定理 3 是本文算法的主要依据。

## 3 算法描述

### 3.1 核心思想

本文算法的核心思想可以用“掏空”法来形象地描述。一般地，算法从点集的最内部开始，逐步并行地“挖除”非凸壳顶点，直到剩下的点全为凸壳顶点为止。具体地说，本文算法不断地对点集进行分类，每次找到数个凸壳点、一部分非凸壳点和一部分不确定的点——可能是凸壳点的点，删除所有找到的非凸壳点，对不确定的点迭代地分类和删除，直至不再有不确定的点。

### 3.2 算法流程

求任意平面点集 $S$ 的凸壳算法主要分为 4 个步骤，分别为初始不完全凸壳点序列生成、点集的分类、对应不完全凸壳边链中各条有向边的迭代生长和删除非凸壳顶点，如图 2 所示。

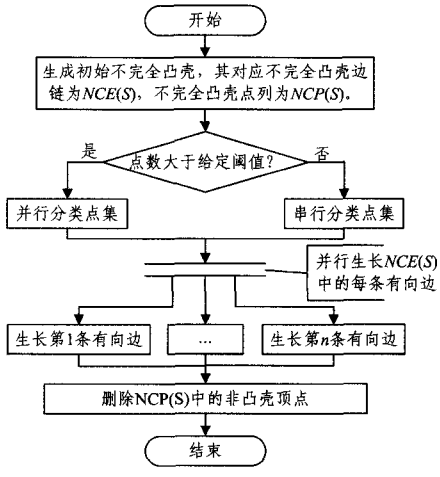


图 2 算法流程图

#### 3.2.1 初始不完全凸壳点序列的生成

理论上只要找到任意两个不同的凸壳点，其都可以作为初始不完全凸壳点列。为能在第一步就删除大量的非凸壳点，分别找到沿 $x$ 坐标、$y$ 坐标方向的 4 个极值点。在这 4 个点中可能存在相同的点，例如 $x$ 坐标的最大值点和 $y$ 坐标的最大值点可能为同一个点，该点既是 $S$ 中的最上方点，同时也是最右方点。删除这 4 个点中相同的点，对相同的点只保留一个备份，当 $S$ 中所有点不全共线时，至少能找到 3 个点，将找到的点按逆时针序排列作为初始不完全凸壳点列。

#### 3.2.2 点集的分类

对于前一步找到的初始不完全凸壳点列 $NCP(S)$，其对应的不完全凸壳边链为 $NCE(S)$，根据定理 2 的结论可知，点集 $S$ 中的点要么在 $NCE(S)$中所有有向边左侧，要么在 $NCE(S)$中某一条有向边的右侧。由凸壳的定义知，在 $NCE(S)$中所有有向边的左侧的点一定不是凸壳点，将这些点从点集中删除。其余点按照其在哪个有向边的右侧进行分类。这样，$NCE(S)$中每条有向边就都包含其右侧的所有点的数据。为叙述方便，将某条有向边右侧的点称为该边的外点。

由定理 2 和定理 3 可知，在点集的分类过程中，点的去向是唯一的。考虑 4 条有向边的情况，设 $S$ 为待分类的点集，$e_1$、$e_2$、$e_3$、$e_4$ 分别为待插入外点的有向边，则 $S$ 中的点要么作为 $e_1$ 的外点，要么作为 $e_2$、$e_3$ 或 $e_4$ 的外点，要么为非凸壳点。对点集 $S$ 按 $e_1$、$e_2$、$e_3$、$e_4$ 进行分类的过程如下：

(1)如果 $S$ 中的点数不大于 MAXCLS 的 4 倍，转(5)；

其中，MAXCLS 是对点集分类时进行并行、串行选择的阈值，它表示单个分类任务处理点集的点数的最小值。

(2)将 $S$ 划分为两个大小近似相等的点集合 $S_1$ 和 $S_2$。

(3)对点集 $S_1$ 按 $e_1$、$e_2$ 进行并行分类：将 $S_1$ 划分为两个大小近似相等的点集合 $S_{11}$ 和 $S_{12}$，从 $S_{11}$ 中删除 $e_1$ 的外点，将这些点加入 $e_1$ 的外点集合，并行地对 $S_{12}$ 和 $e_2$ 执行相同过程，将 $S_{11}$ 和 $S_{12}$ 中剩余的点合并到原来的 $S_1$ 中；并行地对点集 $S_2$ 按 $e_3$、$e_4$ 执行相同的过程。

步骤(3)中的两个并行任务执行完成后，$S_1$ 中不再包含 $e_1$、$e_2$ 的外点，同样 $S_2$ 中不再包含 $e_3$、$e_4$ 的外点。

(4)交叉地执行步骤(3)中的并行过程，即对点集 $S_1$ 按 $e_3$、$e_4$ 进行并行分类，并行地对点集 $S_2$ 按 $e_1$、$e_2$ 进行并行分类，过程与步骤(3)中的相同；转(6)。

(5)遍历 $S$ 中的每个点 $P$，判断 $P$ 是否为 $e_1$、$e_2$、$e_3$ 或 $e_4$ 的外点，若是，则从 $S$ 中删除 $P$，并将 $P$ 插入到对应的外点集合中。

(6)分类过程结束。

对于 3 条有向边的情况，过程与上述类似，不再赘述。迭代过程中，将多次对特定的点集按两条新生成的有向边进行分类，对点集 $S$ 按 $e_1$、$e_2$ 进行分类的过程如下：

(1)如果 $S$ 中的点数不大于 MAXCLS 的 2 倍，转(5)。

(2)将 $S$ 划分为两个大小近似相等的点集合 $S_1$ 和 $S_2$。

(3)从 $S_1$ 中删除 $e_1$ 的外点，将这些点加入 $e_1$ 的外点集合，并行地对 $S_2$ 和 $e_2$ 执行相同过程。

这两个并行任务执行完成后，$S_1$ 中不再包含 $e_1$ 的外点，同样 $S_2$ 中不再包含 $e_2$ 的外点。

(4)交叉地执行步骤(3)中的并行过程，即从 $S_1$ 中删除 $e_2$ 的外点，将这些点加入到 $e_2$ 的外点集合，并行地对 $S_2$ 和 $e_1$ 执行相同过程；转(6)。

(5)遍历 $S$ 中的每个点 $P$，若 $P$ 为 $e_1$ 的外点，则从 $S$ 中删除 $P$，并将 $P$ 插入到 $e_1$ 的外点集合中；若 $P$ 为 $e_2$ 的外点，则从 $S$ 中删除 $P$，并将 $P$ 插入到 $e_2$ 的外点集合中。

(6)分类过程结束。

#### 3.2.3 有向边的迭代生长

对不完全凸壳边链 $NCE(S)$中的每一条有向边 $P_0P_1$ 并行地进行迭代生长。迭代生长的过程为：

(1)若有向边 $P_0P_1$ 的外点集合为空，则返回空点列。

(2)从有向边 $P_0P_1$ 的外点集合中找出到该有向边的颜氏距离最大的点 $P$，根据定理 1 该点 $P$ 必为 $S$ 的凸壳点。

(3)连接 $P_0P$、$PP_1$ 生成两条新的有向边 $P_0P$ 和 $PP_1$，对原有向边 $P_0P_1$ 的外点集合按有向边 $P_0P$ 和 $PP_1$ 重新分类，得到 $P_0P$ 和 $PP_1$ 的外点集合。

(4)若有向边 $P_0P_1$ 的外点集合中点的数量不大于 MAXGROW 的 2 倍，则转(6)。

(5)并行地迭代生长有向边 $P_0P$ 和 $PP_1$ 分别得到点列

$L_1$ 和 $L_2$，令点列 $L=\{P_0\}\cup L_1\cup\{P\}\cup L_2\cup\{P_1\}$，转(10)。

(6)定义链表 $E$ 来存储若干有向边，将 $P_0P$ 和 $PP_1$ 插入 $E$。

(7)对于 $E$ 中的每条有向边 $e$，若其外点集合不为空，则按照(2)、(3)两步的方法得到两条新的有向边 $e_1$ 和 $e_2$；从 $E$ 中删除 $e$，并在原来 $e$ 的位置插入 $e_1$ 和 $e_2$，其中 $e_1$ 与 $e$ 的起点相同。

(8)若步骤(7)中任意一条新生成的有向边的外点集合不为空，则转(7)。

(9)删除 $E$ 中第一条有向边，并令 $L$ 为由 $E$ 中各有向边的起点排列而成的点列。

(10)将点列 $L$ 作为结果返回。

步骤(4)中的 MAXGROW 是进行并行、串行选择的阈值，只有在待生长的有向边的外点数量足够多时，才选择并行算法。随着迭代的深入，外点的数量不断减少，并行算法将不再具有优越性，此时将选择串行策略来结束迭代，并在有限次循环内得出结果。

#### 3.2.4 删除非凸壳顶点

上一步求得的点集 $S$ 的凸壳点列 $LPBCH(S)$ 中包含了若干在 $BCH(S)$ 边上而非顶点的点。依次遍历 $LPBCH(S)$ 中的每一点，设 $P$ 为当前遍历点，$R$ 为 $P$ 的前驱，$Q$ 为 $P$ 的后继。如果 $R$、$P$、$Q$ 3 点共线，就从 $LPBCH(S)$ 中删除点 $P$。经过对 $LPBCH(S)$ 的一次遍历，删除所有非顶点的点，就能得到点集 $S$ 的凸壳顶点序列 $LVBCH(S)$。

### 3.3 算法正确性

本文算法主要通过若干步迭代，删除所有非凸壳顶点，找到所有凸壳顶点。为了说明算法的正确性，既要说明没有凸壳顶点被删除，又要说明算法最终删除了所有非凸壳顶点。

算法的第二步删除了初始不完全凸壳边界内的点，这些点都是非凸壳顶点；算法的第三步迭代过程中不断地删除了满足定理 3 条件的所有点，它们是非凸壳顶点；算法的最后一步删除了在 $BCH(S)$ 边上而非顶点的点。因此，算法没有删除任何凸壳顶点。

另一方面，在算法的第三步迭代过程结束的条件是任何一条有向边的外点集合为空，又根据定理 2 知道每一次迭代生长的点只在原有向边的右侧，所以迭代结束所得的不完全凸壳边界是非凹的。最后，算法第四步将此不完全凸壳边界的边上的点删除，得到一个凸多边形，即剩下的点全为凸壳顶点。

## 4 算法分析

### 4.1 算法复杂度分析

算法开始时存储了 $n$ 个点的数据，纵观算法的各个步骤，仅仅是将这些点从一个集合移动到另一个集合中，考虑移动过程中产生的临时变量，算法的存储空间小于 $2n$。所以算法的空间复杂度为 $O(n)$。

算法第一步生成初始不完全凸壳点列时，从 $n$ 个点中找到沿 $x$ 坐标、$y$ 坐标方向的 4 个极值点，时间复杂度为 $O(n)$；第二步分类大小为 $n$ 的点集，时间复杂度也为 $O(n)$；第三步迭代生长有向边的迭代次数小于 $\log m$，$m$ 是凸壳顶点数，每一次迭代最多对 $n$ 个点进行分类，时间复杂度为 $O(n\log m)$；第四步删除非凸壳顶点需要遍历检查所有凸壳点，时间复杂度小于 $O(n)$。所以算法的总体时间复杂度为 $O(n\log m)$。特别地，在最坏情况下，即初始点集中所有点都是凸壳点时，算法的时间复杂度达到 $O(n\log n)$。

### 4.2 并行计算开销分析

并行计算开销主要包括系统开销和算法开销。

线程数和调度策略[12]是影响并行性能的两个重要方面。本文主要基于并行模式库(Parallel Patterns Library，PPL)来实现多核架构下的并行计算。PPL 是基于并发运行时的计划和资源管理组件构建的。它使用任务组和工作队列来组织多任务的并行计算。任务组会将任务推入工作窃取队列，而不是为每个任务创建新的线程。计划程序从该队列中移除任务，并用可用的计算资源执行任务。因此，使用并行模式库，在最大程度上降低了线程创建、线程切换等系统开销。

另外，本文算法各并行任务之间相互独立，没有引入大量额外的计算，因此节点间通信开销、同步的开销可以忽略不计。算法通过并行、串行的自适应选择来控制任务粒度，使其既不太大也不太小，以提高算法并行性能。因为如果任务粒度太大，可能存在个别任务的执行时间远远长于其它任务的情况，造成系统的瓶颈；反之，如果任务粒度太小，算法将由于并行计算开销所占比例太大而失去优越性。

### 4.3 实验结果分析

本文算法基于 C＋语言并使用标准模板库(Standard Template Library，STL)和并行模式库实现。

为测试本文算法的并行性能，对 $Z_{3\text{-}2}$ 算法进行了优化，使其与本文算法具有可比性。与本文算法类似，它用颜氏距离代替了欧式距离，用栈结构消除了递归。原 $Z_{3\text{-}2}$ 算法、优化后的 $Z_{3\text{-}2}$ 算法与本文算法在双核处理器平台上的执行时间对比如图 3 所示。图 3 揭示了：

(1)算法对 $10^6$ 数目的点集的凸壳的求解时间接近 200 毫秒，充分说明了本算法的高效性；

(2)优化后，算法的性能已经有较大的提高；

(3)通过并行计算进一步减少了计算时间，当处理器数目更多时，算法执行时间更短。

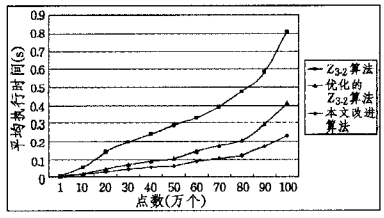


图 3 算法执行时间

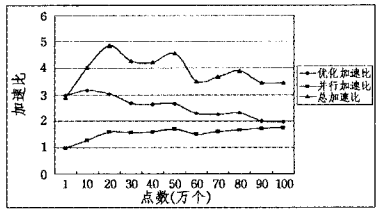


图 4 算法加速比

图 4 显示了算法在优化、改进前后的加速比，实验得出算法的平均并行加速比在 1.55 左右，总体加速比在 3.89 左右。其中，并行加速比在点数较大时接近于 2，这说明了算法的并

(下转第 57 页)

第二种情况下，SemTagP 算法和 ISLPA 算法社区划分的部分结果如表 4 所列。SemTagP 算法将社区划分为 13 个，其中部分社区结果内部包括了多个成分。ISLPA 算法在社区划分时也会出现类似情况，但 ISLPA 算法将内部的成分进一步划分，获取了 18 个社区结构。所以 ISLPA 划分结果中含有两个以上标签前缀一样的社区结构，如表 4 中存在两个标签前缀为 programming 的社区结构。

表 4　算法 SemTagP 和 ISLPA 的部分社区划分结果

| SemTagP 算法 | | ISLPA 算法 | |
|---|---|---|---|
| 标签 | 用户数目 | 标签 | 用户数目 |
| programming | 59 | novel | 44 |
| novel | 55 | psychology | 39 |
| psychology | 41 | programming_1 | 33 |
| operatingSystem | 33 | operatingSystem | 32 |
| economic | 23 | programming_2 | 25 |

**结束语**　针对现有语义标签传播算法 SemTagP 的不足和在线社会网络的特征，本文提出了改进的语义标签传播算法(ISLPA)。ISLPA 算法综合考虑网络中的标签语义关系和用户的各类交互关系，将在线社会网络映射为有向加权网络，并对其进行社区划分。ISLPA 算法接近线性时间复杂度。实验证明，ISLPA 能很好地对真实网络进行社区划分。在下一步工作中，将研究 ISLPA 算法在重叠社区发现上的应用。

## 参 考 文 献


[1] Newman M E J. Detecting community structure in networks [J]. The European Physical Journal B-Condensed Matter and Complex Systems, 2004, 2(38): 321-330

[2] Kernighan B, Lin S. An efficient Heuristic Procedure for Partitioning Graphs[J]. Bell System Technical Journal, 1970, 2(49): 291-307

[3] Clauset A M, Newman M E J, Moore C. Finding community structure in very large networks[J]. Physical Review E, 2004, 6(70): 1-6

[4] Newman M E J. Fast algorithm for detecting community structure in networks[J]. Physical Review E, 2003, 6(69): 5-11

[5] Palla G, Derényi I, Farkas I, et al. Uncovering the overlapping community structure of complex networks in nature and society [J]. Nature, 2005, 7043(435): 814-818

[6] Raghavan U N, Kumara S. Near linear time algorithm to detect community structures in large scale networks[J]. Physical Review E, 2007, 3(76): 106- 115

[7] Ereteo G, Gandon F, Buffa M. SemTagP: Semantic community detection in folksonomies [C] // IEEEWICACM International Conferences on Web Intelligence and Intelligent Agent Technology. 2011: 324-331

[8] 王延鹏. 复杂网络重叠社区发现算法研究[D]. 太原：太原理工大学，2011：24-26

[9] Bastian M, Heymann S, Jacomy M. Gephi: An Open Source Software for Exploring and Manipulating Networks[J]. American Journal of Sociology, 2009, 2: 361-362

[10] Pons P, Latapy M. Computing communities in large networks using random walks[J]. Journal of Graph Algorithms and Applications, 2005, 2(10): 191-218

[11] Mika P. Social Network and the Sematic Web [M]. Semantic Web and Beyond, 2007

[12] Passant A, Laublet P. Meaning of A Tag: A Collaborative Approach to Bridge the Gap Between Tagging andLinked Data[J]. Evolution, 2008, 5(41): 1-5


(上接第 19 页)

行开销较小。从图 4 还可以看出，并行加速比有较为稳定的趋势，而优化加速比抖动较大。这是因为算法的优化过程改变了算法的基本数据结构和执行流程，而并行化的过程不改变算法自身结构。

**结束语**　本文基于 $Z_{3\text{-}2}$ 算法提出了改进的并行凸壳算法。首先，基于颜氏距离来分类点集和找下一个凸壳点，效率优于使用欧氏距离和正负划分；其次，以较小的并行开销实现多核架构下的并行计算；再次，利用阈值判断，避免了深层递归和小粒度任务的分解；最后，从点集的内部开始，逐步迅速地删除大量的非凸壳点，避免了大量的无效重复计算。理论和实验结果表明，改进算法利用颜氏距离减少了计算量，在多核架构下充分利用并行计算资源对海量数据集进行求解，并行开销小，获得了比原 $Z_{3\text{-}2}$ 算法更高的效率。因此，算法不仅能求解一般性凸壳问题，而且能更快速地求得海量($10^6$ 个以上)的点集和含有大量非凸壳点的点集的凸壳。本文提出的理论和思想可以应用到三维和更多维的情况中，这也是下一步的研究目标。

## 参 考 文 献


[1] 周培德. 计算几何-算法分析与设计(3 版)[M]. 北京：清华大学出版社，2008

[2] Graham R L. An efficient algorithm for determining the convex hull of a finite planar set[J]. Information Processing Letters, 1972, 1(1): 132-133

[3] 金文华，何涛，刘晓平，等. 基于有序简单多边形的平面点集凸包快速求取算法[J]. 计算机学报，1998，21(6)：533-539

[4] 赵军，曲仕茹. 平面点集凸壳的快速算法[J]. 计算机工程与应用，2009，45(1)：56-58

[5] Reif J H, Sen S. Optimal parallel randomized algorithms for tree-dimensional convex hulls and related problems [J]. SIAM J. Comput., 1992, 3(21): 466-485

[6] 张三元，马利庄. 平面散乱点集凸包并行算法[J]. 浙江大学学报：工学版，1999，33(4)：432-440

[7] 郝小柱，胡祥云，戴光明，等. 平面点集凸包的并行算法研究[J]. 计算机应用，2005，25(10)：2462-2464

[8] 周启海，黄涛，林珣，等. 基于动态基线倾角与基线距离最大化的凸壳并行新算法[J]. 计算机科学，2008，35(4)：244-247

[9] 陈伟，杜凌霞，陈红. 多核架构下的数据处理算法优化策略综述[J]. 计算机科学与探索，2011，5(12)：1057-1075

[10] 邓亚丹，景宁，熊伟. 基于共享 Cache 多核处理器的 Hash 连接优化[J]. 软件学报，2010，21(6)：1220-1232

[11] 张忠武，吴信才. 海量数据凸壳快速优化算法研究[J]. 微计算机信息，2011，27(8)：194-196

[12] 张思乾，程果，陈荦，等. 多核环境下边缘提取并行算法研究[J]. 计算机科学，2012，39(1)：295-298