Journal of Computer Applications
计算机应用,2013,33(8):2177 - 2183
ISSN 1001-9081
CODEN JYIIDU
2013-08-01
http://www.joca.cn

文章编号:1001-9081(2013)08-2177-07
doi:10.11772/j.issn.1001-9081.2013.08.2177

月面地形重构系统中的并行 Delaunay 算法设计

王 喆[1*],高三红[1],郑慧英[1],李立春[2]
(1.北京跟踪与通信技术研究所,北京 100094; 2.北京航天飞行控制中心,北京 100094)
(*通信作者电子邮箱 wzhe06@163.com)

摘 要:三角剖分过程是影响三维重建系统实时性的瓶颈之一,为提高三角剖分速度,基于共享内存多核计算机设计并实现了并行 Delaunay 算法。该算法在分治三角剖分算法的基础上,通过改进子三角网归并过程及 Delaunay 三角网优化过程避免了并行计算中的数据竞争问题。利用月面仿真实验场真实地形数据在 50 万到 500 万不同规模的点云数据集上进行了实验,加速比最高可达 6.44。除此之外,对算法复杂度、加速比以及并行效率进行了全面分析,并将算法实际应用于月面地形重构系统,实现了虚拟地形的快速构建。

关键词:Delaunay 算法;并行计算;地形重构;开放多处理;多维树
中图分类号:TP311.1 **文献标志码**:A

Parallel Delaunay algorithm design in lunar surface terrain reconstruction system

WANG Zhe[1*], GAO Sanhong[1], ZHENG Huiying[1], LI Lichun[2]
(1. *Beijing Institute of Tracking and Telecommunication Technology, Beijing* 100094, *China*;
2. *Beijing Aerospace Control Center, Beijing* 100094, *China*)

Abstract: Triangulation procedure is one of the time bottle-necks of 3D reconstruction system. To increase the speed of triangulation procedure, a parallel Delaunay algorithm was designed based on a shared memory multi-core computer. The algorithm employed divide-and-conquer method and improved conquer procedure and Delaunay mesh optimization procedure to avoid data competition problem. Experiments were conducted on datasets with range from 500000 to 5000000 gathered on the lunar surface simulation ground, and the speedup of the algorithm reached 6.44. In addition, the algorithm complexity and parallel efficiency were fully analyzed and the algorithm was applied in the lunar surface terrain reconstruction system to realize fast virtual terrain reconstruction.

Key words: Delaunay algorithm; parallel computing; terrain reconstruction; Open Multiple Processing (OpenMP); K-Dimension tree (KD-tree)

0 引言

三角剖分是利用离散点云进行地形重构的关键步骤,Delaunay 算法经过对离散点云的三角剖分形成满足 Delaunay 性质的不规则三角网,进而通过纹理渲染形成虚拟地形。串行 Delaunay 算法的时间复杂度在 $O(n \text{ lb } n)$ 到 $O(n^2)$ 之间,由于进行地形重构的三维点云规模往往达到几百万甚至上千万的规模,三角剖分过程在大数据量条件下容易成为系统时间瓶颈,因此对 Delaunay 算法的优化研究一直在进行。随着并行计算技术的蓬勃发展,并行 Delaunay 算法开始成为研究的热点。但已有对并行 Delaunay 算法的研究多利用公共数据集或仿真数据集进行算法实验,很少应用于实际系统,在算法设计上多采用基于分布式内存架构的逐点插入方法或多核分区法。本文基于分区归并方法利用 OpenMP 共享内存并行计算框架设计了并行 Delaunay 算法,从算法设计上解决了数据竞争问题,在 16 核计算机上最高达到 6.44 的加速比,并实际应用于月面地形重构系统,针对探月任务的特点,提出了增量式三角网构建方法。此外,利用月面仿真实验场获取的真实地形数据进行了多数据集上的三角剖分实验,根据实验结果对算法复杂度、加速比、并行效率进行了深入的分析,最终给出三角剖分结果及月面地形重构系统生成的虚拟地形。

1 相关工作

1.1 月面地形重构系统

月面地形重构系统是月球探测过程中重构月球车所探测地形环境并进而进行遥操作的基础,基于立体视觉的行星表面地形重构系统已被成功地运用在美国火星探测车(Mars Exploration Rover,MER)[1]、欧洲空间局的星球表面探测视觉导航系统[2]等项目中。月球车在漫游过程中,利用自身的立体视觉系统探测周围环境,将双目立体图对下传到地面,由月面地形重构系统重建其所处的三维地形环境。月球车在漫游过程中,每隔一段路程会选择一个观测点,在每个观测点拍摄近 100 幅立体图对,共生成 10 万到 20 万个地形特征点,随着探测过程的进行,多个探测点的数据不断积累,地形特征点云规模将达到百万甚至千万量级,大数据量条件下如何进行快速三维地形重建成为亟待解决的问题。为此,美国国家航空航天局(National Aeronautics and Space Administration,NASA)将云计算技术应用于 MER 项目中的火星三维地形重建系统[3-4]和 DESDynI 地理信息系统[5]中,进行并行立体图对的处理、三角剖分和地形渲染,取得了良好的效果。相比火星探

收稿日期:2013-02-23;**修回日期**:2013-04-16。 **基金项目**:国家自然科学基金资助项目(61173080)。
作者简介:王喆(1988 -),男,河南濮阳人,硕士研究生,主要研究方向:并行计算、虚拟现实; 高三红(1965 -),男,河南焦作人,研究员,硕士,主要研究方向:计算机仿真; 郑慧英(1979 -),女,河南开封人,工程师,硕士,主要研究方向:计算机网络技术; 李立春(1978 -),男,河北元氏人,工程师,博士,主要研究方向:计算机视觉。

测计划中 20 min 以上的通信延时，月地通信的单程延时仅为 1.35 s，这对月面地形重构系统的设计提出了新的挑战。本文对并行 Delaunay 算法进行研究，解决了月面地形重构系统在三角剖分环节的时间瓶颈问题。

1.2 并行 Delaunay 算法

Delaunay 三角网被证明是二维平面三角网中唯一的、最“好”的三角网[6-7]。Delaunay 三角网的定义如下：

定义 1 Delaunay 三角网。给定顶点集合 $V = \{V_1, V_2, \cdots, V_n\}$，边集合 $DT(V)$，如果 $DT(V)$ 满足下述条件，则称 $DT(V)$ 是 Delaunay 三角剖分：设边 $e_{ij} = (V_i, V_j)$，对于任意的 $\triangle V_iV_jV_k$，如果 $e_{ij}, e_{ik}, e_{kj} \in DT(V)$，且 $\triangle V_iV_jV_k$ 的外接圆中不包含 V 中除 V_i, V_j, V_k 和 $\triangle V_iV_jV_k$ 内部顶点外的其他顶点。

并行 Delaunay 算法主要分为分区归并法、逐点插入法和多核分区法。Aggarwal 等[8]设计了并行二维分区归并算法，其归并过程是在两子区域剖分结束后再进行处理。Chen 等[9]在分区时没有进行标准的二分，而是在两子区域交汇处保留重叠点，在归并时不用再进行归并运算，只需将重叠三角网部分进行去除，提高了算法的并行度但增加了算法复杂度。Okusanya 等[10]利用并行逐点插入法实现了三维点云的网格化，由于每次插入新点都需要进行三角网的优化，优化过程往往涉及多个区域，因此多个处理器之间需要进行频繁通信，如何解决通信延迟问题是该类算法的重点。多核分区法关注点云的分解与归并，将三角剖分任务平均分配给多个计算节点进行，Chrisochoides 等[11-12]基于分布式架构设计了并行 Delaunay 算法，Blandford 等[13]完成了共享内存的多核处理器上的多核分区算法实现。

并行 Delaunay 算法需要解决的主要问题是如何在合适的并行计算平台上进行任务分解，最大限度减少并行计算带来的通信延迟，提高并行效率。之前的研究中为解决数据同步与竞争的问题，增加了算法的复杂程度，大部分研究没有给出详尽的算法复杂度、加速比以及并行效率分析。本文基于通信代价极小的共享内存多核计算平台，在算法设计上避免了数据冲突，设计并实现了基于分区归并思想的并行 Delaunay 算法。

2 并行 Delaunay 算法设计

并行 Delaunay 算法基于串行分区归并 Delaunay 算法[14]进行设计，首先利用 KD 树（K-Dimension tree，KD-tree）方法完成点云划分与任务分配，每个处理器采用串行分区归并算法完成当前子点云的三角剖分，再对子三角网进行任务级归并和优化。

2.1 基于 KD 树的点云划分及任务分配方法

KD 树[15]是一种多维数据结构，被广泛用于高维空间数据索引和查询，它通过超平面把一个空间递归划分为两个子空间。

设 D 为样本维数，d 为分割维索引，$h \in [d_{\min}, d_{\max}]$，其中 $d_{\min}$ 和 $d_{\max}$ 分别为超矩形第 d 维的下界和上界。若 D 维超矩形被一个正交于第 d 维的超平面分割为两个子超矩形，则这个超平面可以表示为：

$$H = \{|\boldsymbol{x} \in \mathbf{R}^D; x_d = h|\}$$

被超平面分割形成的两个子超矩形 R_l 和 R_r 可分别表示为：

$$R_l = \{|\boldsymbol{x} \in \mathbf{R}^D; x_d \leqslant h|\}$$

$$R_r = \{|\boldsymbol{x} \in \mathbf{R}^D; x_d > h|\}$$

在构建 KD 树过程中，选择分割维 d 及在分割维上确定分割超平面位置 h，具有多种策略[8]。本文旨在根据地形点的横坐标及纵坐标对三维地形点云进行分割形成用于并行三角剖分的子点云，可以利用二维空间的 KD 树分割方法进行处理。月面纹理的稀疏程度差异导致地形特征点云的疏密程度差异明显，如果分割维和分割点的选择不当，会导致各子点云的数目不均，进而造成各计算节点任务的负载不平衡，影响并行效率。本文对中点分割策略进行改进，提出了滑动分割点策略。

在介绍分割策略前先明确以下几个概念。1）长宽比：超矩形最长边与最短边的比值。2）延展度：指定多维空间上的某一维，点集中的点在该维上的最大坐标与最小坐标的差。3）分割维：在某一次分割中被选定的用于进行点集分割的维。

设当前点云点集为 V，$|V|$ 表示 V 中点的个数，C 表示点云的规模阈值，即当 $|V| \leqslant C$ 时停止分割过程，否则利用滑动分割点策略进行分割。

算法 1 滑动分割点策略。

步骤 1 遍历 V 中的点，选择延展度最大的维作为分割维，设分割维的延展度为 E；选定分割维中点为分割点。

步骤 2 按照分割点将 V 划分为左右子点集 V_L 和 V_R。

步骤 3 计算 $S = |1 - |V_L|/|V_R||$，当 $S < 0.2$ 时，跳转至步骤 5。

步骤 4 计算 $B = (|V_L| - |V_R|)/(|V_L| + |V_R|)$，分割点按照 B 的符号方向移动。若步骤 4 过程不超过 3 次，跳转至步骤 2。

步骤 5 结束分割过程。

滑动分割点策略对中点分割策略[8]进行了优化，根据左右子点云数量的差别对分割点进行调整，大量减少了由于点云分布不均造成的任务分解不均衡问题。最坏情况下增加了最多 3 次点云遍历，根据真实地形数据的实验，平均遍历次数为 1.35。算法中点云规模阈值 C 的选取与处理器个数 P 相关。当任务个数在 $3P$ 到 $4P$ 之间时，并行 Delaunay 算法的并行效率最高，因此点云规模 C 一般取 $|V_L|/3P$。图 1 显示了基于 KD 树的点云划分过程，在划分过程中，KD 树非叶节点的任务会压入子三角网归并任务队列，叶节点的任务被压入子点云剖分任务队列，用于任务的并行执行。

2.2 子三角网归并策略

各处理器利用分区归并法完成各子点云的三角剖分过程后，需要对各子三角网进行归并，使之成为整张三角网。传统的归并方法需要首先查找两子三角网凸壳的上下公切线，再对子三角网与公切线围成的凹多边形进行剖分，完成归并过程。本文对其进行优化，在查找上下公切线的同时利用过渡连线完成三角网的归并过程。在给出详细归并策略之前首先给出一组定义：

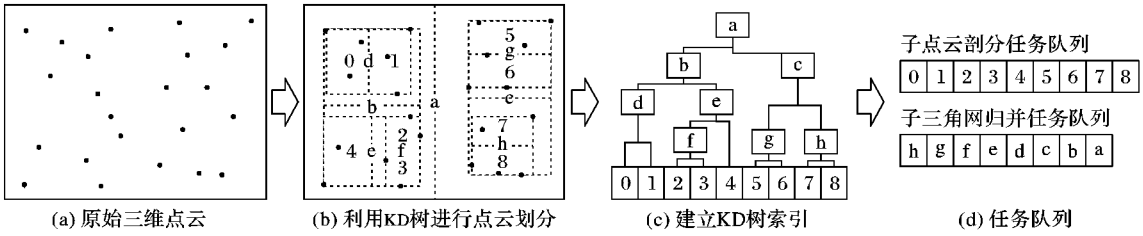
定义 2 凸壳。对于点集 V，若存在凸多边形，它的顶点均为点集 V 中的点，且 V 中所有其他点都在该凸多边形的内部，则该凸多边形称为点集 V 的凸壳，记为 $CH(V)$。

定义 3 后继点与前趋点。设 V_i 为点集 V 的凸壳 $CH(V)$ 中的点，从 V_i 开始，以逆时针方向遍历 $CH(V)$ 中的顶点，所遇到的第一个顶点称为 V_i 的后继点，记为 $SUCC(V_i)$；同样，若

以顺时针方向遍历,则把遇到的第一个顶点称为是 V_i 的前趋点,记为 $PRED(V_i)$。

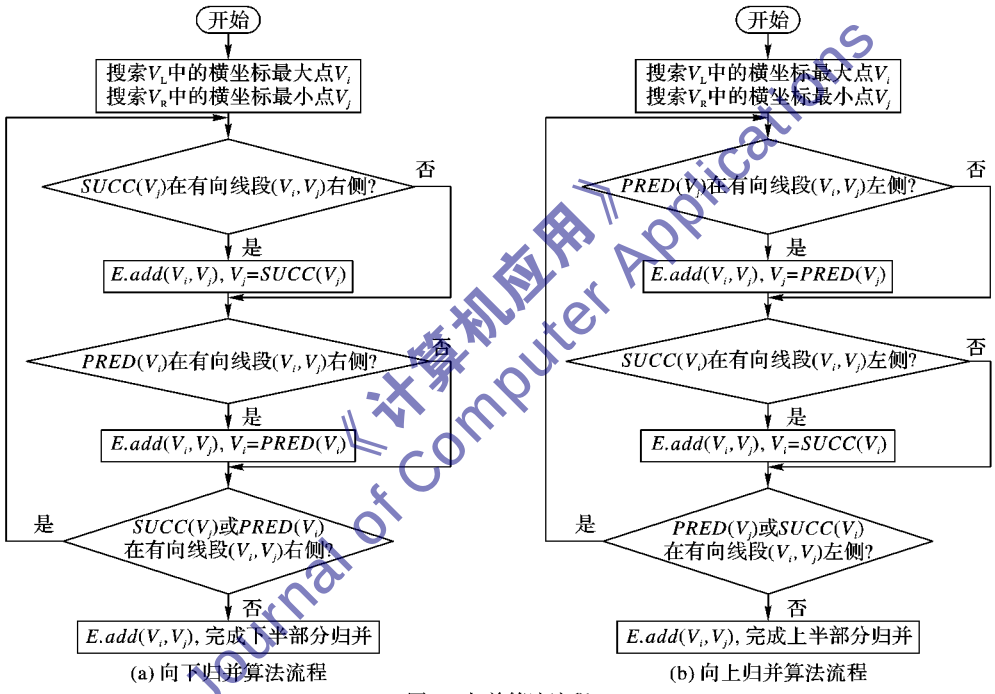
设需要归并的两子三角网点集分别为 V_L 和 V_R,V_R 中点的横坐标均大于 V_L,边集合 E 初始为空,$E.add(V_1,V_2)$ 表示将边(V_1,V_2)加入边集合 E,则 V_L 和 V_R 的归并算法流程如图 2 所示:首先搜索 V_L 的最右点 V_i 和 V_R 的最左点 V_j,连接 V_iV_j,分别向上和向下进行归并过程。在归并过程结束后,边集合 E 将左右两子三角网连接起来,共同组成了归并后的三角网。归并过程是基于 KD 树的任务划分过程的逆过程,从根节点向叶节点的过程即任务划分的过程,从叶节点到根节点的过程是三角网的归并过程。

子三角网归并过程可以并行执行,但由于不同深度之间的任务节点存在数据依赖,必须在深一层的归并过程结束后才能够进行上层的归并过程。每一轮内的归并任务分配在不同处理器上并行执行,每轮之间串行执行。图 3 为 4 核条件下的并行子三角网归并过程示意图。



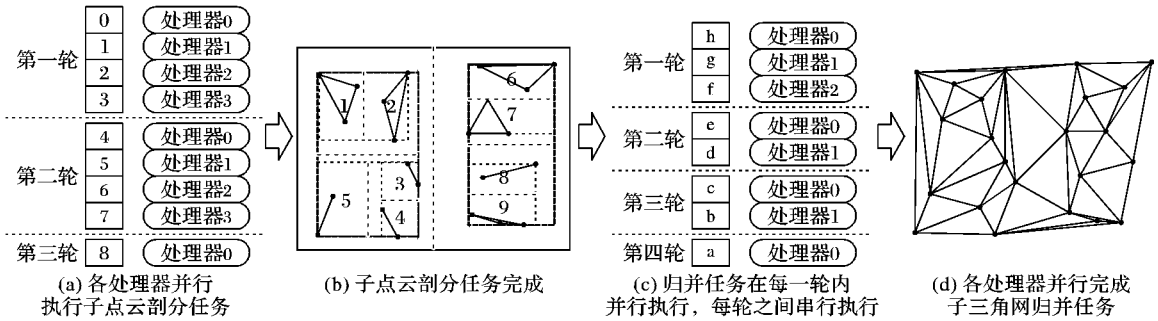
(a) 原始三维点云　(b) 利用KD树进行点云划分　(c) 建立KD树索引　(d) 任务队列

图 1　基于 KD 树的点云划分过程示意图



(a) 向下归并算法流程　(b) 向上归并算法流程

图 2　归并算法流程



(a) 各处理器并行执行子点云剖分任务　(b) 子点云剖分任务完成　(c) 归并任务在每一轮内并行执行，每轮之间串行执行　(d) 各处理器并行完成子三角网归并任务

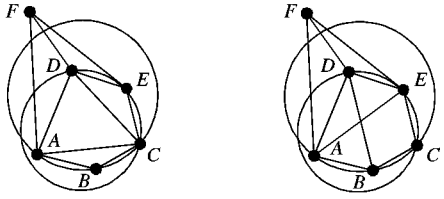
图 3　并行子三角网归并过程示意图

2.3　Delaunay 优化过程的数据竞争问题

Delaunay 优化过程是指将普通三角网优化为满足 Delaunay 性质的三角网的过程。较为常用的方法是运用 Delaunay 三角网的空外接圆性质,即对由两个有公共边的三角形组成的四边形进行判断,如果其中一个三角形的外接圆中含有第四个顶点,则交换由两个三角形所构成的四边形的对角线。在交换对角线后,会改变原有三角网的拓扑结构,新生成两个三角形,这时需要对这两个三角形进行空外接圆检测,递归进行此过程,直到新生成的两个三角形也满足空外接圆性质为止。

并行 Delaunay 优化过程在进行空外接圆检测过程中可能发生数据竞争问题。以图 4 为例,如果同时对 $\triangle ABC$ 和 $\triangle CDE$ 做空外接圆检测,$\triangle ABC$ 外接圆中包含点 E,$\triangle CDE$ 外接圆中包含点 A,结果均不满足空外接圆性质。这时需要同时

交换四边形 $ABCD$ 及四边形 $ACED$ 的对角线，线段 AE 与 BD 交叉造成错误结果。



(a) 原三角网拓扑结构 (b) 数据竞争问题造成的错误结果

图 4 Delaunay 优化过程的数据竞争问题

要解决并行 Delaunay 优化算法中的数据竞争问题有两种方法，最直接的方法是加入互斥锁机制，利用 OpenMP 语言中的 Critical 指导语句声明代码中的临界区，在当前进程执行临界区代码时会阻塞其他进程。由于每次空外接圆的判定与对角线的交换都需要进入临界区，其他进程大部分时间都处在阻塞状态，实验证明在 16 核计算机上运行 Delaunay 优化过程加速比仅为 2.3 ~ 2.6，远远没有发挥并行计算的优势。

第二种方法是在算法设计上消除数据竞争的可能性。并行 Delaunay 优化过程发生数据竞争的原因是因为各并行分区在执行空外接圆检测时，拓扑结构的改变递归地扩散到其他区域，导致与其他区域的优化过程发生冲突。这种冲突本质上是不同处理器对相同数据操作造成的冲突。要解决该问题，就需要防止处理器对数据的操作扩展到其他处理器所辖的范围。为此，可以把优化过程从全局优化分散到每次归并过程中。在形成整张三角网之前，在每次归并完成后对新加入三角网的边及其所在四边形的边进行优化，由于此时每个处理器对应的三角网还没有与其他处理器所辖三角网连通，数据竞争问题必然不会发生，称该过程为并行局部 Delaunay 优化过程。并行局部 Delaunay 优化过程的总计算量相比三角网完全生成后进行的全局优化过程的计算量有所增加，但由于从根本上解决了数据竞争问题，因此对于并行 Delaunay 算法加速比的提升效果显著。

2.4 并行 Delaunay 算法框架及伪码表示

2.1 ~ 2.3 节分析了 Delaunay 算法各关键步骤的并行化实现，并行 Delaunay 算法首先利用 KD 树进行任务分解，在各区域利用分区归并算法并行完成三角剖分任务后，进行任务级的归并过程，在各任务所辖子三角网归并的过程中，为保证生成三角网满足 Delaunay 性质，需要在每次归并过程完成后，对新生成的边进行 Delaunay 优化。在 KD 树根节点的任务完成后，形成整张 Delaunay 三角网，完成点云的三角剖分任务。并行 Delaunay 算法的伪码表示如下：

算法 2 并行 Delaunay 算法。

```
points = data_read()                                    //数据读取
tasks = task_decompose(points)                          //任务分解
foreach task in tasks                                   //任务分类
  if task.type == TRIANGULAR_TASK
    triangular_tasks.add(task)
  else if task.type == MERGE_TASK
    merge_tasks.add(task)
  end
end
#pragma omp parallel for num_threads(thread_num) shared(points)
foreach task in triangular_tasks                        //并行子点云三角剖分
  triangular(task)
end
foreach task in merge_tasks                             //任务级归并过程
  if thislevel!=task.level
    #pragma omp parallel for num_threads(thread_num) shared
    (points)
    foreach task in thislevel_tasks
      newedges = merge(task.leftmesh, task.rightmesh)
      foreach edge in newedges
        optimize(edge)
      end
    end
    thislevel.clear()
    thislevel++
  end
  thislevel_tasks.add(task)
end
```

程序主要分为任务分解、子点云三角网格化，以及子三角网归并与优化三个主要步骤。程序中使用三个函数功能如下：

void triangular(Task task)：串行三角网格化函数，函数接受一个点云三角网格化任务 task，利用分区归并算法完成三角网格化过程；

vector < Edge > merge(Mesh leftmesh, Mesh rightmesh)：子三角网归并函数，该函数利用 2.2 节描述的子三角网归并方法进行子三角网的归并，返回归并过程产生的边集合；

void optimize(Edge edge)：三角网优化函数，接收待优化的边，对已改变为对角线的四边形进行空外接圆检测，如果不满足空外接圆性质，则交换对角线，并递归地对四边形四边进行优化。

伪码中以#pragma omp parallel 开头的语句是 OpenMP 并行编程语言中的预编译指令，将 for 循环通过并行编译分解到多核中并行执行。

2.5 并行 Delaunay 算法的增量式应用

月球车在进行月面探测的过程中，每行进 10 m 进行一次地形勘察，在多个俯仰角利用导航相机拍摄近 100 幅立体图对下传到地面。地面操控中心需要将新地形与原有地形进行拼接后得到全部已勘测区域的月面地形图。随着地形数据的不断积累，如果每次地形重构均对全部原有地形数据进行重新处理，会导致系统的实时性与数据量累次增大的矛盾不断突出。

为解决该问题，可在 2.4 节定义的并行 Delaunay 算法基础上经过改进形成增量式 Delaunay 算法。在三角剖分阶段，基于分区归并思想的 Delaunay 算法本身就是通过先分区再进行归并的过程递归生成三角网，因此在获得新探测点的三维地形点云数据后，首先运用并行 Delaunay 算法对新点云数据进行三角剖分，再调用 merge 函数与原三角网进行合并形成新的 Delaunay 三角网。设 V 为新点云中的特征点集合，M 为原三角网，则增量式 Delaunay 算法的具体步骤如下：

步骤 1 将 V 中的点逐一变换到与 M 相同的世界坐标系下；

步骤 2 利用并行 Delaunay 算法将 V 进行三角剖分，形成三角网 M'；

步骤 3 利用归并算法合并 M 与 M'，即 merge(M,M')；

步骤 4 优化合并后的三角网，形成 Delaunay 三角网。

并行 Delaunay 算法的增量式应用避免了每次地形重构都将全部地形点云进行三角剖分的时间开销，而是仅进行新加入点云的三角剖分，提高了系统实时性。

3 实验及分析

3.1 并行 Delaunay 算法时间复杂度分析

通过分析并行 Delaunay 算法的时间复杂度，可以明确影

响算法加速比及并行效率的关键因素。设点云规模为 n,处理器个数为 P,任务划分时子点云三角剖分任务个数为 C。算法共分为任务分解、子点云三角剖分和任务级三角网归并三个阶段。

基于 KD 树方法的任务分解过程,任务个数为 C, KD 树深度为 $\mathrm{lb}(C)$,每层分解需要遍历所有点,可得任务分解过程的时间复杂度为 $n\,\mathrm{lb}(C)$。

子点云三角剖分过程中子三角网平均规模为 n/C,利用分区归并算法进行三角剖分,递归深度为 $\mathrm{lb}(n/C)$,单个任务的时间复杂度为 $(n/C)\mathrm{lb}(n/C)$,C 个任务总的时间复杂度为 $n\,\mathrm{lb}(n/C)$,在 P 个处理器上并行执行,故并行子点云三角剖分过程的复杂度为 $(n/P)\mathrm{lb}(n/C)$。

任务级归并过程是任务分解的逆过程,需要进行 $\mathrm{lb}(C)$ 层归并,每层在最坏情况下遍历所有点,故串行时间复杂度为 $n\,\mathrm{lb}(C)$,由于接近根节点时每层的任务数小于处理器个数,故并行时间复杂度为

$$\left(1+\frac{1}{2}+\frac{1}{4}+\cdots+\underbrace{\frac{1}{P}+\cdots+\frac{1}{P}}_{\text{共}\lceil \mathrm{lb}(C)-\mathrm{lb}(P)\rceil\text{个}}\right)n \leqslant \left(2+\frac{\mathrm{lb}(C)}{P}\right)n$$

故任务级归并过程的时间复杂度为 $\left(2+\frac{\mathrm{lb}(C)}{P}\right)n$。

并行算法总的时间复杂度用下式表示:

$$T(n)=A_1\cdot n\cdot \mathrm{lb}(C)+A_2\frac{n}{P}\mathrm{lb}(n/C)+A_3\left(2+\frac{\mathrm{lb}(C)}{P}\right)n$$

其中 A_1、A_2、A_3 表示不同阶段复杂度常数系数。显然,由上式可知,计算时间随着问题规模 n 增大而增加,随着处理器个数 P 的增加而减小。当任务个数 C 增大时,子三角网剖分任务的时间缩短,但任务分解与归并的时间会加长;C 减小则情况相反。经过 10 组不同数据集的实验表明 C 的选择对于算法的影响不大,当 C 取 P 的 3 到 4 倍时算法通常拥有相对最高的加速比。

3.2　算法各阶段分析

实验中将文中设计的基于分区归并的并行 Delaunay 算法简称为 DCPD(Divide and Conquer Parallel Delaunay)算法,分为任务分解、子点云三角剖分、任务级三角网归并三个阶段,其中任务分解串行执行,子点云三角剖分完全并行执行,任务级三角网归并部分并行执行。利用最高达 500 万点云规模数据进行算法各阶段的测试分析。实验的编译环境为 Visual Studio 2008,操作系统为 Windows Server 2008,并行程序采用 OpenMP 实现。硬件平台采用联想 R525G3 服务器;CPU 为 Intel Xeon E5620,双路共 16 核,主频 2.4 GHz;内存 8 GB。测试结果如表 1 所示。

表 1　DCPD 算法各阶段消耗时间　　ms

点云规模	任务分解		子点云三角剖分		任务级三角网归并		总时间	
	串行	4 核	串行	4 核	串行	4 核	串行	4 核
500 000	43	41	1 415	383	1003	407	2 461	831
3 000 000	642	640	11 325	2 954	5 479	2 556	17 446	6 150
5 000 000	1 290	1 290	17 212	4 563	7 818	3 939	26 320	9 792

利用表 1 的实验结果,进行 DCPD 算法各阶段所占运行时间比例、算法加速比、并行效率分析,通过对不同规模数据集下的实验结果进行平均后得出表 2 的分析结果。

表 2　DCPD 算法各阶段实验结果分析

算法阶段	时间比例/%		加速比		并行效率	
	串行	4 核	串行	4 核	串行	4 核
任务分解	3.45	9.51	—	1.00	—	0.25
子点云三角剖分	62.60	46.91	—	3.76	—	0.94
任务级三角网归并	33.95	43.58	—	2.19	—	0.55
合计	100	100	—	2.82	—	0.71

通过分析表 1 与表 2 结果可得出 DCPD 算法在各阶段的并行特点,算法在任务分解阶段采用串行任务分解的方式,串行算法与并行算法在任务分解阶段耗时相同,但由于该阶段所占比例低于 10%,因此对于 DCPD 算法的加速比影响较小;在子点云三角剖分阶段,各子点云间完全没有任务依赖,并行效率达到 0.94,接近完全并行的理论并行效率,该阶段串行算法中运行时间所占比例高达 62.6%,有效提高了 DCPD 算法的加速比;在任务级三角网归并阶段,采用部分并行的方式,并行效率为 0.71,最后几轮归并无法利用所有的计算节点,加速比相对较低。

3.3　并行 Delaunay 算法对比实验

在并行 Delaunay 算法研究中,分区归并法与逐点插入法一直是两种主要的研究方向。2011 年,Nguyen 等[16]在自己实现的 Galois[17] 并行库框架下实现了基于逐点插入的 Delaunay 算法(简称 Nguyen 算法),数据集采用通过图像边缘检测获得的 1 733 360 个特征点,在多种共享内存多核环境下进行了系统实验。本文采用同样数据集规模下的 DCPD 算法实验结果与之进行对比,对比结果如表 3 所示。

表 3　并行 Delaunay 算法实验结果对比

算法	平台	运行时间/s	加速比
Nguyen 算法	4 核 Intel Xeon X5570 处理器,主频 2.93 GHz	6.29	2.35
	4 核 AMD Opteron 8384 处理器,主频 2.7 GHz	10.51	2.65
	8 核 Sun UltraSPARC T2 Plus 处理器,主频 1.4 GHz	27.26	5.48
DCPD 算法	4 核 Intel Xeon X5570 处理器,主频 2.93 GHz	3.994	2.89
	8 核 Intel Xeon E5620 处理器,主频 2.4 GHz	2.743	4.01

通过表 3 的对比结果可以发现,DCPD 算法在运行时间上远低于 Nguyen 算法,主要原因是逐点插入法在每次插入新点时需要查找该特征点所在三角形,造成时间开销大于分区归并的逐层归并过程。而 Nguyen 算法在 8 核环境下虽然运行时间远大于 DCPD 算法,但加速比却达到 5.48,显示了 Nguyen 算法较好的并行性。

3.4　算法整体加速比与并行效率分析

采用 3.2 节的实验环境,利用 50 万到 500 万的 10 组数据集进行 DCPD 算法整体加速比与并行效率测试,测试结果如表 4 所示。

图 5 显示了各规模数据集上,利用并行 Delaunay 算法进行实验,处理器数量与处理时间的关系。由图可以看出利用

OpenMP 实现的并行 Delaunay 算法在各规模点云数据上均对三角剖分过程起到了明显的加速作用，随着处理器数量的增加，处理时间逐渐减少。

图 6 显示了处理器数量与算法加速比之间的关系。随着处理器个数增加，加速比也随之增大，最大可达 6.44。但当处理器个数超过 4 之后，加速比的增长速度逐渐趋缓。

表 4 并行 Delaunay 算法处理不同规模点云数据时不同处理器数量的处理时间 ms

点云规模	处理器个数								
	1	2	4	6	8	10	12	14	16
500 000	2 461	1 282	831	778	599	506	455	430	416
1 000 000	5 099	3 250	1 756	1 534	1 429	1 054	950	885	791
1 500 000	7 841	4 996	2 862	2 164	2 042	1 571	1 436	1 351	1 271
2 000 000	10 735	6 270	3 994	3 083	2 743	2 243	2 017	1 868	1 821
2 500 000	14 161	8 797	5 669	4 181	3 867	2 981	2 716	2 479	2 488
3 000 000	17 446	10 231	6 150	5 224	4 472	3 804	3 756	3 234	3 114
3 500 000	19 210	11 133	6 883	6 127	5 399	4 572	4 328	4 303	4 020
4 000 000	21 840	12 226	7 774	6 851	5 728	5 054	4 700	4 475	4 373
4 500 000	23 616	12 606	9 032	7 147	6 158	5 244	4 945	4 627	4 760
5 000 000	26 320	14 145	9 792	8 066	7 278	5 976	5 547	5 270	5 382

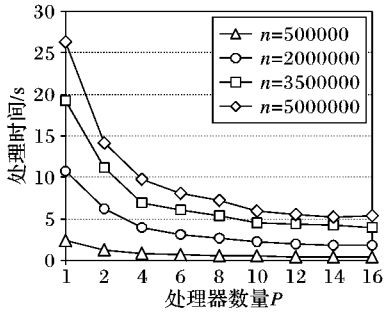


图 5 处理器数量与处理时间关系

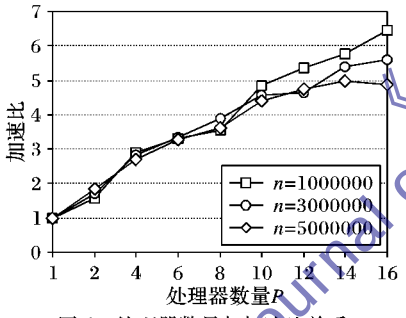


图 6 处理器数量与加速比关系

图 7 显示了并行效率与处理器个数之间的关系，从中看出随着处理器个数的增加，并行效率成下降趋势。

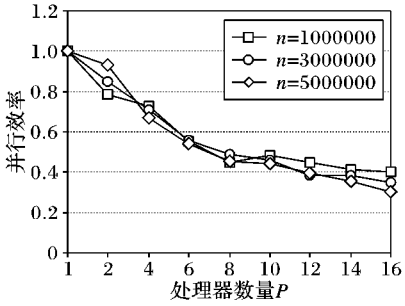


图 7 并行效率 E 与处理器个数 P 的关系

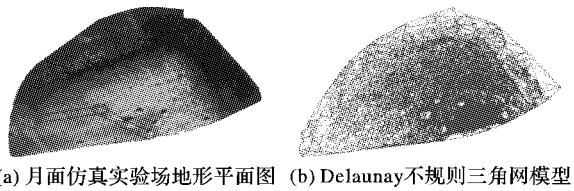
造成并行效率下降的原因有如下两点：

1）共享内存架构的内存瓶颈问题。在处理器规模超过 4 之后，多个处理器同时进行内存的读写，达到内存带宽极限。该问题是基于 OpenMP 的共享内存多处理器架构的体系瓶颈问题，在算法设计上很难解决。

2）任务分解与任务级归并过程所占的时间比例随着程序并行度的增加而增大。由于任务分解与任务级归并过程无法做到 100% 的并行化，而随着处理器数量的增加，任务划分的粒度更细，任务分解与归并所占时间比例增加，造成程序并行度的降低，从而导致并行效率的下降。

3.3 三角网格与虚拟地形显示

利用月球车在月面仿真实验场拍摄的立体像对数据，进行了月面地形重构系统实验，实验结果如图 8 所示。图 8（a）是月球车在月面仿真实验场拍摄的地形立体像对经过拼接后形成的地形平面图。经过月面地形重构系统生成三维地形特征点云，利用本文的并行 Delaunay 算法形成 Delaunay 三角网（如图 8（b）所示），利用虚拟现实模型语言（Virtual Reality Modeling Language，VRML）初步形成了月面地形示意图（如图 8（c）所示），最终通过地形渲染由月面地形重构系统生成月面三维环境（如图 8（d）所示）。


(a) 月面仿真实验场地形平面图 (b) Delaunay不规则三角网模型

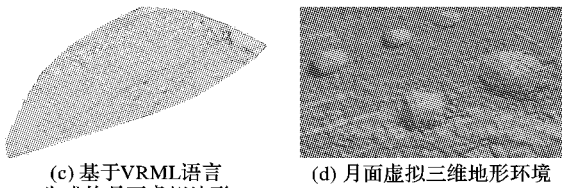

(c) 基于VRML语言生成的月面虚拟地形 (d) 月面虚拟三维地形环境

图 8 月面仿真地形重建过程

4 结语

本文对 Delaunay 算法的并行化方法进行研究，在任务分解过程中提出了滑动分割点策略进行点云划分，在子三角网归并过程改进了子三角网归并及 Delaunay 三角网优化算法。利用 OpenMP 语言在 16 核共享内存计算机上进行了算法实验，最高达到 6.44 的加速比；对算法复杂度、加速比及并行效率进行分析，得出内存的带宽瓶颈影响算法并行效率的结论；在下一步的研究中将对算法的内存占用率进行优化。将算法实际应用于月面地形重构系统，大幅度提高了系统在三角剖分过程的速度，在月面仿真实验场实验中实时精确地生成地形三角网，进而生成月面三维地形。

参考文献：

[1] STEVEN W S. Mars exploration rovers [J]. Bulletin of the Ameri-

can Astronomical Society, 2008, 40: 500.
[2] VERGAUWEN M, POLLEFEYS M, van GOOL L J. A stereo-vision system for support of planetary surface exploration [J]. Machine Vision and Applications, 2003, 14(1): 5 – 14.
[3] CHANG G, LAW E, MALHOTRA S. Demonstration of LMMP workflow system using cloud computing architecture [C]// SECLOUD'11: Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing. New York: ACM, 2011: 70 – 71.
[4] BUI B, CHANG G, KIM R, *et al.* Demonstration of LMMP (Lunar Mapping and Modeling) using Amazon's elastic compute cloud [C]// SECLOUD'11: Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing. New York: ACM, 2011: 69 – 69.
[5] TRAN J J, CINQUINI L, MATTMANN C A, *et al.* Evaluating cloud computing in the NASA DESDynI ground data system [C]// SECLOUD'11: Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing. New York: ACM, 2011: 36 – 42.
[6] GREEN P J, SIBSON R. Computing dirichlet tessellafons in the plane [J]. The Computer Journal, 1978, 21(2): 168 – 173.
[7] LEWIS B A, ROBIRTSON J S. Triangulation of planar regions with applications [J]. The Computer Journal, 1978, 21(4): 324 – 332.
[8] AGGARWAL A, CHAZELLE B, GUIBAS L, *et al.* Parallel computational geometry[J]. Algorithmica, 1988, 3(1-4): 293 – 327.
[9] CHEN M-B, CHUANG T-R, WU J-J. Efficient parallel implementations of 2D Delaunay triangulation with high performance fortran [C]// PPSC'01: Proceedings of 10th SIAM Conference on Parallel Processing for Scientific Computing. Philadelphia: SIAM, 2001.
[10] OKUSANYA T, PERAIRE J. 3D parallel unstructured mesh generation [C]// Joint ASME/ASCE/SES Summer Meeting, Special Symposium on Trends in Unstructured Mesh Generation. [S. l.]: ASME, 1997: 109 – 115.
[11] CHRISOCHOIDES N, NAVE D. Simultaneous mesh generation and partitioning for Delaunay meshes [C]// Proceedings of the Eighth International Meshing Roundtable. South Lake Tahoe: [s. n.], 1999: 55 – 66.
[12] CHRISOCHOIDES N, NAVE D. Parallel Delaunay mesh generation kernel[J]. International Journal for Numerical Methods in Engineering, 2003, 58(2): 161 – 176.
[13] BLANDFORD D K, BLELLOCH G E, KADOW C. Engineering a compact parallel delaunay algorithm in 3D [C]// SCG '06: Proceedings of the Twenty-second Annual Symposium on Computational Geometry. New York: ASM, 2006: 292 – 300
[14] 罗斌. 基于约束数据域三角剖分的高精度 DEM 快速生成技术及实现[D]. 西安: 长安大学. 2006
[15] TAMMINEN M. Comment on quad- and octtrees[J]. Communications of the ACM, 1984, 27(3): 248 – 249.
[16] NGUYEN D, PINGALI K. Synthesizing concurrent schedulers for irregular algorithms [C]// ASPLOS'11: Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2011: 333 – 344.
[17] KULKARNI M, PINGALI K, WALTER B, *et al.* Optimistic parallelism requires abstractions [C]// PLDI '07: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2007: 211 – 222.

(上接第 2170 页)

4　结语

本文针对海量的语义 Web 服务的组合问题,提出了一种基于服务簇和 QoS 的组合方法。基于服务簇进行服务搜索,搜索空间小,语义比较的次数少,搜索速度快。基于服务的最优组合 QoS 值动态确定阈值进行服务过滤,可以获得多个最优组合。采用服务簇内部过滤和高效的冗余处理方法,可大量减少组合数量,保证组合中冗余服务最少,并且可在百万级海量服务库中保证执行速度。本文仅针对服务响应时间的 QoS 属性给出了最优组合算法,如何权衡多 QoS 属性获得最优组合,仍然需要进一步研究。

参考文献:

[1] 徐小良, 陈金奎, 吴优. 基于聚类优化的 Web 服务发现方法[J]. 计算机工程, 2011, 37(9): 68 – 70.
[2] NAYAK R, LEE B. Web service discovery with additional semantics and clustering [C]// ICWI'07: Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence. Washington, DC: IEEE Computer Society, 2007: 555 – 558.
[3] 孙萍, 蒋昌俊. 利用服务聚类优化面向过程模型的语义 Web 服务发现[J]. 计算机学报, 2008, 31(8): 1340 – 1353.
[4] 刘书雷, 刘云翔, 张帆, 等. 一种服务聚合中 QoS 全局最优服务动态选择算法[J]. 软件学报, 2007, 18(3): 646 – 656.
[5] KONA S, BANSAL A, GUPTA G. Automatic composition of semantic Web services [C]// ICWS'07: Proceedings of the 5th International Conference on Web Services. Washington, DC: IEEE Computer Society, 2007: 150 – 158.
[6] KWON J, LEE D. Non-redundant Web services composition based on a two-phase algorithm [J]. Data & Knowledge Engineering, 2012, 71(1): 69 – 91.
[7] NASERI M, TOWHIDI A. QoS-aware automatic composition of Web services using AI planners [C]// ICIW'07: Proceedings of Internet and Web Applications and Service. Washington, DC: IEEE Computer Society, 2007: 29 – 35
[8] SIRINA E, PARSIAB B, WU D, *et al.* HTN planning for Web service composition using SHOP2[J]. Journal of Web Semantics, 2004, 1(4): 377 – 396
[9] HUANG Z Q, WEI J, HU S L, *et al.* Effective pruning algorithm for QoS-aware service composition [C]// CEC'09: Proceedings of IEEE Conference on Commerce and Enterprise Computing. Washington, DC: IEEE Computer Society, 2009: 519 – 522
[10] WEI J, CHARLES Z, HUANG Z Q, *et al.* QSynth: A tool for QoS-aware automatic service composition [C]// ICWS'10: Proceedings of International Conference of Web Services. Washington, DC: IEEE Computer Society, 2010: 42 – 49
[11] 邓式阳, 杜玉越. 一种基于逻辑 Petri 网的 Web 服务簇模型[J]. 计算机应用, 2012, 32(8): 2328 – 2332.
[12] JIANG X, WU B, DU Y Y. Architecture of dynamic service composition using logic Petri nets [J]. Journal of Software Engineering & Applications, 2011, 4(10): 585 – 589.
[13] DU Y Y, JIANG C J, ZHOU M C. Modeling and analysis of real-time cooperative systems using Petri nets [J]. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 2007, 37(5): 643 – 654.
[14] MURATA T. Petri nets: properties, analysis and applications [J]. Proceedings of the IEEE, 1989, 77(4): 541 – 580.
[15] DU Y Y, QI L, ZHOU M C. A vector matching method for analyzing logic Petri nets [J]. Enterprise Information Systems, 2011, 5(4): 449 – 468.