分类号 TP391 学号 14043043

UDC 密级 公开

工程硕士学位论文

高性能影像数据瓦片化关键技术研究

硕士生姓名 刘世永

工 程 领 域 电子与通信工程

研 究 方 向 大数据处理与空间信息技术

指 导 教 师 李军 教授

国防科学技术大学研究生院

二〇一六 年 十二 月

Data Tiling Technology for Remote Sensing Image Based on High Performance Computing

Candidate: ShiYong Liu
Advisor: Prof. Jun Li

A dissertation
Submitted in partial fulfillment of the requirements
for the professional degree of Master of Engineering
in Electronic and Communication Engineering

Graduate School of National University of Defense Technology
Changsha, Hunan, P. R. China

December, 2016

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：高性能影像数据瓦片化关键技术研究

学位论文作者签名：[signature]　　日期：2016年10月28日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：高性能影像数据瓦片化关键技术研究

学位论文作者签名：[signature]　　日期：2016年10月28日

作者指导教师签名：[signature]　　日期：2016年10月28日

目　录

表 目 录

图 目 录

摘　要

随着卫星技术的日益发展，遥感影像的时间空间分辨率得到大规模提高，传统的瓦片化方法面对大规模影像数据速度缓慢，无法满足 GIS 应用对数据快速发布以及快速可视化的要求。当前并行与分布式计算的火热，因此如何利用并行技术实现数据快速瓦片化功能是现在高性能地理信息系统领域内大规模遥感影像快速可视化研究的一个热点问题。本文分别从单幅以及分幅影像出发，探讨如何通过高性能计算以及大数据技术，减少算法计算和 IO 耗时，加快影像数据瓦片化速度的方法。具体进行了以下内容的研究工作：

第一，解决单幅大规模影像的快速瓦片化问题。在单幅遥感影像数据量越来越大背景下，传统方法在性能上已经远远达不到应用需求，随着大数据技术以及高性能计算的发展如何利用这些多核以及大内存的优势来加快单幅大影像的瓦片化速度是一个急需解决的问题。针对所提出的问题，本文实现了一种混合并行的影像金字塔构建算法、影像数据快速切片算法，通过这两个并行算法实现了单幅大影像的影像金字塔以及瓦片金字塔的快速生成技术，算法稳定性高，当设置的进程并行数过少时，每个进程下会自动细分多个线程对进程的任务进行二次细分，不仅可以提升算法速度，而且可以避免单个进程因数据量过大导致类型溢出的问题。从实验结果来看，以 115.9*GB* 单幅大影像创建金字塔为例，本文算法最快可以达到 GDAL 的 1893.31 倍，ArcGIS 的 85.3 倍，MPI 算法的 10.28 倍。

第二，解决分幅影像数据集的快速瓦片化问题。分幅影像数据集与单幅大规模影像相比，其特点是影像数量多，但是其中的每幅影像数据量不大，并且影像和影像之间还会有重叠部分，因此需要对相邻影像进行拼接裁剪操作来处理相交区域中的无效值问题。因此针对这些有大量重叠区域的分幅数据集，如何快速取出相交区域，并快速拼合及瓦片化是一个难点。针对上述问题本文实现了影像数据并行分幅输出算法、并分别提出了基于 MPI 实现以及 Spark 分布式实现的分幅影像快速瓦片化技术。算法与传统方法和技术相比，实现了流程的完全自动化，并且支持断点恢复功能，避免了断电等突发情况产生的影响。算法经过湖南省测绘部门的实际应用与测试，测试数据量最大达到 8*TB*，实验表明本文分幅算法可以将传统工作流程中 3 个月的工作时间缩短至短短的 3 天，算法在应急响应，数据快速发布以及国土部门数据生产中具有很好应用场景。

关键词: **高性能计算；大数据处理；地理信息系统；瓦片化；遥感影像**

ABSTRACT

With the rapid development of network technology, traditional client/server (C/S) architecture of the desktop version of GIS applications have been gradually to the use of browser/server (B/S) architecture WebGIS. B/S architecture because of its free installation, simple operation, suitable for processing large data, B/S mapping model becomes more and more popular, and gradually become the mainstream. At the same time, with the development of satellite technology, the time and spatial resolution of remote sensing images have been substantial increased. Traditional large-scale image data tiling process is slow, can not meet the rapid data release requirements of WebGIS application. Therefore, it is significant to realize the fast tiling technology of large-scaled image in the WebGIS.

Based on the above problems, in order to speed up the image data tiling speed, this paper discusses the method that how to reduce the computation of the algorithm and the IO time consuming by the high performance computing technology. Specific research work is as follows:

First, to solve the problem of large single image data Rapid tiling, this paper propose the parallel image pyramid building algorithm and the image data rapid slicing algorithm. Through the two parallel algorithm we achieve the image pyramid and the pyramid of tiles Quick Generation technology in single large image.

Second, solving the problem of fast image dataset tiling. The fast image dataset tiling technology has a very wide range of requirements for the Department of Surveying and mapping, the map area may be covered by several maps, so how to quickly get the intersection, fast split and tiling it is a difficulty. This paper realizes a fast tile image technology. The experimental results show that the proposed method can greatly improve the efficiency of land surveying and mapping departments.

Key Words: HPC; Big Data; GIS; Tile; Remote Sensing Image

符号使用说明

HPC	高性能计算 (High Performance Computing)
C/S	客户端 / 服务器（Client/Server）
B/S	浏览器 / 服务器（Browser/Server）
GIS	地理信息系统（Geographic Information System）
WebGIS	网络地理信息系统（Web Geographic Information System）
GeoTIFF	地理标签图像文件格式（Geographic Tag Image File Format）
IFH	金字塔文件头（Image File Head）
IFD	金字塔图像文件目录（Image File Directory）
DE	金字塔标签（Directory Entry）
MPI	消息传递接口（Message Passing Interface）
OpenMP	共享存储并行编程（Open Multi-processing）
TMS	瓦片地图服务（Tile Map Service）
GDAL	开源栅格空间数据转换库（Geospatial Data Abstraction Library）
ArcGIS	ESRI 公司出品的一个地理信息系统系列软件的总称
BIP	波段按行交叉存储（Band Interleaved by Pixel Format）
BSQ	波段顺序格式存储（Band Sequential Format）
MBR	最小外包框（Minimum Box Region）
ParaOvr	本文并行影像金字塔构建算法名称
paraMosaic	本文并行镶嵌算法名称
Hadoop	海杜普 –Apache 基金会所开发的分布式系统基础架构
HDFS	一种分布式存储系统（Hadoop Distributed File System）
Storm	Twitter 公司开源的一个分布式、容错的实时计算系统
Spark	加州伯克利分校 AMP 实验室开源的类 Hadoop 通用并行框架
RDD	弹性分布式数据集 –Spark 中数据和计算的基本抽象
Geotrellis	Azavea 公司开源的分布式地理空间数据处理框架

第一章　绪论

1.1　本文的选题背景和研究意义

随着空天技术的发展与深入，人类在地理空间信息领域不断地前进，现有的探知技术与实现手段也越来越先进，现在人类通过使用各种技术和工具来观察记录这个世界正在发生的事情，以期望分析这些数据来得到有价值的信息，这些观测数据包括有各种类型卫星的对地观测数据，天文望远镜获取到的外太空影像，物探化探的监测数据等。这些实测数据正在源源不断地更新和记录着，并且随着传感器精度的不断提高，这些数据所对应的数据量正在几何倍数的方式增大[1]。在网络带宽受限的情况下这些大量的数据要发布到 Internet 上进行显示和分析，目前使用瓦片化技术是业内普遍采用的方法[2]，这种方式是通过预先将影像切成一张张瓦片，当影像加载的时候，通过计算当前浏览器可视域范围，只显示在可视域范围内的瓦片，这种方式可以极大地减少数据的加载量，提高影像的加载效率[3]。数据瓦片化的速度直接决定了影像可以在 web 上进行发布和展示的时间，因此快速瓦片化技术对影像数据发布有强烈时效性要求的领域例如应急响应，军事侦查，地图制图等都有着急切的需求。

目前的大规模影像数据其大规模主要体现在两方面，第一、单幅影像的数据量非常大，随着遥感影像的分辨率不断提高，越高的分辨率意味着需要更多的空间进行存储，因此覆盖相应区域的影像数据量就会更加的大；以 GF-2 号卫星为例，GF-2 空间分辨率为 1.0m，一幅覆盖全国区域范围的 GF-2 卫星影像其数据量就达到 300GB 左右，而如果是分辨率更高的无人机航拍影像其数据量将会更大，以湖南省一个普通县城为例，无人机影像分辨率 0.2m，一幅覆盖全县的遥感影像数据量就会达到 100GB 以上。第二、影像的数量非常多，现阶段遥感影像获取的途径非常多，从平台上进行划分，大到太空中的卫星，小到街边的摄像探头，都能够成为影像的获取来源，并且同一平台可能又会搭载多个不同的传感器，因此相应区域其可能由大量单幅数据量不是那么大、但是种类不同的影像数据集组成；以湖南一个普通县城为例，其标准分幅数据集中每幅影像大致 80MB 左右，影像总数量可以达到 1000 多幅以上。无论是对单幅大规模影像进行瓦片化还是对大量小影像的数据集进行瓦片化，传统方法以及技术在性能上往往力不从心，无法满足应急响应等数据在 WebGIS 上快速发布及可视化的需求。

瓦片化技术是当前 WebGIS 数据可视化不可缺少的方法，得益于开放地理空间信息联盟（OGC）制定的网络地图服务的标准以及 Internet 的爆炸式发展，WebGIS 如雨后春笋般蓬勃发展起来，WebGIS 服务正在走进每个人的日常生活[4]。

目前从事相关技术的组织可以大致可分为两个方向：一个是互联网方向，一个是行业应用方向[5]。互联网方向有百度公司的 Baidu Map，腾讯公司的搜搜地图，被阿里收购的高德地图以及 Google 公司的 Google Maps 等；在行业应用方向国外的有 ArcGIS 的 ArcGIS Server，Microsoft 公司的 Terra Server 影像数据服务器等，国内有超图公司的 SuperMap IS、武汉吉奥有限公司开发的 GeoSurf 等[6]。这些公司特别是互联网方向的企业，其每天都有很多新的数据需要发布，需要对新数据进行切片，替换老旧的瓦片，他们对数据瓦片化的速度要求非常高，因为当数据量非常大时如果瓦片化的速度跟不上，可能当前数据还没有切完瓦片，新数据又来了，因此这更对影像数据快速瓦片技术提出了迫切的需求。

当前随着硬件性能的大幅提升以及并行与分布式内存技术的不断发展，例如最近十分火热的 Spark 以及 Hadoop 分布式计算平台，因此如何将高性能并行技术与瓦片化技术结合，利用现在最新的高性能计算以及大数据技术来提高影像瓦片化的速度，这在 WebGIS 领域中是一个非常有意义的研究方向。本文紧贴生产实践部门从应用需求角度出发，针对单幅大规模影像以及分幅影像数据集分别进行研究与探讨，利用大的计算资源来细分任务并行执行，寻求在多种数据类型以及应用场景下的快速瓦片化实现方案，提出了高性能影像数据瓦片化关键技术的实现方法。

1.2 影像数据瓦片化技术研究现状分析

1.2.1 国外研究现状

影像数据瓦片化技术无论是在工业界还是学术界都进行了非常广泛的研究，国外许多全球知名公司以及学者都在瓦片化技术方面申请了大量的专利，比如早在 1991 年 7 月由日本三菱电机株式会社申请了一项图像数据归档系统，其描述的主要实现方法其实就是基于瓦片化技术，其先将原始影像转换成多个不同尺度数据，而后将上述不同缩放图像切割成每个大小相同的块，并将块存储在存储器的相应区域[7]。1997 年 3 月哈里斯公司的 Iodice et al. 申请了一项远程图像显示方法的专利，通过将原始影像建立多分辨率数据集，而后分析出影像中用户感兴趣的部分，通过只传输感兴趣区域的瓦片来达到降低带宽，加快显示的作用[8]。在 1997 年美国微软公司最早在 1998 年 5 月提出了一项基于瓦片化技术实现的纹理图像快速高效显示系统的专利，其方法是基于原始纹理图像生成多层不同分辨率的纹理瓦片数据，而后基于视角的远近对各层纹理瓦片数据进行映射，当进行显示的时候，直接调取分辨率最相似的那层瓦片进行纹理贴图即可，这已经有了瓦片金子的雏形[9]。随后在 1999 年 1 月来自日本的 Kaneda 申请了一种图像压缩存储的设备与方法，这个方法就是将原始影像进行瓦片化，然后再将瓦片压缩存储

在设备的不同的区域，以达到对影像的高效存储与检索[10]。在 2000 年微软公司的 Barlay 在 SIGMOD 提出了一个名为 TerraServer 的空间数据仓库，阐述了如何通过 Internet 将影像数据瓦片以及元数据存储在 SQL 数据仓库中[11]。比较有革命性的是 Google 公司的 Jens et al. 在 2005 年 2 月申请的一项名为数字测图系统的专利，推出了地图瓦片以及瓦片坐标系的概念，其基本理念是将地图预先按照缩放级别切割成固定大小不同层级的栅格图片，而后根据相应的坐标系统，并按照一定的命名以及存储规则为每个栅格图像块构建索引目录，这里每个图像块就被称作瓦片（Tile），当客户端要进行显示的时候，客户端基于当前可视范围的地理范围，计算出所需要瓦片的索引坐标，只需向服务端发送相应瓦片的位置请求，并将接受到的瓦片数据集在客户端进行可视化即可，这样不需要实时渲染和切图，速度大大加快。再加上浏览器本身的本地缓存技术，使地图影像浏览速度大大提高，这套技术被最先用到了谷歌自己的产品 Google Map 上[12]。

而后很多公司在 Google 提出的地图瓦片的基础上，提出了自己的改进方法与相关应用，如兰德麦克纳利公司的 McAvoy et al. 在 2007 年 9 月公开的一项定制挂图印刷系统，通过先将数字地图瓦片化，然后用户可以选择合适的分辨率以及区域进行地图印刷，既能保证打印信息的详细程度，又能确保印制的地图易于阅读[13]；美国 Adobe 系统公司的 Brichford. 在 2009 年 11 月公开了一项超文本语言渲染系统，可以实现在 web 浏览器用超文本语言渲染瓦片数据[14]；以及微软公司的 Jing et al. 在 2010 年之内接连公布三项基于瓦片金字塔实现的专利，分别为图像集群查看接口专利，可视区域图像查看接口专利以及视线位置确定专利[15–17]；随着技术的发展瓦片化技术也已经成功应用到三维领域，如 Lafon. 在 2010 提出一项利用瓦片渲染全景图的技术，通过映射第一个可视区域到第二个可视区域，然后再计算得到所需加载的瓦片图像，实现全景图像的有效呈现[18]；还有 Zhu et al. 在 2011 年提出的了一种在街景图像的瓦片数据上进行三维注解[19]。瓦片化技术应用是如此之广，广大的公司和个人都在基于这项技术实现了非常多的应用，一直到现在都有非常多新技术被提出，如最早提出地图瓦片概念的谷歌公司在 2016 年又基于此陆续申请了两项专利，如用户间共享地理信息技术以及地理信息系统中动态可视图层技术[20, 21]。

不仅工业界中有大量有关瓦片化的成果，学术界亦对此进行了很多研究，发表了很多文章。Surazakov 在 2006 年提出利用 SRTM 以及基于地图的地形数据来估计高山和冰川的变化，在方法中他将地形数据按照经纬度 1° × 1° 进行瓦片划分，而后针对每个瓦片进行分析，最后将所有分析结果合并[22]。Thorvaldsdottir et al. 在 2013 年提出一种综合基因组查看器（IGV），利用瓦片金字塔技术实现高性能基因组数据的可视化[23]。Maiellaro et al. 在 2016 年利用瓦片化方法实现了一种

基于互联网的文化遗产（CH）应用，通过多媒体互动地图与各种可视化方法，使得用户可以快速找到自己需要的结果[24]。Iriberri Gutierrez 在 2015 年提出一种基于地理可视化的四叉树缓存生成的预测分析，通过预测 web 地图可视化中的使用模式来动态生成可以被预缓存的数据，最终达到改善地图可用性的目的，在文章中其定义了一种瓦片缓存预测模型，而且重新规定了瓦片生成的时序细节[25]。Follum et al. 提出一种大区域范围洪水淹没估计模型，同样是通过对洪水淹没图按照经纬度 1° × 1° 进行瓦片划分，然后并行对每一个瓦片进行模拟和分析[26]。Lee et al. 在 2015 年提出了一种应用于移动端 GIS 的瓦片搜索与索引管理算法来优化地图信息，其中包括大规模信息和连续的变化信息的聚集以及快速处理[27]。Langevin et al. 等基于瓦片视觉分析技术提出了一种全球人口生活模式以及地理时间事件检测模型（TBVA），通过对互联网中数十亿条记录的交互式多尺度分析，检测出了人口的运动模式，以及一些异常事件[28]。Behm et al. 提出一种用于地理查询的高效地图检索与排名方法，通过四叉树的瓦片结构，其方法可以高效和精确地从一个指定区域的大量候选地图中检索到自己需要的那幅数字地图[29]。Liu et al. 在 2013 年提出一种交互式大数据可视化技术（imMens），通过数据降采样方法来实现大量数据类型的可视化，其中结合了多源数据瓦片以及并行查询处理技术来实现在分级图间的交互式查询[30]。

2006 年，OpenStreetMap 基金会应运而生。它资助的项目为公众免费地提供了全球详细到城市街道的地图数据和遥感影像数据，扫清了开发面向公众的地理信息系统的障碍[31]。OpenStreetMap 推动了地理信息应用向 Web，向云端发展，海量数据的快速更新要求因此也对快速瓦片化提出了迫切的需求。OpenStreetMap 发布了在线和离线两个版本，离线版本比在线版本提供的更加丰富的功能，这在某种程度上反映了现在 Web 页面制图存在的问题：在数据量逐渐增多之后，远端服务器切片速度将完全跟不上，网络资源和服务器资源的限制，会导致地图瓦片生成有较大的延迟等问题。因此，加速服务器地图瓦片切片效率是一个值得深入探讨的问题。

现阶段应用最广的瓦片切片工具，包括全球知名的 ESRI 公司的 ArcGIS Server 软件、Mapnik 工具包以及 GDAL 类库提供的切片工具等等。ArcGIS Server 是目前国土测绘部门广泛使用的切片工具，它以其优异的空间分析功能以及良好的稳定性，占据了全球 GIS 应用软件的半壁江山；ArcGIS Server 瓦片化工具支持在单机上多线程并行切片，但是其并行程度不高，通过观察 CPU 使用率，可以发现每个核的 CPU 资源占用率都不是很高，并且其不支持分布式，也不能跨系统，不能跨节点利用集群资源，性能可扩展性较差。Mapnik 是一个用于 GIS 应用开发的程序包，其核心是一个 C++ 的共享库，通过这个库来提供空间数据访问以及相关可

视化算法和模式，其支持在多线程环境下运行，支持多种操作系统，通过一定的调度算法，可以实现其在多个节点上并行切片，但是这种方法配置复杂，对专业背景要求很高，一般技术人员很难上手。GDAL 核心同样也是一个 C++ 共享库，通过链接库以及 SWIG 工具实现 JAVA、C#、PYTHON 等其他语言的接口，在其 PYTHON 算法包里有一个 gdal2tiles 脚本提供切片支持，但是其不支持并行，当数据量很大的时性能完全不能满足需求。

在国外，地图瓦片生成的技术和理论都得到了充分的研究，由于现有方法和工具在处理大规模数据瓦片化上的局限性，一些学者尝试使用并行的技术，来加速瓦片化的过程，如 Bielecki et al. 在 2016 年提出一种基于传递闭包图的并行瓦片合并技术来进行瓦片编码[32]。Guevara et al. 在 2016 提出一种基于并行硬件的嵌入式小波图像重建算法，在方法中其通过在 180° 均匀间隔内 RT（Radon Transform）变换获得投影数据，在正弦图中确定每一个相关区块，然后对每个瓦片进行并行重建[33]。Schwambach et al. 提出一种用于非线性约束嵌入式应用的影像瓦片化方法，利用约束编程实现了影像瓦片大小的最优化，采用并行调度的方式实现瓦片化的加速[34]。虽然在高性能瓦片化方面已经有些进展，但是，高性能地图影像切片技术的实现，仍然是现在高性能地理信息系统领域研究的一个热点和难点，并且目前的商业地理信息系统以及相关开源软件及算法中，还没有有公开提供可靠的高性能并行瓦片构建的功能。

1.2.2 国内研究现状

相比于国外，国内在瓦片化算法方面也进行了非常多的研究，曹冬冬等人提出一种基于瓦片化算法的并行 QR 分解及其实现，通过对原始矩阵分块，每个块内的数据连续存储，而后各个瓦片独立进行分解，其通过 MPI/OpenMP 混合并行技术实现了瓦片分解的并行化[35]。陈欢等人提出一种地理矢量数据快速可视化技术，通过利用集群中的多节点多核计算资源实现了单个瓦片的多线程并行绘制功能[36]。刘晰等人提出了一种并行海量数据瓦片化方法，其针对现有将海量 4D 数据转换成瓦片金子塔方法的不足，利用对地理空间区域进行任务划分来实现影像的并行切片[37]。周郑芳等人基于瓦片模型以及分布式并行存储系统，设计实现了一个遥感影像并行瓦片服务，用于快速存储以及检索以瓦片方式组织的遥感影像[38]。杨子煜等人利用瓦片分割思想提出一种基于多核阵列体系结构的嵌套循环并行优化算法，实现了减少串行代码转换成并行代码中嵌套循环所占的时间[39]。刘义等基于 MapReduce 框架，提出了一种基于分辨率和空间范围自动匹配的瓦片金字塔索引生成方法，分别详细描述了 Map 和 Reduce 阶段瓦片并行生成以及并行合并的处理方法[40]。陈小潘等人采用双缓冲队列的思想，提出一种海量地形数

据并行处理方法来加快大规模地形数据的绘制速度，其通过将地形数据的处理过程和渲染过程分别独立并行执行，并根据瓦片加载优先级进行任务分配，来最终达到给算法提速的目的[41]。殷军茹等人针对分布式环境下海量空间数据的快速可视化问题提出一种基于数据库存储方案的瓦片数据快速分发方法来实现多用并发访问时的快速响应[42]。邓雪清等人提出一种针对于地形服务器系统的瓦片金字塔模型以及线性四叉树瓦片索引方法，并探讨了相应的数据分布方法[43]。杜波等人提出一种基于 MapReduce 的栅格影像并行切片系统，设计了一种利用瓦片缓存技术的瓦片金字塔构建方法，通过将格式块文件分配到多个节点并行切片，实现了瓦片金字塔的并行构建[44]。张锦林等基于实时流计算框架 Storm 实现了一种海量遥感影像瓦片金字塔并行构建方法，并且基于 HBase 设计了瓦片金字塔的存储模型，通过将瓦片编码转化为 HBase 数据库表的键值，而后将瓦片数据存入 HBase 中[45]。刘坡等人提出一种大规模遥感影像全球金字塔并行构建算法，利用 GPU 来加速重采样的计算，通过多线程来提高数据的 IO 速度，并利用四叉树策略提高显存中数据的重复利用率[46]。刘振东提出了一种大规模三维地形高效可视化方法，在方法中其提出一种多线程并行切片技术，而且实现了相关的快速调度算法，以及众多小瓦片文件压缩成瓦片大文件的方法[47]。原发杰提出了一种数据库与文件系统接合的瓦片数据存储管理系统，其采用分布式文件系统存储瓦片影像实体，瓦片索引以及元数据存储到各节点下的数据库文件中，同时实现了影像数据与元数据的分布式存储[48]。李晶对三维数字地球构建技术进行了研究，在其实现过程中通过利用瓦片化技术对全球高程数据进行切片，实现了瓦片内多分辨率层次的构造，并对地形瓦片不同分辨率间的裂隙进行了修补，解决相邻分辨率间的突跳问题[49]。

国内学术界对瓦片化理论与应用进行了很多研究，并且对其并行化实现方法也进行了很多探索。同时国内工业界基于 WebGIS 的影像数据瓦片化服务也进行了大量相关的实践探索，主要是以天地图、百度地图和高德地图为代表的互联网电子地图以及 MapGIS、SuperMap、Geostar 等为首的国内 GIS 应用服务商，目前高性能地理计算平台在这方面走在了全国技术的前列，是一个非常有应用前进的研究方向。

在国内，影像数据瓦片化的并行方法也是地图学研究的难点和热点。学者们从多种理论和技术手段出发，寻找高性能影像数据瓦片化的实现方法，目前也有了一些非常好的成果，这些技术都推动了国内地图数据瓦片化的进展，但目前仍还没有非常适用于工程应用并且高性能并行的可靠方法。本文主要从影像数据瓦片化并行构建出发，吸收前人的研究成果，通过实验和测试，进而提出并阐述了一些高性能影像数据瓦片化关键技术的相关实现方法。

1.3 本文的研究内容和组织结构

1.3.1 文章研究内容

当前，大数据时代，空间数据正爆炸似地增长。由于传感器精度的的大幅地提高，高分辨率遥感卫星被大量地发射上天，遥感影像文件的数据量也急剧增加[50, 51]。同时，随着云服务云计算的迅速发展，基于 B/S 架构的 WebGIS 已经成为 GIS 未来的前进方向。因此在网络环境下实现大规模影像数据的快速可视化成为了研究的一个热点问题，在此背景下实现高性能影像数据瓦片化相关技术就成为重中之重，这也是本文的重点研究内容。具体包括以下三个内容：

第一，影像并行构建金字塔

借鉴传统的影像金字塔模型的思想，提出了一种并行构建影像金字塔的机制。金字塔是一种栅格数据的多分辨率组织结构，简单来说，金字塔结构就是由原始栅格影像开始，建立起一系列不同分辨率的栅格影像，不同分辨率的栅格影像对应不同的金字塔级，影像金字塔可用于用于生成瓦片金字塔，以及 MapNik 工具等动态切图需求。本文提出的算法通过深入研究金字塔文件的数据结构，提出一种高效稳定的金字塔文件并行访问技术，即在创建空金字塔文件时预先对金字塔文件结构进行规划，达到每个进程能够对金字塔文件进行精确访问，打破了每个进程都必须按固定条带大小写的限制，同时也彻底解决了黑边现象以及零碎 IO 的问题，大幅提高了并行 IO 的速度，使影像金字塔的构建效率得到进一步大幅提高。

第二，影像数据并行切分合并技术

本文中影像数据并行切分技术主要研究了三方面 1、影像快速切片技术 2、基于接图表的批量快速分幅输出技术 3、批量地图瓦片并行融合技术。影像快速切片技术实现了对单幅影像并行切片，这也是影像数据瓦片化技术的核心；基于接图表的批量快速分幅输出技术实现了批量对影像进行分幅输出，这部分主要是为了满足国土测绘部门的需求；批量地图瓦片并行融合技术实现了相同行列号瓦片的融合，主要是为了处理瓦片中的无效值区域。

第三，影像数据并行镶嵌拼接技术

本文中影像数据并行镶嵌拼接技术研究主要分两方面 1、影像数据的并行镶嵌技术 2、地图瓦片并行合成技术。影像镶嵌技术在地图数据处理中主要是应对工程实践中需要先把一些小影像进行拼接形成一个大影像消除相邻影像重叠区域的无效值后再进行切片的情况；地图瓦片并行合成技术主要是通过某一层级的瓦片来快速合成其底层的瓦片，这种方法可以不需要对原影像进行操作，提高了瓦片切片的效率。

1.3.2 文章组织结构

本文共有五章。文章结构如下图所示：

第一章，绪论。阐述论文的研究背景和研究意义。总结当前 WebGIS 可视化的发展现状，影像数据瓦片化概念，现有的瓦片化技术方案。介绍文章的研究内容以及解决问题的思路。

第二章，介绍瓦片金字塔模型。根据开放地理联盟（OGC）提出的瓦片地图服务标准（TMS），阐述当前常见的一些瓦片组织方式，包括瓦片的切分方法、存储方式、索引方式。TMS 为瓦片切片提供了准则，基于 TMS 准则切出来的瓦片具有通用性移植性。

第三章，介绍单幅大影像快速瓦片化方法。根据大影像数据量大的特点通过构建影像金字塔的方法实现实时生成可视范围内瓦片，采用多进程并行的方式加快切片的速度。

第四章，介绍分幅影像数据集瓦片化方法。根据国土测绘部门的需要，根据接图表实现影像数据的快速分幅输出，阐述分幅输出的核心方法，以及而后如何对大量分幅输出数据集进行瓦片化操作。

第五章，总结全文研究内容以及取得的成果，指出目前研究过程中存在的问题，以及未来的展望。

第二章　瓦片金字塔模型

本章主要介绍瓦片金字塔模型的原理、作用以及当前主流商业应用的瓦片切分方法。瓦片金字塔模型是当前应用最广的多层次地图数据组织模型，通过瓦片金字塔模型，前端在进行放大和缩小操作时，可以有效地减少数据读取的空间查询时间。通过只加载可视区域范围内的瓦片，可以减少数据加载量，降低网络传输压力，提高前端的数据可视化速度。目前关于瓦片金字塔模型的相关研究也有一些，如徐虎等设计与实现了基于中间件的瓦片地图服务[52]，霍亮等对瓦片金字塔模型技术进行了研究与应用[53]，等等。

2.1　瓦片金字塔概念

瓦片最早是由谷歌地图提出并进行应用的，其采用特定的切割方式对采用WebMercator 投影坐标的世界地图栅格影像进行切片，由于 WebMercator 投影方便计算机进行计算，随后各大主流 WebGIS 和互联网地图应用商都采用基于WebMercator 投影坐标系的方式进行切片，例如国内的有百度地图、高德地图等，国外有 Google Map、Bing Map 等[54]。如图 2.1，瓦片金字塔主要原理为：基于某个特定的地图投影坐标系，将曲面的地球投影到二维平面，而后将该二维平面进

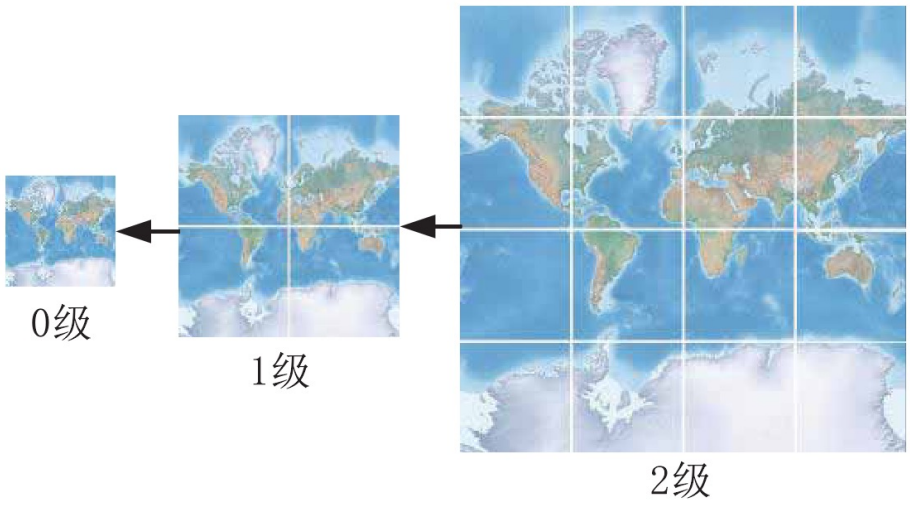


图 2.1　瓦片金字塔模型

行多尺度地划分，即相当于制作了多个不同分辨率层级的数字地图。各层级对应相应编码，层级越高地图所对应的分辨率越高；而后对每一层级的全球空间范围地图按照某种空间划分方法进行格网划分，划分成若干行和列的固定尺寸的正方形栅格图片，这些切分出来规整的单个格网单元称为瓦片，各层及的划分方法都是相同的。瓦片划分方法需满足以下条件：（1）每个层级下的所有瓦片可以无缝

拼合成一张全球空间范围的世界地图。(2) 每个瓦片有唯一编码，根据编码可以解算该瓦片对应空间范围。(3) 在某一层级下给定一坐标点可以根据其空间坐标解算其所在瓦片的编号。每一层级瓦片对应一层金字塔，各个层级的瓦片构成了整个瓦片金字塔模型。

2.2 瓦片坐标系

所有瓦片的编码都是基于瓦片坐标系下进行的，瓦片坐标系的原点一般在左上角或者左下角，TMS 规范中是在左下角，但是现有的 Google、MapNIK 切片系统都是选用左上角点作为原点，本文所描述的瓦片坐标系原点都是基于左上角点。见图 2.2，瓦片的编码方式采用以下方法，层级用 z 表示，瓦片经线方向上编号为 x，纬线方向上编号为 y，因此每一个瓦片都可以通过一个三维元祖（x,y,z）来唯一描述。

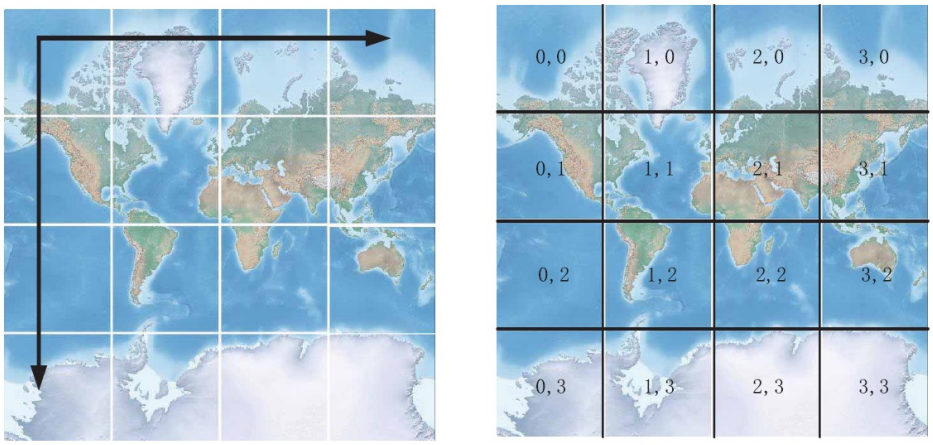


图 2.2 瓦片坐标系

瓦片在文件系统的中的存储结构就是基于上述编码方式的三级目录存储结构，其中第一级为瓦片级别 z，第二级为瓦片列号 x，第三级为瓦片行号 y。以路径“root/z/x/y.png”作为瓦片文件的存储路径，其中 root 为瓦片输出根目录，因此只需要正确解析瓦片的编码就可以获取地图瓦片相应的访问路径。

2.3 瓦片投影坐标系

瓦片金字塔模型中的投影坐标系可以有多种，但从计算机实现难易的角度出发，目前最广泛采用的是 WebMercator 投影，它是 Mercator 投影的一种变种[?]。见图 2.3为了介绍 WebMercator 投影，首先必须先对 Mercator 投影进行说明：Mercator 投影中文名为墨卡托投影，是正轴等角圆柱投影，由荷兰地图学家墨卡托于 1569 年创立，其是预先假定地球被套入一个圆柱体中，赤道与圆柱的内侧相切，而后在

地球的中心放入一盏灯，光线会把地球球面投影到圆柱体上，然后再展开圆柱体，这就形成了一幅墨卡托投影的世界地图，其原点在经纬度 (0,0) 处即赤道与中央经线的交点[55]；这种投影方式可以保证赤道处无变形，并且保持了地图上方向和角度的正确性，如果循着墨卡托投影图上两点间的直线航行，方向不变可以一直到达目的地，因此基于上述优点被广泛应用于航海图和航空图；但是这有一个问题由于理论上南北极是永远无法投影到圆柱体上，并且随着纬度的增高其变形越大，为了解决这个问题使 Mercator 投影更加适合于 web 浏览器上，web 墨卡托投影忽略了墨卡托投影中南北两级纬度大于 85.051° 的变形较大区域，把椭圆形的地球投影成平面上边长等于赤道周长的正方形[56]。因此可以得知以经纬度表示其大地坐标范围为 [-180°,-85.0511287798°,180°,85.0511287798°]，相对应的投影坐标范围为 [-20037508.3427892*m*,-20037508.3427892*m*,20037508.3427892*m*,20037508.3427892*m*]，数值 20037508.3427892*m* 为赤道周长的一半。

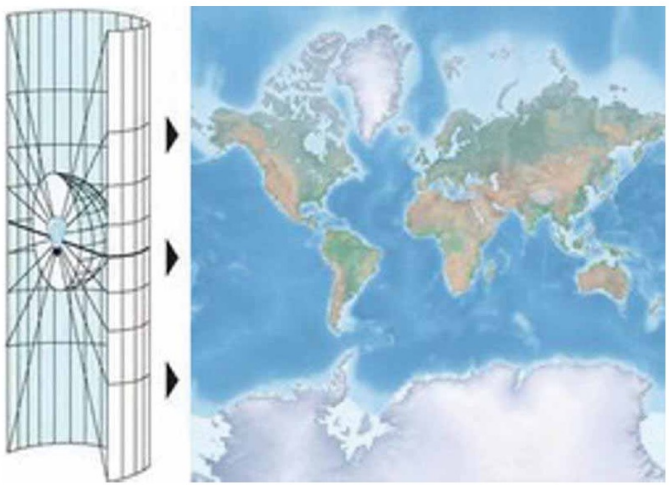


图 2.3　墨卡托投影示意

每一层中的瓦片划分方法一般采用均匀四分的划分方法，即以赤道和中央经线的交点作为中心，不断对地图进行四分，直到每个格网大小为 *tilesize*×*tilesize* 为止，其中 *tilesize* 表示单个瓦片的边长[55]。如图 2.1所示，基于上述的划分方法，第 0 层金字塔用一个瓦片就能表示整张世界地图，第 1 层金字塔要用 4 张瓦片，2 层 16 张，以此类推，每层金字塔要用 4^z 个瓦片来表示整个世界地图，z 为当前瓦片金子塔层级。

2.4　瓦片金字塔模型意义

当前瓦片金子塔模型被广泛应用于众多互联网地图服务中，例如国外的 Google Map、Apple Map、Bing Map 等等，国内有搜搜地图、百度地图、高德地图等等。瓦片金字塔模型在实际应用中具有以下诸多优势：(1) 瓦片被一次绘制后，

即被当做缓存数据保存在服务器中，后续如果有对相应瓦片的访问请求则无需重新绘制，不仅可降低绘制引擎的压力而且能够提高响应速度。(2) 瓦片之间不存在依赖性，可以利用高性能或分布式集群平台进行高效快速切分和处理。(3) 现有的瓦片金字塔模型大多满足 TMS 标准中制定的接口和协议，相互之间具有一定的通用性和可移植性[57]。

现如今瓦片金字塔已经成为 WebGIS 数据可视化的一个标准模型，因此实现影像快速切分并构建瓦片金字塔模型已经成为 WebGIS 数据快速发布的一个迫切需求。

2.5 瓦片金字塔构建方法

针对单幅影像以及分幅影像的瓦片金字塔构建方法，本文分别进行了相关研究与探索，单幅大规模影像瓦片金字塔构建方法与分幅影像瓦片金字塔构建方法既有相关又互相区别，相关流程图如 2.4所示：

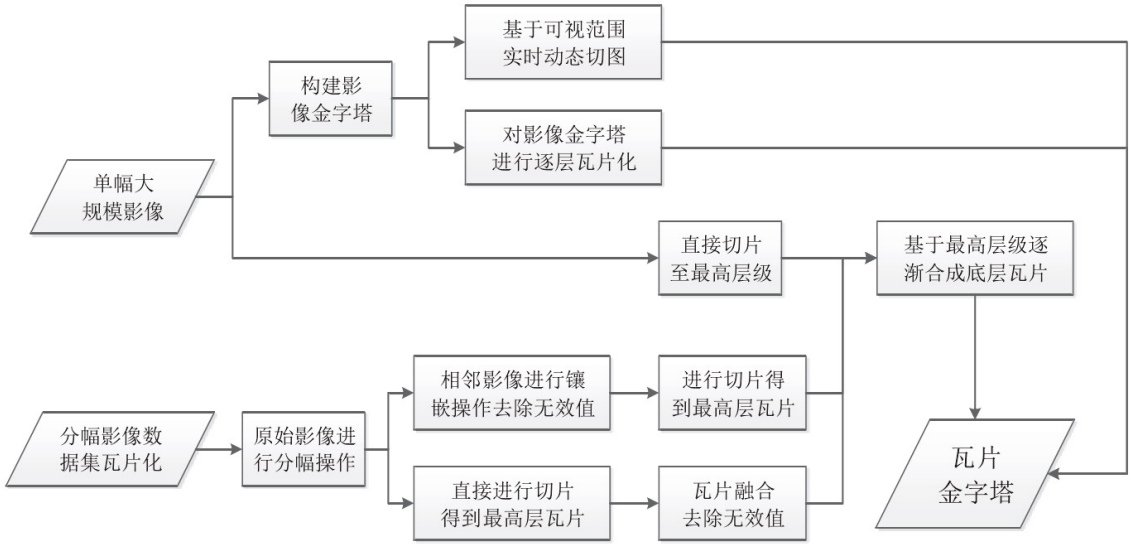


图 2.4 瓦片金字塔构建流程

基于图 2.4所示，单幅大规模影像构建瓦片金字塔本文主要描述了以下两种实现方法：

第一种、*构建影像金字塔法*，先对单幅影像构建影像金字塔，而后可以利用 Mapnik 基于游览器的可视区域及当前地图的缩放层级对影像金字塔文件实时进行切片，形成瓦片金字塔；还有一种就是直接基于影像金字塔的各层级，逐层进行瓦片化，最终形成瓦片金字塔。Mapnik 主要用于实时处理，简单来说就是用户在浏览器前端访问地图，看到哪里，我就切到哪里，当然其形成的瓦片金字塔并不是完整的；而后一种直接基于影像金字塔文件逐层切片，这种方法主要用于预处

理，直接通过影像金字塔生成完整的瓦片金字塔数据。

第二种、瓦片合成法，直接基于原始影像的分辨率切片至其最高层级瓦片，而后基于其最高层级瓦片，采用相邻四张瓦片长宽各 $\frac{1}{2}$ 降采样，进行拼接的方式合成下一层瓦片，逐层进行合成，直至最后拼接成只剩一张瓦片为止，至此瓦片金字塔构建完毕。

同样本文分幅影像数据集的瓦片金字塔构建方法主要也有两种：

第一种、拼接裁剪的方式，首先对原始影像数据集内的数据进行分幅操作得到标准分幅影像，而后将相邻影像进行拼接裁剪去除无效值的影响，得到一个完整区域的大图，而后对这个大图进行切片形成最高层级瓦片，最后采用上述所说的瓦片合成的方法，生成瓦片金字塔。

第二种、瓦片融合的方式、首先跟第一种方法一样，对原始影像数据集进行分幅，而后不同的是不进行拼接裁剪操作直接对分幅影像进行切片，最后通过瓦片融合的方式去除无效值得到最高层瓦片，而后与上面类似，采用瓦片合成的方式生成瓦片金字塔。

2.6 本章小结

本章主要阐述了瓦片金字塔模型的相关原理，包括瓦片金字塔的概念、瓦片坐标系、瓦片投影坐标系、瓦片坐标系在 WebGIS 中的应用以及本文所提出的瓦片金字塔构建方法等。瓦片金字塔实际上就是一个由多个不同分辨率影像组成的数字地图，用于浏览器在执行放大、缩小等操作是能够实现无缝浏览。瓦片坐标系就是瓦片自身的坐标系，这个坐标系下每个瓦片有一个唯一的行列号编码，基于地理位置可以直接解算得到其对应瓦片的行列号，并且这个坐标系也直接决定了瓦片的存储结构。瓦片投影坐标系指的是瓦片的投影方式，瓦片是基于这个投影系下进行切片的，目前最常用的瓦片投影坐标系为 WebMercator 投影坐标系。

第三章　单幅大影像快速瓦片化方法

本章介绍单幅大影像快速瓦片化的方法，单幅大影像由于其一幅影像所包含的数据量非常大的特点，传统串行方法在应对这么大的数据量时，IO 耗时将会非常严重，由于同一幅影像内的数据其关联性大，并行难度较高，因此目前研究并行处理单幅大数据量影像的并不是很多，所以本章基于上述问题提出了一套地理空间数据并行 IO 的处理机制，并基于这个机制实现了并行构建影像金字塔算法。单幅大影像快速瓦片化方法本文提出了两种实现机制一种是基于影像金字塔实现，另一种是基于瓦片合成的方法。影像金字塔主要用于 MapNIK 等动态实时切图需求，浏览器前端浏览到哪个层级的地图，就调出与其分辨率相近的影像金字塔层进行切片，通过使用影像金字塔可以避免直接对原始影像进行操作，极大减少数据的处理量；瓦片合成方法适用于预处理阶段，通过预先将影像的某一层瓦片切出来，然后基于该层瓦片不断合成下一层瓦片。

3.1　并行构建影像金字塔

影像金字塔是一种栅格数据的多分辨率组织结构（图 3.1）。简单来说，金字塔结构就是由原始栅格影像开始，建立起一系列不同分辨率的栅格影像，不同分辨率的栅格影像对应不同的金字塔级，同时金字塔也是栅格影像的一种有损压缩方式[58]。构建金字塔以后，可以改善栅格影像显示性能，当用户需要对栅格影像进行不同分辨率地放大、缩小或平移时，通过选择一个与用户视图相近分辨率的数据进行可视化，从而系统只需进行少量的计算和查询就可以返回结果，不需

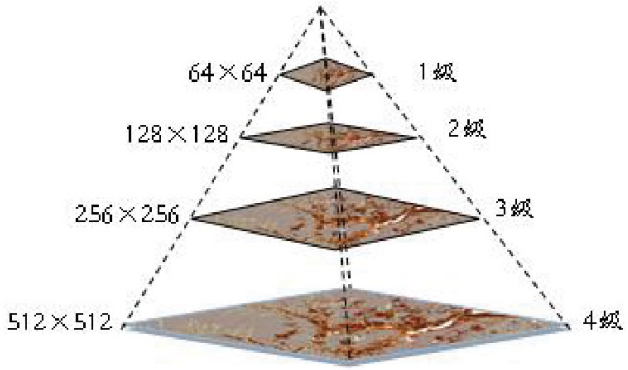


图 3.1　金字塔文件组织方式

要进行逐级采样计算，大大减少数据显示时间[59]。因此当浏览器前端进行放大缩小等浏览操作时，首先找到与当前浏览器视图地图分辨率相近的影像金字塔层级，通过对该层影像金字塔层进行实时切片，可以提高切片效率，提升系统相应

速度[61, 62]。对于 WebGIS 来说，其主要功能就是数据发布与实时浏览，而要做到当进行放大缩小以及移动等操作时依然能够做到流畅显示，就意味着系统要做到对可视范围内的数据进行实施快速切片。特别是当抗震救灾、军事战争等对数据应急发布要求高的情况下，可能无法预先给你足够的时间将整幅影像进行瓦片化，只能是用户浏览到哪就把这部分数据取出，然后将对应数据进行瓦片化操作，进行显示。因此这对快速构建影像金字塔和快速切片算法提出了很高的要求。

当前在提高金字塔生成效率方面的研究主要集中两个方面，对已有金字塔模型的改进，以及金字塔并行化[63]，例如 Cheng et al. 利用全球划分网格来改善传统影像金字塔模型[64]。为了保证金字塔文件的通用性，对已有金字塔模型的改进已无法大幅金字塔创建效率。个人计算机受限于其自身的软硬件资源，无法快速应对大规模的影像处理任务，在计算机硬件飞速发展，计算资源越来越容易获得的背景下，采用多处理器、多节点的并行处理机制构建金字塔已经成为必然趋势[65–68]。Zheng et al. 提出一种分布式方法来提高金字塔构建速度[69]，在他的方法中其根据等分瓦片的原则分解为多个宫格，通过对每个宫格作为输入数据，并行创建金字塔，但是这种方法对于各节点任务划分以及处理边界宫格数据上比较复杂。Kang et al. 提出基于 CUDA 的 GPU 并行构建金字塔方法[63]，这种方法需要不断将数据从主机内存读入到 GPU 内存，无法利用集群并行，而且会提高系统架构成本。刘义等提出基于 MapReduce 进行遥感影像瓦片金字塔批量构建的方法[70]，这种方法利用并行磁盘系统以及集群技术，可以有效处理大规模影像数据，但是其数据分布存储以及合并耗时较长。赫高进等提出基于 MPI 的大规模影像金字塔并行构建算法，通过对重采样以及 IO 两级并行，取得了不错效果[72]，其未对金字塔文件结构进行深入研究，在创建空金字塔文件后，必须先通过调用 GDAL 的 IO 函数给每个条带写入部分数据后，每个进程才能获取其相应条带的偏移位置，这部分是比较费时的。在处理多波段影像时其采用基于文件视图的 BIP 存储格式，为了应对 BIP 存储方式中同一波段内零碎 IO 的问题，其采用文件视图的方式，虽然这种方式能够提高并行 IO 的效率，但是作用有限，并且随着文件数据量增大，创建文件视图的开销也会越来越大。

为了提高算法效率，本文提出一种名为 ParaOvr 的并行构建影像金字塔算法，其利用共享外存的集群系统以及 MPI/OpenMP 的多进程与多线程混合并行策略来对单幅大数据量影像并行构建金字塔，算法在多进程不足以满足应用需求的情况下，采用多线程进行补充，不仅极大地提高算法效率，而且稳定性也得到了很好的提升。该算法通过深入研究金字塔文件的数据结构，提出一种高效稳定的金字塔文件并行访问技术，即在创建空金字塔文件时预先对金字塔文件结构进行规划，达到每个进程能够对金字塔文件进行精确访问，打破了以往每个进程都必须按固

定条带大小写的限制，同时也彻底解决了常见的黑边现象以及零碎 IO 的问题，大幅提高了并行 IO 的速度，使影像金字塔的构建效率得到进一步大幅提高。

3.1.1 影像金字塔文件数据组织

影像金字塔存储在单个文件中，其文件格式共有两种类型分别为金字塔 (.ovr) 以及递减分辨率数据集 (.rrd)。在 ArcGIS 10.0 之前版本创立的金子塔为 rrd 格式，之后的为 ovr 格式。ovr 文件是目前用于存储栅格数据集的金字塔图层的主流格式，与 rrd 格式相比，ovr 格式的优势在于允许数据压缩和控制金字塔的质量，而 rrd 不允许压缩[73]。最重要的一点 ovr 格式的数据结构是公开的，任何人都有读写的权限。

基于上述特征，本文所研究的并行构建金字塔都是基于 ovr 格式的金字塔。ovr 文件采用与 TIFF 文件一致的数据结构，其文件结构主要分为以下三部分: 分别为 IFH(Imge File Head 文件头)，DATA(数据区)，IFD(Image File Directory 图像文件目录)[74]。如表 3.1所示，IFH 共占 8 个字节，记录了文件的标识、版本号以及第一个 IFD 的偏移位置。

表 3.1 IFH 数据结构

偏移量	描述	取值
0-1	字节顺序标志位	II 或者 MM
2-3	TIFF 标志位	42
4-7	第一个 IFD 偏移量	任意位置但必须是 2 的倍数

IFD 是金字塔文件的标签区，其记录了以下三个部分: 该层金字塔的标签数 (简称 DEC)、每个标签的数据结构 (简称 DE)、下一个 IFD 的偏移量 (简称 NIFD)。结构见表 3.2。金字塔的属性在文件中都是通过标签（DE）来表示（见表 3.3），其个数不定，用户可根据需求进行扩展。每个标签描述该层金字塔的一个属性，例如标签 Tag=0100 代表该层金字塔的宽度，Tag=0101 代表该层金字塔的高度。

金字塔数据是按条带或块方式进行组织的，其每一块存放的位置都是通过 StripOffsets 标签进行定义的，并且 StripOffsets 标签是获取数据偏移量有且仅有的唯一方式，所以金字塔文件中的每块数据可以写入除去 IFH 和 IFD 所在的任何地方，因此在写入金字塔数据时逻辑上相邻的两块数据可能会因物理上不相邻而写入错误[74]。为了实现磁盘快速读写，所以有必要对金字塔数据的存储位置进行预先组织，使数据位置与 IFH 以及 IFD 位置互不干扰，并且同一层金字塔数据在文件上的存储位置连续。

DATA 是金字塔中存放重采样数据的区域，为了实现数据的连续存储，

表 3.2　IFD 数据结构

名称	字节数	描述
DEC	2	该层金字塔的标签总数
DE(1)	12	42
DE(1)	12	标签 1
DE(2)	12	标签 2
		……
DE(n)	12	标签 n
NIFD	4	下一个 IFD 相对文件开始出偏移

表 3.3　DE 数据结构

名称	字节数	描述
Tag	2	本属性的标签编号
Type	2	本属性值数据类型
Length	4	该类型数据的数量
ValueOffset	4	变量值相对文件开始处的偏移量

本文采用 IFH-DATA-IFD 的组织方式，即先写 IFH 再写数据最后写 IFH。如图 3.2所示，以第一层金字塔为例，假设该层金字塔数据的长为 *OvrXsize*，宽为 *OvrYsize*，波段数为 *nBands*，标签总数为 *DEC*，则该层影像数据大小 $Img\_size = OvrXsize \times OvrYsize \times nBands$。在文件开始处 0 ～ 3 字节偏移处按表 3.1所示标准写入 IFH 标志位信息，4 ～ 7 字节偏移处写入第一个 IFD 的偏移位置 $IFD\_Offset = 8 + OvrXsize \times OvrYsize \times nBands$。影像数据字节偏移范围为 $8 \sim 8 + OvrXsize \times OvrYsize \times nBands$ 最后再写 IFD，其大小 $IFD\_Size = 2 + DEC \times 12 + 4$；其中 StripOffsets 标签中第 m 个 strip 的值 $Strip\_Offset = 8 + m \times stripSize$，*stripSize* 为条带大小。

其余层金字塔组织方式与第一层类似，只不过其重采样数据存放的初始偏移位置紧贴上一层金字塔 IFD 的后面。假设金字塔层级总数为 n，则第 lev 层金字塔数据初始偏移量 $DATA\_Offset(lev)(lev = 0, 1, \cdots, n)$ 如式 3.1所示。

$$DATA\_Offset(lev) = \begin{cases} 8 & lev = 0 \\ DATA\_Offset(lev-1) + Img\_IFD & lev > 0 \end{cases} \tag{3.1}$$

其中，

$$Img\_IFD = Img\_Size + IFD\_Size \tag{3.2}$$

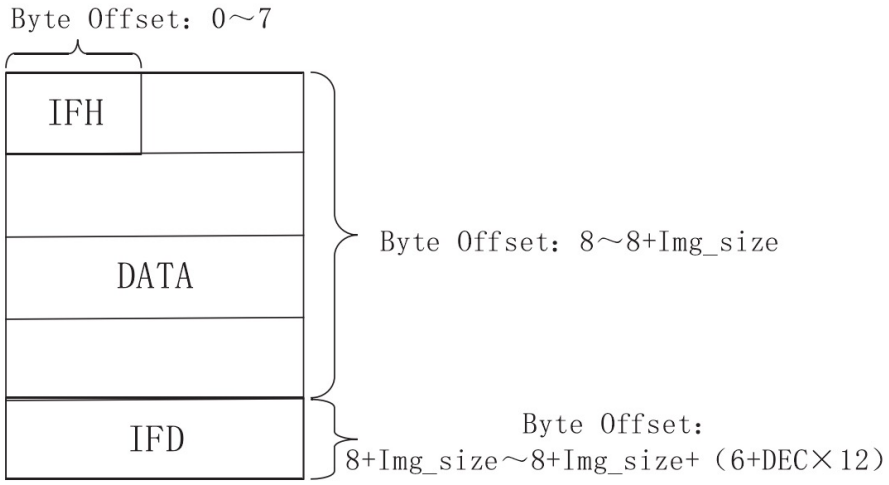


图 3.2　金字塔文件组织方式

第 *lev* 层金字塔第 $m(m \leqslant Ysize)$ 个 *strip* 偏移值 *SRTIP_Offset* 解算方法，如式 3.3所示。

$$SRTIP\_Offset(levm) = DATA\_Offset(lev) + m \times stripSize \tag{3.3}$$

3.1.2　金字塔并行构建任务划分

并行构建金字塔中各个进程的任务划分是一个非常重要的过程，任务分配的好坏直接决定着最终算法的性能。由于采用按行划分效率最高[75, 76] 本文算法采用分波段按行划分的方式进行数据分配，各个进程按照划分好的数据大小，各自从原影像相应偏移处地读取条带数据。如图 3.3所示，假设原影像长为 *ImgXsize*，宽为 *ImgYsize*，参与运算的进程总数为 *n*，因此按行划分原影像每个波段被划分为 *n* 个条带，分别为 Strip0～Strip(n-1)。则可知各个进程 *rank*(i) 读取数据大小 *srcBuffsize* 满足式 3.4。

$$srcBuffsize = srcBuffXsize \times srcBuffYsize \tag{3.4}$$

其中，

$$srcBuffXsize = ImgXsize \tag{3.5}$$

$$srcBuffYsize = \begin{cases} ImgYsize/n & i < n-1 \\ ImgYsize - [(n-1) \times \frac{ImgYsize}{n}] & i = n-1 \end{cases} \tag{3.6}$$

其中任意一个进程 *rank*(i) 在该波段内的起始读取位置 *offset*(i) 满足式 3.7，其中 *i* 表示每个进程的进程号 $(i = 0, 1, \cdots, n-1)$。

$$offset(i) = i \times (ImgYsize/n) \tag{3.7}$$

每个进程根据金字塔级别分别对其所属的数据进行重采样，并分别把重采样结果写入金字塔文件。

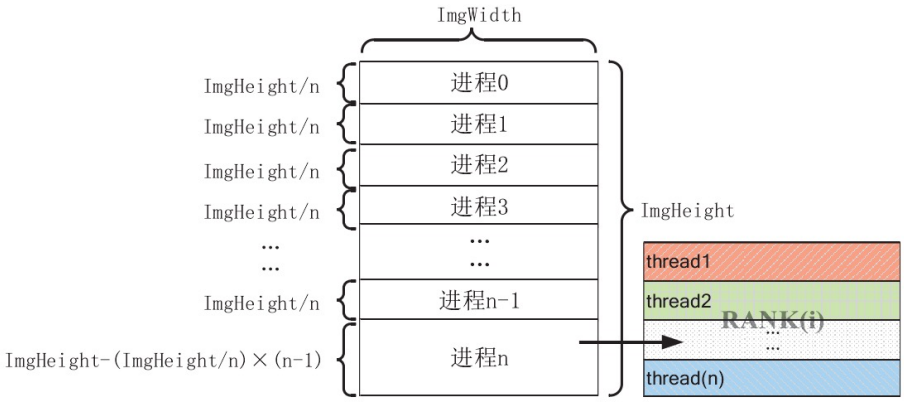


图 3.3　各进程数据划分

基于上述任务划分方法，如果影像过大，而参与的进程数不足的话，会导致每个进程数据量过大，受限于 GDAL 的 RasterIO 函数一次读取数据量不能超过 Integer 类型上限，如果单纯这么划分有可能导致类型溢出。针对这种情况，本文采用多进程与多线程结合的方式，如图 3.3所示，当一个进程的数据量达到 Integer 上限后，采用 OpenMP[77] 对该进程进行多线程化，对该进程数据量进行再次细分，整型类型发生溢出的上限假设为 *TypeMAX*，即整型最大值 2147483647; 则各进程申请的线程总数 2^t 满足以下关系:

$$\frac{srcBuffsize(i)}{2^t} \leqslant TypeMAX \leqslant \frac{srcBuffsize(i)}{2^{t-1}} \tag{3.8}$$

3.1.3　并行重采样方法

每个进程对其所属的数据采用最邻近方法进行重采样，即长和宽方面各每隔 2^{lev} 取一个像素，*lev* 为当前金字塔层级。见图 3.4，图中以进程数 *n*=3 为例对一个 7×11 的影像数据进行第一层金字塔的重采样操作，黑色块为采样结果。进一步推导出各 *lev* 层级相对原始数据的采样步骤，由于边界的问题导致相邻进程即使读取的数据量一样，但最终重采样的大小不一定相同，如图所示 *Rank*(0) 重采样大小为 2，*rank*(1) 重采样大小为 1，*rank*(2) 重采样大小为 3；其原理为: 任务划分采用行划分，影像数据被平均划分给各个进程，但是由于取整的问题，所以除了最后一个进程外其余进程的数据量都是一样的，但是数据量一样，其重采样后大小不一定一样，这里主要是反驳以往常见思维误区。以图 3.4为例，这个影像的大小是 7×11，被三个进程划分，$\frac{11}{3}$ 取整等于 3，所以进程 0 和进程 1 分到三行数据，进程 2 分到 (11-2×3=5) 五行数据，当为其创建第一级金字塔时，按照最邻近

采样法即影像的长宽方向各自隔两个像素点取一个像素，图中黑色部分框格就是相应采样结果，从图中可以看到，各个进程所属数据包含黑色框格行数进程 0 为 2、进程 1 为 1、进程 2 为 3，各黑色窗格在各进程内行偏移起始位置 *RowOffset* 满足进程 0 为 0、进程 1 为 1、进程 2 为 0。因此各进程对其数据重采样时不能简单的从数据位置处进行采样，应该从行偏移 *RowOffset* 处进行重采样才是正确的。

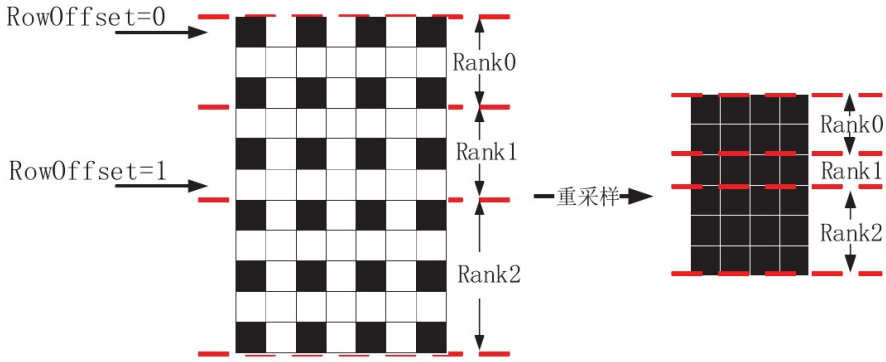


图 3.4　数据并行重采样过程

这种问题主要原因是各个进程采样起始位置 *RowOffset*(i) 不一定相同，其计算方式如下:

$$RowOffset(i) = \begin{cases} 0 & offset(i)\%2^{lev} = 0 \\ 2^{lev} - offset(i)\%2^{lev} & offset(i)\%2^{lev} \neq 0 \end{cases} \tag{3.9}$$

其中% 表示取余。

offset(i) 表示第 i 个进程在波段内的起始读取位置; 各进程重采样结果数据大小 *resBuffsize*(i) 可表示为式 3.10

$$resBuffsize(i) = \frac{ImgXsize}{2^{lev}} \times \frac{ImgYsize/n - RowOffset(i)}{2^{lev}} \tag{3.10}$$

基于这种重采样方法，各进程只需从磁盘中读取一次原始数据到内存，进行不同层级重采样时，只需设置不同的采样间隔，从原始数据中抽稀采样即可，有效地减少了磁盘 IO 的耗时。

3.1.4　重采样数据并行写入方法

如图 3.5所示金字塔文件数据写入分串行和并行两部分，先是基于串行的方法，由主进程 *Rank*0 创建一个空的金字塔文件框架，基于前文所述的 IFH-DATA-IFD 的组织方式，预留出 DATA 的位置空间，把文件头（IFH）以及各层金字塔的文件目录（IFD）利用 LibTIFF 库的 TIFFWriteDirectory() 函数写入框架文件相关

位置中。

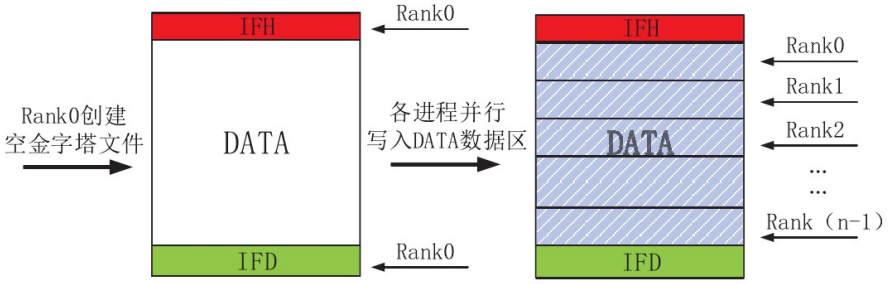


图 3.5　重采样数据并行写入

各进程采用 MPI[71] 提供的并行 IO 接口将重采样后的数据分条带的写入相应的 DATA 数据区内。赫高进等人提出一种基于 MPI 文件视图的聚合 IO 方法[72]，这种方法采用 BIP（波段按行交叉）进行数据组织，以三波段为例如图 3.6所示，即图像按顺序存储第一个像元的全部波段，接着是第二个像元的全部波段，然后是第三个像元的全部波段，交叉存储直至所有像元为止[73]。同一波段间数据不连续，所以其对单波段以及多波段需要分别考虑，特别是其对多波段文件的处理比较复杂，需要创建文件视图。

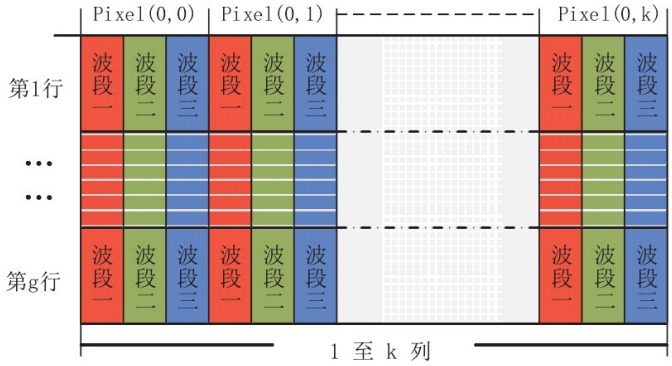


图 3.6　BIP 数据组织结构

针对 BIP 数据组织方式在并行 IO 方面的不足，本文提出一种基于 BSQ（波段顺序格式）的数据组织方式（图 3.7），即存储完一个波段后紧接着存储下一个波段，波段内部数据连续，这种格式最适合于对单个波段内任意部分空间 (x, y) 的存取[73]，这种方式克服了 BIP 方式读写不连续，创建文件视图复杂的缺点。每个波段内数据都是连续的，采用这种数据组织方式，不需要创建文件视图，也不需要对单波段和多波段分别考虑，各进程在并行写入时只需计算出其在金字塔文件中的绝对偏移位置，而后根据偏移位置把重采样后数据连续写入金字塔文件中即可。根据前文所述的波段划分情况，*lev* 级金字塔，第 *q* 波段，第 *i* 个进程 *Rank*(i) 的绝对偏移位置 *AbsOff* 满足式 3.11，其中 *OvrXsize*、*OvrYsize*、*resBuffsize* 分别为前文

所述的当前金字塔层级的长宽以各进程对应的重采样数据大小，*DATA_Offset*(i) 为第 *i* 层金字塔数据初始偏移量。

$$AbsOff = DATA\_Offset(i) + q \times OvrXsize \times OvrYsize + \sum_{j=0}^{i-1} resBuffsize(j) \tag{3.11}$$

在使用 MPI 的并行 IO 函数 MPI_File_write_at 将重采样数据写入时，只需将每个进程绝对偏移位置 *AbsOff*，以及其重采样数据缓冲区大小 *resBuffsize* 作为实参代入函数中，即完成重采样数据的并行写入。

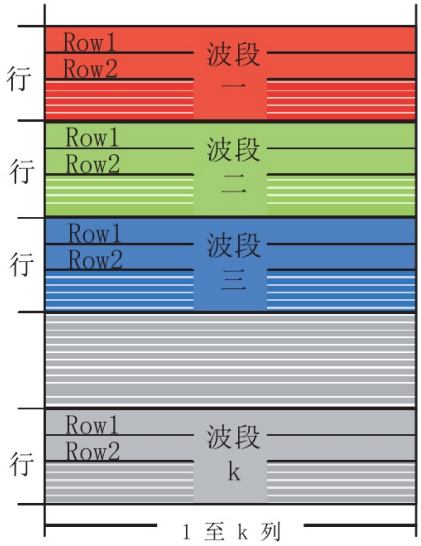


图 3.7　BSQ 数据组织结构

3.1.5　算法执行步骤

本文提出的基于 MPI 的并行构建金字塔方法流程图如图 3.8所示，算法具体描述如下:

步骤 1、*算法初始化*，初始化设定金字塔级数 *level*、进程总个数 *n* 和金字塔级数迭代初始值 *lev* 为 0 级; 消息传递接口（MPI）为每个进程分配一个进程号 *i*，$0 \leqslant i < n$，作为该进程的唯一标识，其中 *i*，*level*，*n* 都为整数; 后续可以通过对该进程号描述相应进程的操作;

步骤 2、*获取原始栅格影像，各进程读取原始栅格影像的元数据信息*; 所述栅格影像的元数据信息包括原始栅格影像的长，宽，波段数，数据类型;

步骤 3、*指定一个进程为主进程，创建空金字塔框架文件*，并对创建的空金字塔框架文件采用多线程技术进行结构组织; 具有过程为: 主进程根据金字塔级数

和栅格影像的元数据信息计算待创建空的金字塔框架文件的大小; 若金字塔框架文件不超过 4GB，则创建 GeoTIFF 格式的金字塔文件; 如果金字塔文件大小超过 4GB，则创建 BigTIFF 格式的金字塔文件; 其中，GeoTIFF 表示地理参考标签影像文件格式，BigTIFF 表示大规模标签影像文件格式; 主进程采用多线程的方式并行设置相应标记位把数据区和标签区的位置划分出来; 而后将每层金字塔的元数据信息写入金字塔文件中标签区相应标记位，将数据区预留; 所述金字塔的元数据信息主要包括该层金字塔的长、宽、波段数、数据类型和该层金字塔各条带数据在数据区内的物理偏移位置。

步骤 4、*数据任务划分*: 按照行划分的方式把栅格影像的数据量平均划分给各个进程，所述行划分方式具体如下: 以像素为单位，栅格影像的长为 *Xsize*，宽为 *Ysize*，波段个数为 *nBands*，进程总数为 *n*，则每个进程分配得到数据量 *DataSize* 等于 $(Ysize/n) \times Ysize \times nbands$；每个进程根据进程号计算其所分配得到的数据块在原始影像上的逻辑偏移位置；

具体计算方式如下: 假设当前进程的进程号为 *i*(0 ⩽ *i* < n)，则该进程所读取数据的逻辑偏移位置 *DataOffset* 为 $i \times Ysizen \times Xsize$;

步骤 5、*读取数据*: 如果每个进程分配的数据量 *DataSize* ⩽ 2147483647 各个进程根据步骤 4 中的数据划分的逻辑偏移位置，采用 GDAL 类库的 RasterIO 函数将栅格影像的各波段数据依次读取到内存；

如果每个进程分配的数据量 *DataSize* > 2147483647，即超过整型上限出现类型溢出的情况，进一步将每个进程下再细分多个线程，细分后的多个线程再对该进程的数据块进行细分，具体细分过程如下: 将 *DataSize* 不断除以 2 进行二分，直到满足以下条件 $2147483647 \times 2^{k-1} \leqslant DataSize \leqslant 2147483647 \times 2^k$ 为止，*k* 为二分次数，取整数; 而后采用 OpenMP 技术在该进程下细分 2^k 个子线程，将进程的数据量 *DataSize* 平均分配给其 2^k 个子线程，则每个子线程读取的数据量为 $DataSize/2^k$ 。当子线程对父进程的数据细分后，各线程分别计算其相对父进程内存空间起始位置的偏移值，具体计算方法如下: 假设当前线程的线程号为 $it(0 \leqslant it < 2^k)$，*it* 取整数，则其偏移值为 $it \times DataSize/2^k$ 。各线程根据相对父进程的偏移值以及父进程的逻辑偏移位置，计算得到各线程从栅格影像上读取数据的逻辑偏移位置等于。由于各子线程是共享父进程的内存空间，因此各线程根据其偏移值利用 GDAL 类库的 RasterIO 函数将个数据并行独立读取到父进程的内存空间中。

步骤 6、*重采样*，各个进程根据当前金字塔级数采用最邻近内插算法依次对步骤 5 中读取的数据进行对应粒度为的重采样，*lev* 为当前金字塔级数；将各波段

重采样后的结果存入内存中；

步骤 7、并行写出结果，各个进程根据进程号以及波段号解算各波段在结果文件中的写入偏移位置；根据写入偏移位置采用 MPI 的 MPI_File_write_at 函数将内存中各波段重采样后的结果依次并行写入金字塔文件；

具体方法如下: 首先利用 libtiff 类库的 TIFFGetField 函数获取到该层金字塔数据区起始偏移位置 *ovrOffset*，设当前进程的进程号为 i，波段号为 *iband*，$0 \leqslant iband < nBands$；以像素为单位当前层级金字塔大小为 $ovrXSize \times ovrYSize$，则各进程写入偏移位置为 $ovrOffset + ovrXSize \times ovrYSize \times iband + resBuffsize$，*resBuffsize* 为前 i 个进程重采样数据大小之和。

步骤 8、循环生成其余金字塔层，将当前金字塔级数加 1，如果当前金字塔级数小于等于设定金字塔级数，则返回步骤 6；如果当前金字塔级数大于设定金字塔级数，则金字塔构建完毕。

3.1.6 实验与分析

为了保证实验准确性，实验平台采用两台硬件环境一模一样的超微计算机，其中一台安装 Linux 操作系统用于运行本文算法、GDAL 金字塔算法、以及赫高进等人提出的算法，另一台安装 windows 操作系统用于运行 ArcGIS 金字塔工具，超微详细硬件环境见表 3.4。

表 3.4 算法实验环境

类型	描述
处理器	Intel(R) Xeon(R) CPU E5-4620 8core×4（双线程）
内存	512GB
操作系统	Windows7 64 位、CentOS6.3.X86_64(Linux)

实验数据采用不同大小级别的单波段以及多波段影像，分别对其构建 9 级金字塔，重采样方法为最邻近法，压缩方式为无压缩，实验数据详细信息见表 3.5。

实验 1、评价程序的并行化程度。实验将上述六个数据按照数据类型、波段数、大小分为三组，分别为 {1.TIF、2.TIF}，{ 3.TIF、4.TIF}，{ 2.TIF、3.TIF、5.TIF、6.TIF}。分别统计上述三组实验数据算法总耗时随进程变化的情况，取 10 次测试结果中去除最大值和最小值后的平均值为实验结果。

从图 3.9和图 3.10可以看到不同组数据随着进程数变化算法的耗时情况，总结如下: (1) 一定范围内随着进程数的增加，算法效率逐步提升，但是随着进程数持续增加，算法耗时逐步趋于一个稳定值，如果再继续增大，算法速度在某一程度

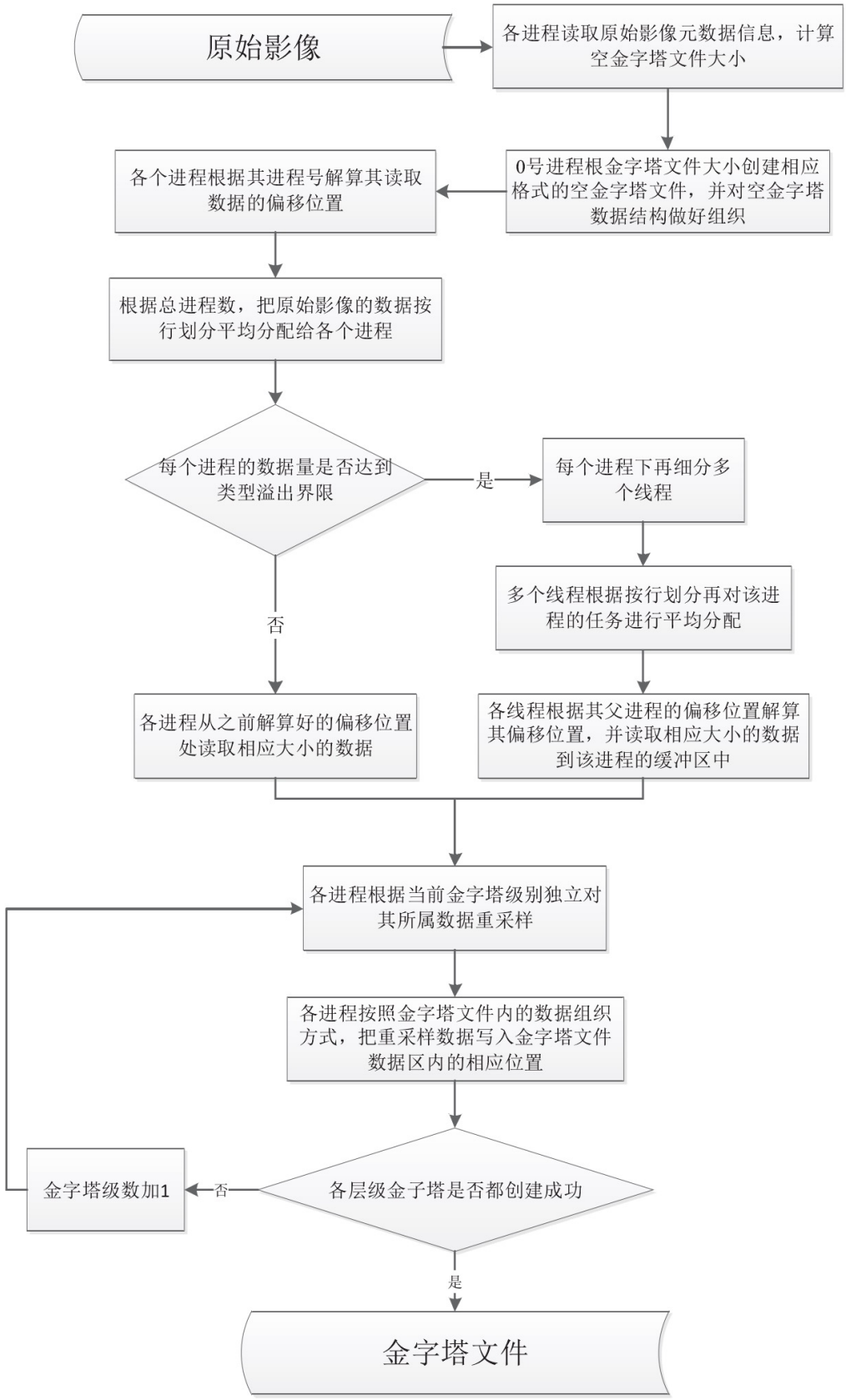
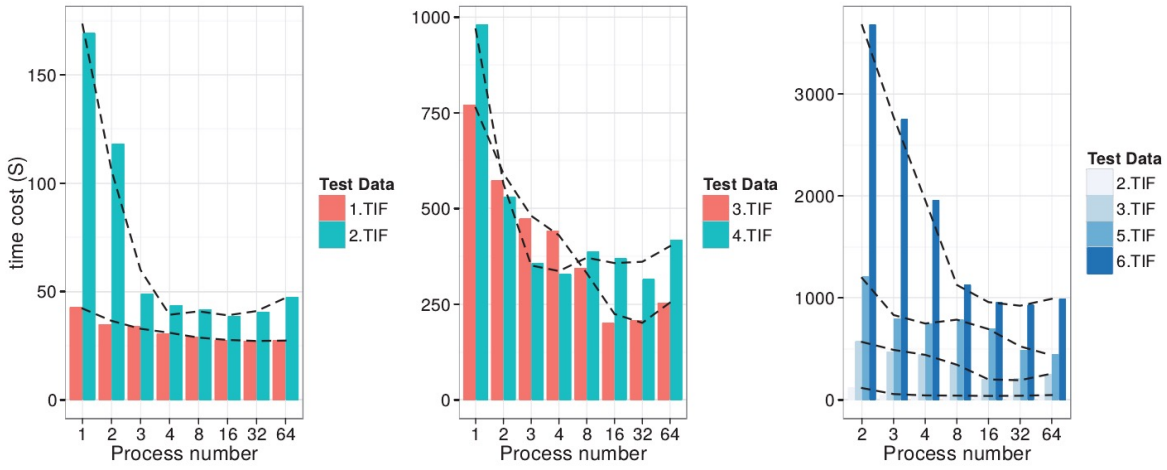

原始影像
各进程读取原始影像元数据信息，计算空金字塔文件大小
0号进程根金字塔文件大小创建相应格式的空金字塔文件，并对空金字塔数据结构做好组织
各个进程根据其进程号解算其读取数据的偏移位置
根据总进程数，把原始影像的数据按行划分平均分配给各个进程
每个进程的数据量是否达到类型溢出界限
是
每个进程下再细分多个线程
多个线程根据按行划分再对该进程的任务进行平均分配
否
各进程从之前解算好的偏移位置处读取相应大小的数据
各线程根据其父进程的偏移位置解算其偏移位置，并读取相应大小的数据到该进程的缓冲区中
各进程根据当前金字塔级别独立对其所属数据重采样
各进程按照金字塔文件内的数据组织方式，把重采样数据写入金字塔文件数据区内的相应位置
各层级金子塔是否都创建成功
否
金字塔级数加1
是
金字塔文件

图 3.8　算法流程图

表 3.5　算法实验数据

名称	尺寸（Pixel）	数据类型	波段数（个）	大小（GB）
1.TIF	72001×48001	Float32	1	12.9
2.TIF	87040×58368	Byte8	3	14.2
3.TIF	220672×86272	Byte8	3	53.2
4.TIF	432001×144001	Byte8	1	115.9
5.TIF	136448×90368	Byte8	3	137.81
6.TIF	428142×232572	Byte8	3	278.2

反而会出现下降；（2）随着数据量的增大，算法的并行化效率愈加明显。通过深入探究出现这种现象的原因，得知随着进程数的增加，任务被划分给更多进程执行，每个进程对应任务规模相应更小，所以算法整体性能得到提升。当进程数目增大到一定程度后，受限于硬盘的读写速度，算法性能趋于稳定，但是如果继续增大那么可能出现各进程读写竞争的情况，导致性能下降。所以针对不同规模的影像，应采用相应合理的进程数才能获得最优的执行效率，通过测试表明本文提出的并行构建金字塔算法对不同规模的影像具有良好的扩展性。



(a) 数据类型划分　(b) 波段数划分　(c) 大小划分

图 3.9　多情况下算法加速比对比

实验 2、ParaOvr 与 GDAL 性能对比。GDAL 是一个功能强大的地理栅格数据转换库，其有个名为 gdaladdo 的金字塔生成工具。在同一硬件环境下，实验选取上述 6 幅遥感影像，在不同影像规模大小下通过比较并行金字塔算法性能与 GDAL 算法性能，实验结果见图 3.11。实验数据采用 10 次测试结果中去除最大值以及最小值后的均值。

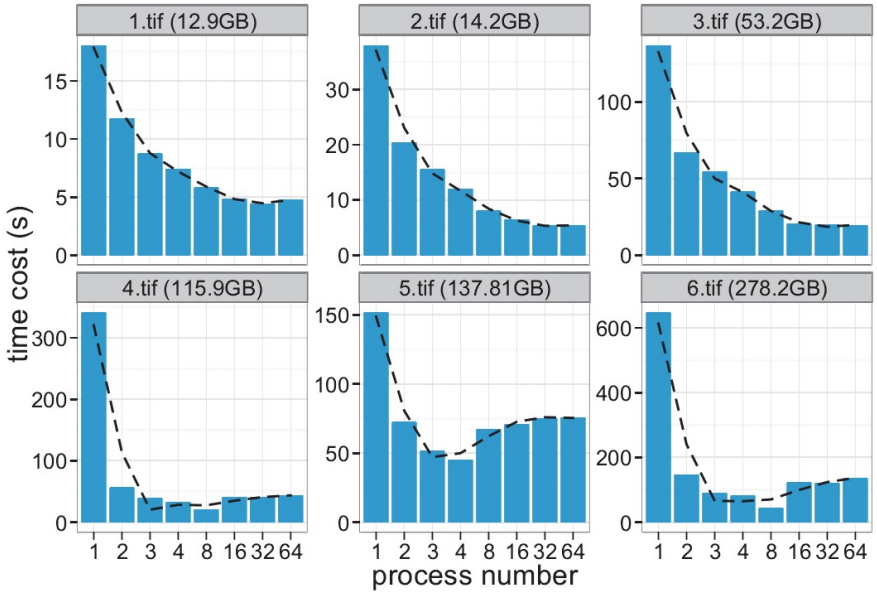


图 3.10　各数据并行程度

通过实验分析，可以得知随着数据规模的增大，GDAL 金字塔算法的性能越来越差，当测试数据规模达到 100GB 时，GDAL 已经远远无法满足应用需求了，当数据规模达到 200GB 以上时，GDAL 在运行 135 个小时后直接错误异常无法再继续执行，性能相比 GDAL 产生了数量级的提升。经过分析，GDAL 金字塔算法由于是串行算法，其无法跨节点，也无法利用多核计算资源，并且其金字塔数据存储采用固定分块模式，在某种程度上会增加 IO 的读写次数，在处理大规模影像数据时无法利用集群大内存的优势。本文算法在数据划分采用行划分，相比 GDAL 的块划分来说能够更好地利用计算机的内存资源，减少与磁盘的 IO 次数。每个进程处理对应条带的数据，实现了重采样并行与读写并行的双重并行，因此算法加速性能十分明显。

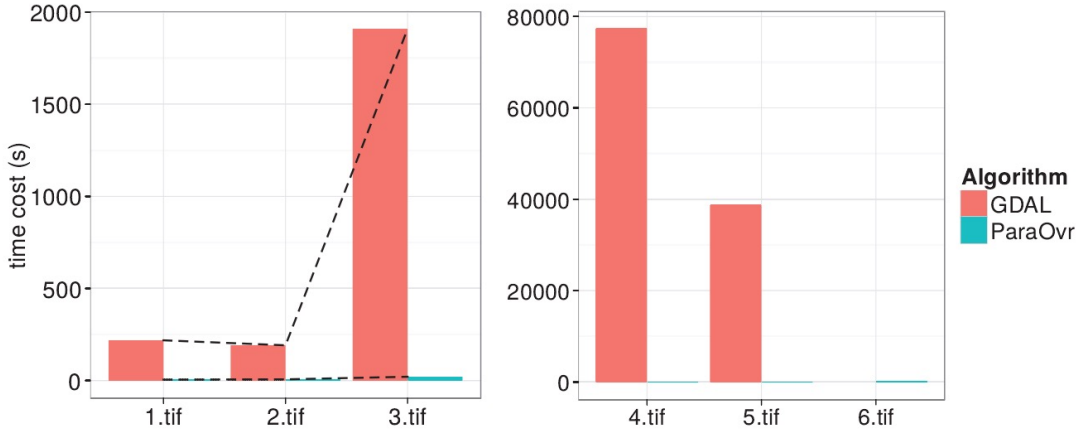


图 3.11　本文算法与 GDAL 性能对比

实验 3、ParaOvr 与 ArcGIS 性能对比。ArcGIS 是目前全球最主流的商业 GIS 软件，实验采用 ArcGIS Desktop10.2，测试数据采用上述 6 幅遥感影像。自 ArcGIS10.0 以后版本，其 ArcToolbox 中的构建金字塔工具支持设置并行参数因子，在相同 16 个进程的条件下，分别测试 ArcGIS 和 ParaOvr 在不同规模影像下的性能，实验结果采用 10 次测试结果中去除最大值以及最小值后的均值，实验结果详细信息见图 3.12。

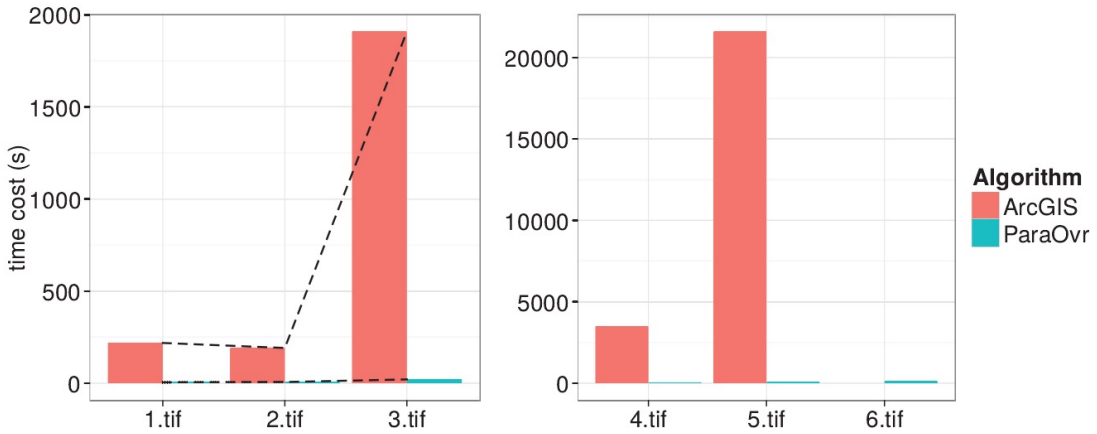


图 3.12　本文算法与 ArcGIS 性能对比

通过实验结果可以看到，虽然 ArcGIS 金字塔构建工具提供并行参数因子，但是其并行效果并不明显。与 GDAL 相比，ArcGIS 在处理单波段数据的性能要好得多，但是同样其在处理大规模影像数据时其效率远远无法达到应用。当用 6.TIF 影像对 ArcGIS 进行了测试，程序在运行到 75h 后，ArcGIS 发生错误异常导致程序崩溃，可以得知当处理数据达到 200GB 以上时 ArcGIS 也已经处于极限无法再进行处理。综上可以看到随着数据规模的增大，本文算法 ParaOvr 效率以及稳定性都远远高于 ArcGIS。

实验 4、ParaOvr 与 MPI 算法性能对比。赫高进等人基于 MPI 实现了一种并行构建金字塔的方法[72]，为了验证本文算法与其性能优劣，实验进程数采用 16，测试数据采用上述 6 幅遥感影像，比较在不同影像规模下两种算法的性能差异，实验结果详见图 3.13。

通过分析，赫高进等人算法针对单波段数据可以保持较高的效率，但是对于多波段数据，由于其采用基于 BIL 的文件视图读写方式，导致读写效率偏低，特别是当数据的波段数越多，数据大小越大时，这种问题越明显，从测试数据 5.TIF 以及 6.TIF 可以看到在处理单幅大规模的多波段影像时，算法也出现异常，无法得到实验结果。综上可以看到本文算法与赫高进等人算法的效率随着波段数大致呈正比例关系，本文算法在针对多波段数据时能够达到与单波段一样的效率，并

且在面对大规模数据的情况下，本文算法效率明显更优。

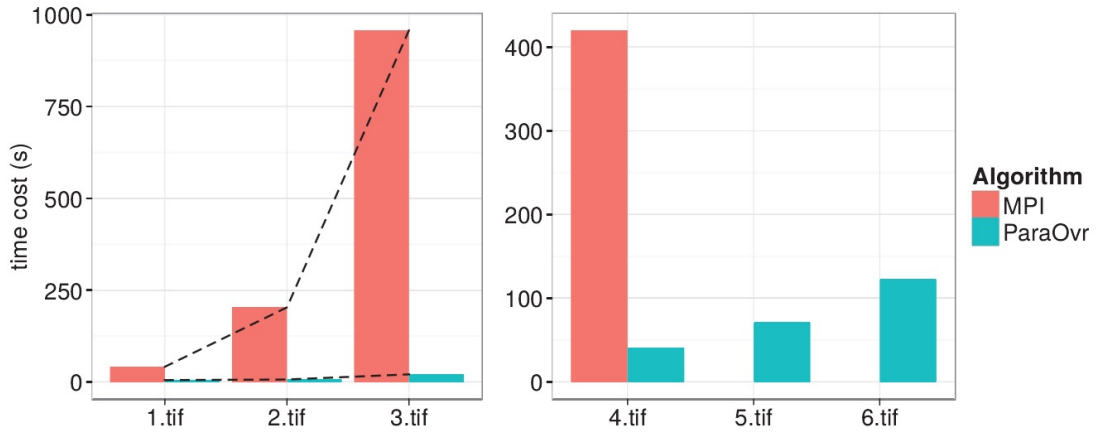


图 3.13　本文算法与 MPI 算法性能对比

3.2　影像数据快速切片算法

影像切片是指将指定地理范围的影像，在某一级别比例尺下，基于一定的切割准则切割成若干行和列的固定尺寸的正方形栅格图片的过程，这些规整的影像切片又称为瓦片。每个瓦片在该级别比例尺下都有一个独立的编码，该编码由瓦片的行列号以及比例尺的级别构成。浏览器浏览影像时，可以根据当前的地理范围动态地计算所需显示的影像瓦片的行列号，通过行列号以及级别得到瓦片的编码，而后通过编码向服务器端请求相应瓦片进行显示，从而达到快速响应目的。前面所述的金字塔算法主要面向浏览器前端进行放大缩小等操作时实时切片的需求；本节所述的影像快速切片算法主要用于预处理，不需要影像金字塔，直接根据原始影像的分辨率，计算与其分辨率最接近的瓦片层级，而后将原始影像瓦片化到该层级，而后通过该层瓦片依次合成低层级瓦片。这种方法只需对原始影像执行一次切片操作，而后就可以合成整个瓦片金字塔，在实时性要求不高的情况下，可以极大提高瓦片缓存的生成效率。

目前大部分瓦片金字塔的生成方法都是必须先生成影像金字塔，而后读取影像金字塔各个层级，再对该层级数据进行切片形成瓦片金字塔。当影像数据量非常大的时候，建立各层级金字塔，以及分别读取各层级金字塔数据进行切片，这是非常耗时的一个操作，目前国内外在瓦片金字塔的快速生成技术上都进行了大量的研究，如王慧，申家双等基于影像金字塔模型提出了一种具有高分布性能的大区域遥感影像管理以及能够快速提供多分辨率影像的瓦片金字塔模型[78]。殷福忠等研究基于瓦片金字塔地图的 Web 地图数据快速发布关键技术，包括瓦片金字塔地图的生成、存储、发布、应用等[79]。

3.2.1 瓦片并行切片原理

前文第二章已经详细介绍了瓦片金字塔模型的原理，本章瓦片切片方法就是基于前文提出的瓦片金字塔模型来进行切分，编码以及存储。算法实现的主要流程和思想如下: 先设置主进程、目标瓦片级别 *level*、进程总数 *n* 以及瓦片输出根目录，而后主进程读取原始影像投影坐标、长宽等元数据信息，主进程利用 GDAL(地理空间数据抽象库) 类库的 GDALAutoCreateWarpedVRT 函数将原始栅格影像投影变换到 WebMercator 投影坐标系下，投影变换结果影像以 vrt 格式文件进行保存，其余进程阻塞直到主进程将投影变换结果影像生成完毕为止。

后续的所有切片操作都是基于上述所生成的投影变换结果影像进行，主进程投影变换结果影像生成完毕后，各进程同时打开投影变换结果影像，根据影像的空间分辨率计算其所能切出瓦片的最大级别 *tmaxz* 和最小级别 *tminz*，判断当前设置的目标瓦片级别 *level* 是否大于最大级别 *tmaxz* 或小于最小级别 *tminz*，如果大于最大级别 *tmaxz* 则将 *tmaxz* 重新赋值给 *level*，如果小于最小级别 *tminz* 则将 *tminz* 重新赋值给 *level*。

而后各进程计算当前 *level* 级别下投影变换结果影像地理范围内所覆盖的瓦片行列号范围。根据瓦片行列号范围，采用车轮法为每个进程分配具体处理的瓦片的行列号，所有待处理的瓦片根据瓦片的行列号按照从上到下、从左到右的顺序构成该进程的任务池。各进程从任务池中依次取出瓦片的行列号，根据瓦片行列号以及级别在瓦片输出路径下创建如下目录以及空瓦片文件:“root/level/tx/ty.png”，其中 *root* 为瓦片输出根目录，*level* 为目标瓦片级别，*tx* 为瓦片列号，*ty* 为瓦片行号，字符串“root/level/tx/ty.png”作为瓦片的唯一编码，浏览器前端可以通过这个编码对应的 URL(Uniform Resource Locator，统一资源定位符) 直接访问相应的瓦片数据。

空瓦片文件创建完毕后，各进程根据瓦片级别 *level* 反算该行列号的瓦片其对应的地理坐标范围，根据地理范围求解其与投影变换结果影像的相交区域，计算相交区域相对于投影变换结果影像中的像素坐标以及范围，而后根据像素坐标及范围利用 GDAL 类库的 RasterIO 函数将该部分相交区域数据从投影变换结果影像中读取到该进程的内存空间中。随后各进程将相交区域数据从投影变换结果影像的分辨率下重采样到瓦片 *level* 分辨率下，最后将重采样数据写入之前创建好的空瓦片文件中，则当前瓦片生成完毕。如果进程的任务池中还有瓦片未曾处理，则依次对下一个瓦片进行处理，否则该进程瓦片切分任务完成。

当最高层级瓦片生成好后，各进程采用车轮法进行任务划分，每个进程根据上一层级的四个相邻的瓦片的相对位置，通过拼接以及重采样合成下一层级的瓦片。通过迭代不断基于上一层瓦片合成下一层瓦片，直至整个瓦片金字塔的各层

级瓦片都合成好为止。

3.2.2 瓦片并行切分任务划分

瓦片切片各进程任务划分采用车轮法，下面以图 3.14为例来具体说明整个并行切片算法的任务划分规则。图 3.14描述的是一个以瓦片为单位大小为 5×4 的栅格影像被 8 个进程进行切分的任务划分示例，该任务划分方法名为车轮法，其原理就是各进程按照进程号从小到大的顺序，根据瓦片坐标系下瓦片的分布，从左到右，从上到下依次进行分配，当一个周期完毕后，接着上一周期的位置重复进行上述类似操作，直至各进程将所有瓦片分配完毕。可以通过如下公式计算得到每个进程所分配得到的瓦片的行列号: 假设进程的进程号为 *i*，则进程 *i* 分配得到的瓦片其瓦片行列号集合，集合中元素 [*tx, ty*] 均满足式 3.12。

$$\begin{aligned} tx &= tminX + (pos\%twidth) \\ ty &= tminY - \lfloor (pos/twidth) \rfloor \end{aligned} \tag{3.12}$$

其中，

$$pos = j \times n + i \quad j \in [0\,1 \cdots k] \quad k = \lfloor tcount/n \rfloor \tag{3.13}$$

$$\begin{aligned} twidth &= tmaxX - tminX \\ theight &= tmaxY - tminY \\ tcount &= twidth \times theight \end{aligned} \tag{3.14}$$

其中，*n* 为进程总数，*i* 为当前进程的进程号，[*tmaxX, tmaxY*] 和 [*tminX, tminY*] 为对应瓦片的最大行列号和最小行列号，符号 $\lfloor \rfloor$ 代表向下取整，% 表示取余。

如果当 $tcount\%n \neq 0$ 时，进程号 $i < tcount\%n$ 的部分进程还需处理瓦片行列号 $[tx_2, ty_2]$ 满足以下条件的瓦片:

$$\begin{aligned} tx_2 &= tminX + (pos_2\%twidht) \\ ty_2 &= tmaxY - \lfloor (pos_2/twidth) \rfloor \end{aligned} \tag{3.15}$$

其中 $pos_2 = tcount - tcount\%n + i$，% 为取余，$\lfloor \rfloor$ 为向下取整。

对照图 3.14，将具体数值融入公式后对相关原理进行叙述，便于理解本文所提出算法的思想。首先拿到投影变换结果影像后，根据其地理范围解算与其范围有相交的瓦片，因为每个行列号所对应的瓦片其地理范围是一定的，因此得到相交瓦片行列号的取值范围，本文算法最终目的就是要将影像切割成一小块一小块的瓦片，图 3.14将投影变换结果影像切割成 4 行 5 列共 20 块瓦片，即

tcount=20。并假设解算出来的瓦片列号 *tx* 范围为 0~4，*ty* 范围为 0~3，进程总数 *n*=8，则 $k = \lfloor tcount/n \rfloor = 2$，因无法整除，表示所有进程在执行完两轮分配后，进程号 $i < tcount\%n$ 的部分进程即进程 0、进程 1、进程 2、进程 3 要处理剩下的 $(tcount - n \times \lfloor tcount/n \rfloor$=20-8×2)=4 个瓦片，8 个进程总共要执行 3 轮，才能把所有瓦片任务分配完，而前两轮所有进程都要参与，第三轮部分进程参与。以进程 0

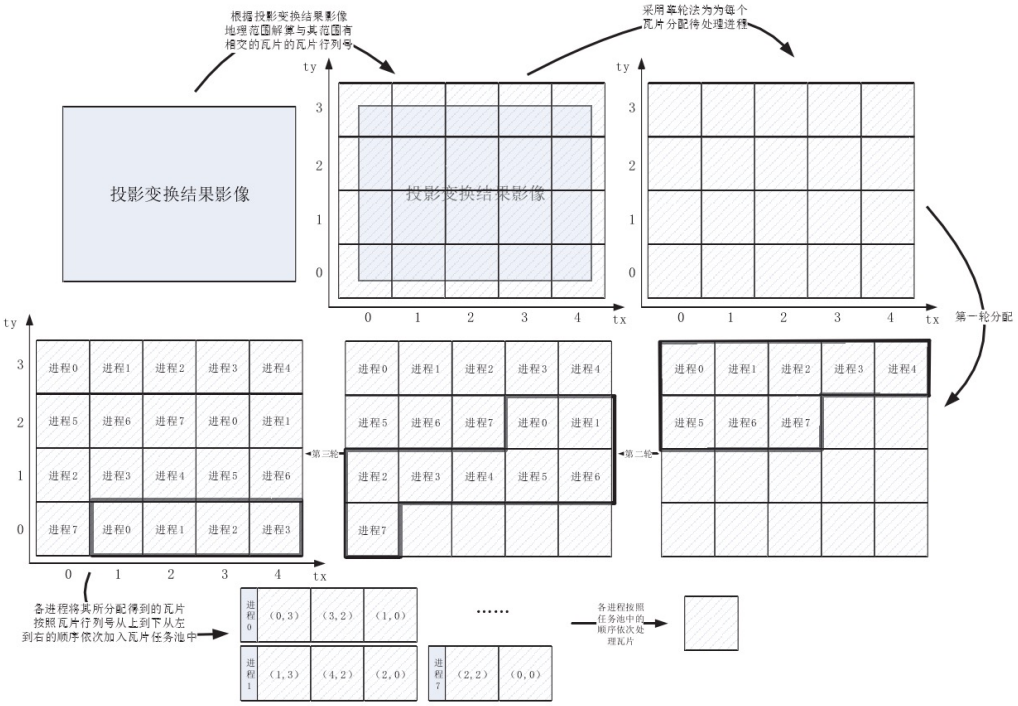


图 3.14　切片任务划分

为例，循环 3 轮，每轮中分配到一个瓦片，在图中按照瓦片行列号从上到下从左到右的顺序分别对应行列号坐标为 (0, 3)、(3, 2)、(1, 0) 的瓦片，将这些瓦片按照顺序依次加入该进程的任务池中，瓦片在任务池中的顺序即序列号 *j*，即 (0,3) 的序列号为 0，(3, 2) 的序列号为 1，(1,0) 的序列号为 2。

为了通过瓦片序列号 j 和进程号 i 反算出对应瓦片的行列号，先要解算中间变量 *pos* 以及 pos_2。以进程 0 的三个瓦片为例，(0, 3) 号瓦片的 *pos* 等于 $0 \times 8 + 0 = 0$，(3, 2) 号瓦片的 *pos* 等于 $18 + 0 = 8$，(1, 0) 号瓦片是针对 $tcount\%n \neq 0$ 的情况，pos_2 等于 $20 - 20\%8 + 0 = 16$。最后通过 *pos* 以及 pos_2 就可以反算所有瓦片的行列号了，比如 (3, 2) 号瓦片的列号 *tx* 就可以这么计算 $tx = (0 + 8\%5 = 3)$，瓦片行号 $ty = (3 - \lfloor 8/5 \rfloor = 2)$；(1, 0) 号瓦片 $tx = (0 + 16\%5 = 1)$，$ty = (3 - \lfloor 16/5 \rfloor = 0)$。这样就将任务池中瓦片的序列号和瓦片的行列号映射了起来，这样各进程只需遍历 *j* 就可以依次处理对应行列号的瓦片。

3.2.3 各进程切片方法

各进程经过任务划分后，确定了各自所需处理的瓦片，根据其所需处理的瓦片构成其进程的任务池，而后每个进程从任务池中依次取出瓦片任务，根据瓦片行列号反算相应地理范围进行切片，图 3.15是本文影像切片过程示意图，首先假设 *rank*(i) 表示第 *i* 个进程，*rank*(i) 当前处理的瓦片行列号为 tx_i ，ty_i ，首先进程

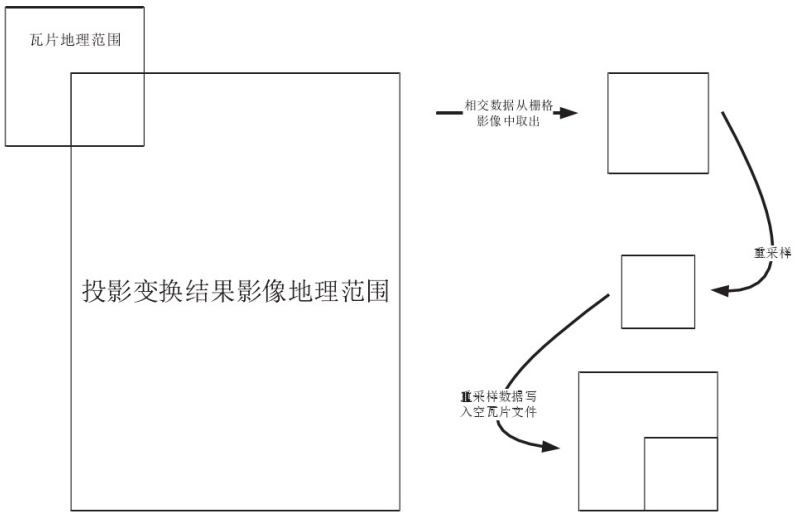


图 3.15 并行切片示意图

rank(i) 在先前瓦片输出目录中的 *level* 文件夹内检查是否已经存在名为 tx_i 的文件夹，如果不存在则在 *level* 目录下创建名为 tx_i 的文件夹。然后 *rank*(i) 根据瓦片行列号以及瓦片级别反算瓦片对应的地理范围 [*gminX*，*gminY*，*gmaxX*，*gmaxY*]，单位为米，具体解算过程如式 3.16。

$$\begin{aligned} gminX &= tx_i \times Resolution - originShift \\ gminY &= ty_i \times Resolution - originShift \\ gminY &= (tx_i + 1) \times Resolution - originShift \\ gminY &= (ty_i + 1) \times Resolution - originShift \end{aligned} \tag{3.16}$$

其中 $Resolution = (2 \times \pi \times 6378137)/(tilesize \times 2^{level})$，$originShift = (2 \times \pi \times 6378137)/2 = 20037508$，单位为 *m*。

然后求解瓦片地理范围与投影变换结果影像地理范围相交的区域，假设相交区域为 [*gistminX*，*gistminY*，*gistmaxX*，*gistmaxY*] ，单位为米，然后计算相交区域在投影变换结果影像中的相对位置以及大小，具体解算过程如算法 3.1所示。而后利用 GDAL 类库中的 RasterIO 函数将 *rx*，*ry*，*rxsize*，*rysize* 填入相应参数位置，将相交区域的原始栅格数据读入 *rank*(i) 的内存空间中。

算法 3.1 解算相交区域在投影变换结果影像中的相对位置以及大小

已知: 投影变换结果影像地理范围 [$gimgminX\,gimgminY\,gimgmaxX\,gimgmaxY$]，投影变换结果影像分辨率 $reslt$

求: 影变换结果影像中以左上角为原点的像素坐标系中的读取位置 rx，ry 以及读取大小 rxsize，rysize。

$rxsize = (gistmaxX - gistminX)/reslt$
$rysize = (gistmaxY - gistminY)/reslt$
if $gistminX == gimgminX$ **then**
    $rx = 0$
else
    $rx = (gistminX - gimgminX)/reslt$
end if
if $gistminY == gimgminY$ **then**
    $ry = 0$
else
    $ry = (gimgmaxY - gistmaxY)/reslt$
end if

由于瓦片的分辨率不一定与影像分辨率一致，因此需要将各进程读入的数据重采样到瓦片相应的分辨率下，重采样方法采用最邻近内插法，重采样后数据大小 *wxsize*，*wysize* 计算方法如式 3.17所示。而后利用最邻近内插法将栅格数据大小从 $rxsize \times rysize$ 重采样到 $wxsize \times wysize$。

$$\begin{aligned} wxsize &= \frac{rxsize \times tilesize}{(gmaxX - gminX)/Geores} \\ wysize &= \frac{rysize \times tilesize}{(gmaxY - gminY)/Geores} \end{aligned} \tag{3.17}$$

根据之前假设，*rank*(i) 当前处理的瓦片行列号为 tx_i ，ty_i ，首先进程 *rank*(i) 在先前的 tx_i 文件夹目录内检查是否已经存在名为 ty_i 的瓦片，如果存在则跳过该瓦片，则从任务池中取出下一个瓦片的行列号，进行下一个瓦片的处理; 如果不存在则 *rank*(i) 利用 GDAL 类库的 CreateCopy 函数创建一个空的瓦片文件，解算重采样数据在瓦片文件中的写入位置 (*wx*，*wy*)，计算方法见算法 3.2。然后利用 GDAL 的 RasterIO 函数将 *wx*，*wy*，*wxsize*，*wysize* 填入相应参数将重采样数据写入创建好的空瓦片文件中。

下面以 *level* 为 10 级，瓦片行列号为 (100, 110) 的瓦片为例阐释解算方法:(100, 110) 号瓦片的行号 *ty*=110，列号 *tx*=100，假设瓦片大小为 256×256 (单位为像素)，其瓦片地理范围为 [*gminX, gminY*，*gmaxX*，*gmaxY*]，那么 *gminX*，*gminY*，*gmaxX*，*gmaxY* 满足式 3.18，可以计算得到为 [−20022220.59, −20020689, −20022065.1, -

算法 3.2 解算重采样数据在瓦片文件中的写入位置

已知: 投影变换结果影像地理范围 [*gimgminX*, *gimgminY*, *gimgmaxX*, *gimgmaxY*], 瓦片对应的地理范围 [*gminX*, *gminY*, *gmaxX*, *gmaxY*], 相交区域范围 [*gistminX*, *gistminY*, *gistmaxX*, *gistmaxY*], 瓦片边长 $tilesize$

求: 数据在瓦片文件中的写入位置 *wx*, *wy*

if $gistminX == gimgminX$ **then**
    $wx = (gistminX - gminX)/(gmaxX\,gminX) \times tilesize$
else
    $wx = 0$
end if
if $gistminY == gimgminY$ **then**
    $wy = (gmaxY\,gistmaxY)/(gmaxY\,gminY) \times tilesize$
else
    $wy = 0$
end if

20020536.1], 然后求解瓦片地理范围与投影变换结果影像地理范围的交集, 假设影像地理范围为 [−20022110.59, −20021579, −20021085, −20020836], 分辨率为 15*m*; 那么可以求解得到相交部分区域的地理范围为 [−20022110.59, −20020689, −20022065.1, −20020836], 将地理范围将相交区域数据从投影变换结果影像中取出, 而后将相交数据从投影变换结果影像的分辨率重采样到 *level* 级别下的瓦片分辨率 (即从 15*m* 分辨率重采样到 152.9*m*), 然后根据相交区域地理范围解算其在瓦片文件中像素坐标系下的相对位置, 最后将重采样数据写入空的瓦片文件中像素坐标系下相应区域。

$$\begin{aligned} gminX &= 100 \times Resolution - originShift = -20022220.59 \\ gminY &= 110 \times Resolution - originShift = -20020689 \\ gminY &= (100+1) \times Resolution - originShift = -20022065.1 \\ gminY &= (110+1) \times Resolution - originShift = -20020536.1 \end{aligned} \tag{3.18}$$

其中 $Resolution = (2 \times \pi \times 6378137)/(256 \times 2^{10}) = 152.9$, $originShift = (2 \times \pi \times 6378137)/2 = 20037508$, 单位为 *m*。

3.2.4 低层级瓦片合成

当最高层级瓦片基于上述所说的并行切分方法切分出来后, 就可以将符合规则的相邻四瓦片进行拼接重采样来合成下一级的瓦片。继续以一幅世界地图为例, *level* 表示地图层级, 地图共被分为 0∼ *levcot* 个层级 (目前主流的地图互联网公司

如 Google Map 等 *levcot* 一般等于 22)，瓦片大小为 *tilesize*（*tilesize* 一般为 256)，每个层级对应相应分辨率的瓦片金字塔图层。各层级瓦片金字塔分辨率 *Resolution* 计算公式如式 3.19。

$$Resolution[level] = \frac{20037508.3427892}{tilesize} \times 2^{levcot} \tag{3.19}$$

每层金字塔共有 4^{level} 个 $tilesize \times tilesize$ 大小的瓦片，由于每一层金字塔的分辨率都是通过上一层金字塔长宽各降采样至原分辨率的二分之一所成，因此可以通过上层 4 块相邻的瓦片来合成下层的瓦片（如图 3.16所示)。

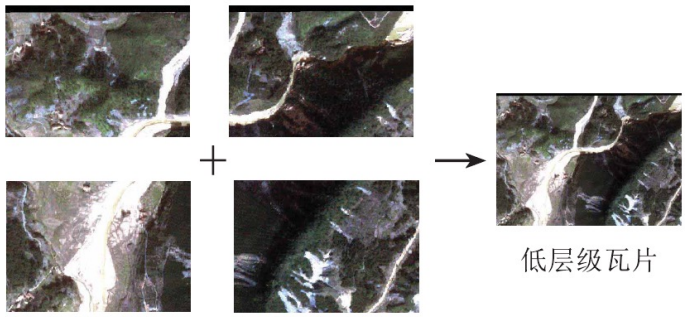


图 3.16　瓦片合成示意图

各进程瓦片合成数据划分采用车轮法如图 3.17所示，根据前文所述的瓦片坐标系统规定，每个进程一次处理一组瓦片，每组包含四块上下左右相邻的瓦片。

各进程采用车轮法，依次处理所有的瓦片数据。因此只需确定每组瓦片中的左上角瓦片的坐标 (*X*，*Y*) 即可确定该组其余瓦片的坐标，假设在该层瓦片集合中的瓦片最小列号为 *tminX*，最小行号为 *tminY*，最大列号为 *tmaxX*，最大行号为 *tmaxY*，根据 TMS 规范，用于合成的每组瓦片中左上角瓦片的行列号应为偶数，所以当 *tminX* 和 *tminY* 为奇数的时候，其不是第一组瓦片的左上角瓦片，应对其向左或者向上扩展一个瓦片[55]。用于任务分配的瓦片外包框 [*minX, minY, maxX, maxY*]，可由式 3.20解算得到。

$$\begin{aligned} minX &= \begin{cases} tminX & tminX\%2 = 0 \\ tminX - 1 & tminX\%2 \neq 0 \end{cases} \\ minY &= \begin{cases} tminY & tminY\%2 = 0 \\ tminY - 1 & tminY\%2 \neq 0 \end{cases} \\ maxX &= tmaxX; \quad maxY = tmaxY; \end{aligned} \tag{3.20}$$

从瓦片坐标为 (*minX, minY*) 的瓦片开始，每四个相邻的瓦片划分为一组，每个

进程1	进程2	进程3	进程4	进程5
进程6	……	进程n	进程1	进程2
进程3	进程4	进程5	进程6	……
进程n	进程1	进程2	进程3	进程4

图 3.17　车轮法任务划分示意图

进程以组为对象进行处理。假设下一层瓦片集合外包框的长为 *wdith*，宽为 *height*，则 *width*，*height* 满足式 3.21。

$$\begin{aligned} width &= \lceil (maxX - minX)/2 \rceil \\ height &= \lceil (maxY - minY)/2 \rceil \end{aligned} \tag{3.21}$$

其中 $\lceil\rceil$ 表示向上取整。

假设进程总数为 *n*，则第 *i* 个进程 *rank*(i) 所属的每组瓦片左上角瓦片坐标 (*X, Y*) 满足式 3.22

$$\begin{aligned} X &= ((k \times n + i)\%width) \times 2 \\ Y &= ((k \times n + i)/width) \times 2 \end{aligned} \tag{3.22}$$

其中 $k = m \times n \quad m \in [0, \mathrm{size/n}]$，$size = widht \times height$；符号% 表示取余。

具体合成方法如下，首先将瓦片的坐标与其文件路径构建一个 Hash 表，使得每个瓦片坐标与其索引路径能够一一映射，而后根据前文所述的任务划分规则，每个进程采取车轮法按顺序处理其所属瓦片组的四块邻近瓦片。而后各进程根据式 3.22计算所属瓦片组的左上角瓦片 Tile1 的坐标 (*X, Y*)，然后容易推导出其邻近的 3 个瓦片的坐标分别为 Tile2(*X+1, Y*)，Tile3(*X, Y+1*)，Tile4(*X+1, Y+1*)[55]。

各进程通过之前 Hash 表中坐标与路径之间的映射关系，根据瓦片坐标获取瓦片所在路径，而后利用 GDAL 库的 RasterIO 函数依次以降采样方式读取这四块瓦片到缓冲区中，即每个瓦片长宽各重采样至原瓦片数据的一半大小[55]（图 3.18）。

数据重采样完成后，开始计算下一层级新的待合成瓦片的索引坐标。假设当前瓦片金字塔级别为 *level* 级，那么其下一级 *level*-1 层瓦片的坐标 (*nX, nY*) 解算方

法可由式 3.23计算得到。

$$\begin{aligned} nX &= X/2 \\ nY &= Y/2 \end{aligned} \tag{3.23}$$

空间索引在空间数据库以及 GIS 中是一项非常关键的技术，其根据空间对象之间存在的某种空间关系，按照一定顺序排列成的一种空间数据结构，它包含着空间对象的概要信息，空间索引直接影响着空间对象的操作速度与效率[52]。目前，绝大多数瓦片地图索引都采用网格索引，这种方式简单方便，本文中采用的 Google Map 瓦片组织方式，具体来说 Google Map 中瓦片的索引机制是 TMS(Tile Map Service) 规范的一个变种，其根据瓦片坐标 (*nX, nY*) 构建结果瓦片的索引路径，假设存放路径根目录为 *root*，则该瓦片的索引路径为“root/level-1/nX/nY.jpg”，按照瓦片的索引路径把缓冲区中的重采样数据保存成瓦片文件输出。往下各层级瓦片操作类似于上述步骤，采用递归的方式，不断基于上一层瓦片来生成下层瓦片，直至所有瓦片最终合并成一张瓦片文件为止。

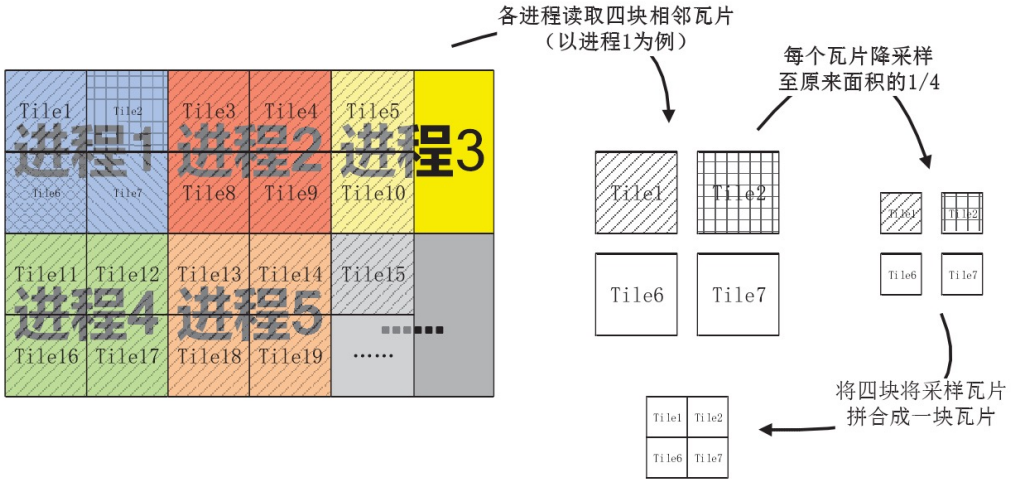


图 3.18　瓦片合成流程示意图

3.2.5　算法执行步骤

总结前文所述的各项原理以及技术步骤，现将影像数据快速切片算法执行步骤总结成图 3.19，具体描述如下:

第一步: 获取原始栅格影像，设置目标瓦片的级别和进程总数。

设定目标瓦片的级别 *level* 以及 MPI 参与的进程总数 n，以及瓦片输出路径。MPI 会为每一个进程分配一个进程号 i（$0 \leqslant i < n$，i 取整数），后续可通过进程号来达到对每个进程单独控制的目的。

第二步: 读取原始栅格影像的元数据信息。

设置其中一个进程为主进程，主进程读取原始栅格影像的元数据信息，包括影像的长宽、波段数，投影坐标系、数据的无效值;

第三步: 将原始栅格影像变换至 WebMercator 投影。

进行切片前首先需将原始栅格影像的坐标系统转换至 WebMercator 目标投影系，然后基于 WebMercator 投影下的坐标进行切分。主进程利用 GDAL 类库读取的原始影像投影坐标系的 WKT 编码，通过对比 WebMercator 的 WKT 编码来判断原始影像是否为 WebMercator 投影系，如果不是则主进程利用 GDAL 类库将原始栅格影像变换至 WebMercator 投影坐标系下，得到投影变换结果影像，以 vrt 格式文件进行保存，并将原始栅格影像的无效值、影像宽度、波段数等元数据信息写入 vrt 文件中，投影变换的详细过程如算法 3.3所示。vrt 文件只记录一些描述性信息，包括投影变换结果影像的长宽、波段数、像素点的数据类型、投影坐标系，以及与原始栅格影像的映射关系。所以 vrt 文件数据量小，因此 vrt 文件用于作为中间文件的投影变换结果影像具有很大的优势，可以减少大量 IO 操作。

算法 3.3 投影变换算法

已知: 原始影像文件名 $Filename$，MPI 进程号 i，主进程号 $master$
求: WebMercator 投影变换影像

```
1:  in_ds = GDALOpen(Filename，GA_ReadOnly)// 打开原始影像
2:  if i == master then
3:      in_srs_wkt = in_ds.GetProjectionRef()// 获取原始影像投影 WKT 编码
4:      OGRSpatialReference out_srs// 获取 WebMercator 投影的 WKT 编码
5:      out_srs.importFromEPSG(900913)
6:      out_srs.exportToWkt(out_srs_wkt)
7:      if out_srs_wkt ！ = in_srs_wkt then
8:          out_ds = GDALAutoCreateWarpedVRT(in_ds，in_srs_wkt,
            out_srs_wkt,
            GRA_NearestNeighbour，0.0，NULL)
9:          out_ds.setnodatavalue()
10:         temp = out_ds.GetDriver().CreateCopy(”result.vrt”, out_ds，0, NULL,
            NULL，NULL)// 投影结果影像保存成 vrt 文件输出
11:     end if
12: end if
```

第四步: 计算投影变换结果影像的瓦片最大级别和最小级别，并更新目标瓦片的级别。

投影变换结果影像的瓦片最大级别为瓦片金字塔层级中最接近投影变换结果

影像分辨率但不大于时的对应层级 *tmaxz*，瓦片最小级别为瓦片金字塔层级中最接近投影变换结果影像不断降采样至一张瓦片后的分辨率但不大于时的对应层级。如果之前设定的目标瓦片级别 *level* > *tmaxz* 则将 *tmaxz* 的值赋给 *level*，如果 *level* < *tminz* 则将 *tminz* 的值赋给 *level*，而后在瓦片输出目录下创建名为 *level* 的文件夹。详情见算法 3.4所示。

算法 3.4 求解影像对应的最大最小瓦片级别

已知: 瓦片单元大小 $tilesize/timestilesize(pix)$，投影变换结果影像长宽 $Xsize \times Ysize(m)$，投影变换结果影像分辨率 $reslt(m)$

求: 投影变换结果影像最大瓦片级别 *tmaxz* 和最小级别 *tminz* 以及目标瓦片级别 *level*

```
GetPixSize(pixelSize) {
R = 6378137// 地球半径（单位 m)
inires = 2 × π × 6378137/tileSize
maxlevel = 22// 世界地图对应瓦片金字塔的最高层级
for each lev in Range(0, maxlevel) do
    if pixelSize > inires/2^lev then
        if i ≠ 0 then
            return i-1
        else
            return 0
        end if
    end if
end for
}// 求解影像分辨率在瓦片金字塔内对应层级
tmaxz = GetPixSize(reslt)// 求解最大瓦片级别
tminz = GetPixSize(reslt × Max(Xsize Ysize)/tilesize)// 求最小瓦片级别[a]
if level > tmaxz) then
    level = tmaxz
end if
if level < tminz) then
    level = tminz
end if
```

[a] Max(a, b): 返回 a，b 中的最大值

第五步: 计算投影变换结果影像的瓦片行列号范围，并按照路径为“根目录 / 目标瓦片级别 / 瓦片列号 / 瓦片行号.png”的路径建立空瓦片文件。

计算当前 *level* 级别下投影变换结果影像地理范围内瓦片的行列号范围，假

设投影变换结果影像地理范围为 [*ominX*，*ominY*，*omaxX*，*omaxY*]，其中该范围为一个矩形四边形。*ominX*，*omaxY* 为左上角点坐标。*omaxX*，*ominY* 为右下角点坐标; 瓦片行列号范围为 *tminX*，*tminY*，*tmaxX*，*tmaxY*，其中该范围为一个矩形四边形。*tminX*，*tmaxY* 为左上角行列号。*tmaxX*，*tminY* 为右下角行列号，则其之间满足式 3.24所示映射关系:

$$\begin{aligned} tminX &= \left\lceil \frac{ominX + originShift}{Resolution \times tilesize} \right\rceil - 1 \\ tminY &= \left\lceil \frac{ominY + originShift}{Resolution \times tilesize} \right\rceil - 1 \\ tmaxX &= \left\lceil \frac{omaxX + originShift}{Resolution \times tilesize} \right\rceil - 1 \\ tmaxY &= \left\lceil \frac{omaxY + originShift}{Resolution \times tilesize} \right\rceil - 1 \end{aligned} \tag{3.24}$$

其中 $originShift = (2 \times \pi \times 6378137)/2$ 即地球周长的一半，$Resolution = (2 \times \pi \times 6378137)/(tilesize \times 2^{level})$；*tilesize* 为瓦片边长大小，$\lceil\rceil$ 为向上取整。

第六步: 任务划分。

任务划分是以瓦片为单位采用车轮法，进程按照第六步任务划分规则，然后根据瓦片行列号按照从上到下从左到右的顺序从任务池中取出瓦片行列号，依次重复进行下面步骤。

第七步: 读取数据。

根据地理范围解算瓦片和投影变换结果影像的相交区域，将相交区域从投影变换结果影像中读出。

第八步: 重采样，并输出瓦片。

将第七步读取出来的数据从投影变换结果影像的分辨率重采样到相应瓦片金字塔级别下的分辨率，将重采样数据写入创建好的空瓦片文件中。如果该进程任务池中还有瓦片任务，则返回第七步依次进行下一个瓦片的切片工作，如果没有则该进程切片任务结束。

第九步: 基于该层瓦片合成底层瓦片。

当第八步把该层瓦片全部生成完后，用递归方法不断通过上层瓦片相邻四个瓦片合成下一层瓦片，直到最终合成到该层只剩一张瓦片为止。

3.2.6 实验与分析

为了保证实验准确性，实验平台采用两台硬件环境一模一样的超微计算机，其中一台安装 Linux 操作系统用于运行本文算法，另一台安装 windows 操作系统用于运行 ArcGIS Service 切片服务，详细硬件环境见表 3.4。实验数据采用不同大

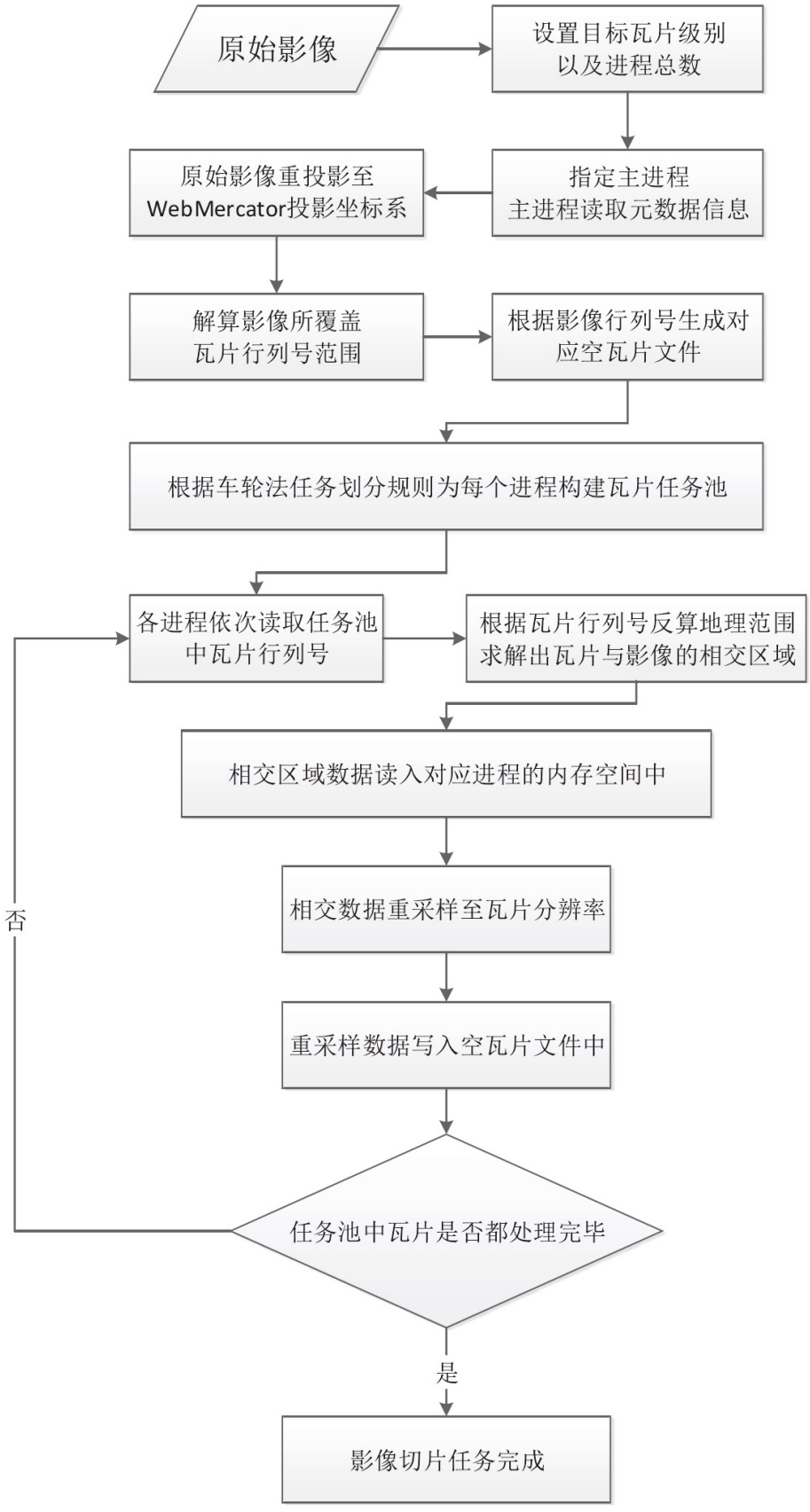

原始影像
设置目标瓦片级别
以及进程总数
指定主进程
主进程读取元数据信息
原始影像重投影至
WebMercator投影坐标系
解算影像所覆盖
瓦片行列号范围
根据影像行列号生成对
应空瓦片文件
根据车轮法任务划分规则为每个进程构建瓦片任务池
各进程依次读取任务池
中瓦片行列号
根据瓦片行列号反算地理范围
求解出瓦片与影像的相交区域
相交区域数据读入对应进程的内存空间中
相交数据重采样至瓦片分辨率
重采样数据写入空瓦片文件中
任务池中瓦片是否都处理完毕
否
是
影像切片任务完成

图 3.19　算法流程图

小，不同分辨率的三波段影像，分别对其进行切片构建瓦片金字塔，实验数据详细信息见表 3.6。

表 3.6　实验数据

名称	尺寸（Pixel）	数据类型	大小	分辨率	最大切片级别	坐标信息
1.TIF	17152×11520	Byte8	605.04MB	2.389 米	15	WebMercator
2.TIF	38265×38376	Byte8	4.62GB	0.00014 度	13	经纬度
3.TIF	87040×58368	Byte8	15.2GB	0.299 米	19	WebMercator

切片级别基于影像分辨率都采用最大切片级别，并行切片算法进程总数设置为 16，由于 ArcGIS 支持多线程加速，为了保证实验可对比性，ArcGIS 切片参数中设置最大线程数 16，图 3.20中纵坐标为算法耗时 (单位为秒)，横坐标为测试影像名称，由于 ArcGIS 不支持对非 WebMercator 投影的影像直接切分成 WebMercator 投影系下的瓦片，因此对应 2.tif 的 ArcGIS 实验数据无法获得，从这也能看到当前商业软件切片方法的复杂与局限性。可以看到本文算法在面对不同大小的栅格影像都保持稳定高效的效率，具有良好的线性加速比，特别是随着影像数据量以及分辨率的提高，这种优势相对 ArcGIS 来说更加明显。针对图中 2.tif 数据量比 3.tif 小，而切片时间却更长的原因主要在于 2.tif 其坐标系统是大地坐标无投影信息，因此需要将影像从大地坐标系投影变换到 WebMercator 投影坐标系，而 3.tif 本身为 WebMercator 投影因此无需进行投影变换，所以这直接影响到了加速性能。

图 3.21是本文算法针对各种规模影像进行切片时执行时间随进程数目变化情况。实验数据采用上述所述的 1.tif，2.tif，3.tif 三幅影像，切片级别采用影像最大切片级别，横坐标是进程数，纵坐标是算法执行时间（单位为秒），表格内数字表示相应算法对应不同数据的耗时（单位为秒）。从图中可以看到:(1) 一定范围内随

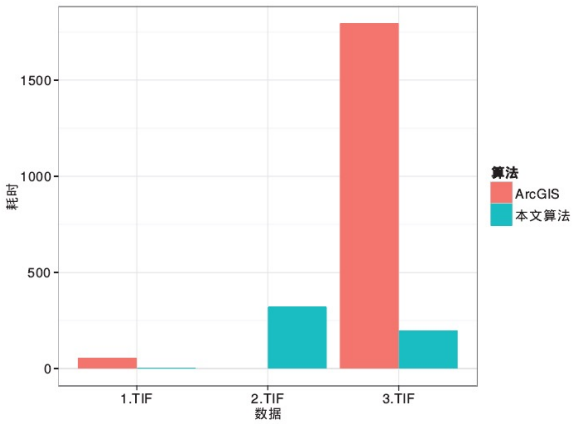


图 3.20　算法与 ArcGIS 性能对比

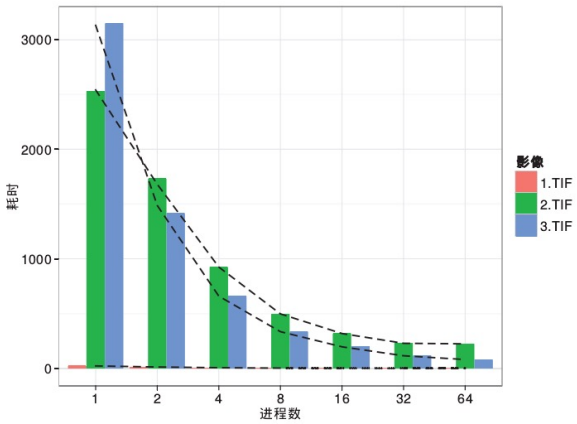


图 3.21　算法耗时随进程变化情况

着进程数的增加，本文算法的效率逐步提升，但是随着进程数的持续增加，算法耗时逐步趋于某一稳定值，如果进程数继续加大，算法速度在某一程度反而会出现下降;(2) 随着数据量的增大，算法的并行化效率愈加明显，并且趋于算法耗时稳定值的进程数也相应更大。其原因如下: 随着进程数的增加，任务被划分给更多进程执行，每个进程对应任务规模相应更小，所以算法整体性能得到提升。当进程数目增大到一定程度后，受限于硬盘的读写速度，算法性能趋于稳定，但是如果继续增大那么可能出现各进程读写竞争的情况，导致性能下降。所以针对不同规模的影像，应采用相应合理的进程数才能获得最优的执行效率，通过测试表明本文算法针对不同规模的影像具有良好的扩展性。

3.3 本章小结

针对单幅大影像快速瓦片化技术目前存在的瓶颈，本章提出的并行构建影像金字塔以及影像数据快速切片算法可以有效利用高性能集群平台来提升单幅大影像瓦片化速度。并行构建金字塔算法用于浏览器前端实时切片请求，影像数据快速切片算法用于预处理提前生成瓦片缓存。并行构建金字塔算法通过深入研究影像金字塔文件的数据组织方式，针对当前一些金字塔构建算法存在的问题和不足，提出一种新的基于 MPI 和 OpenMP 的多进程与多线程混合并行构建影像金字塔的方法，算法通过预先规划金字塔数据组织方式，以及重采样和 IO 两级并行来提高算法效率，算法性能可以随着并行文件系统带宽的扩展得到进一步提升。实验表明本文算法相比现有的金字塔构建方法，性能得到了极大的提升，以 115.9*GB* 影像为例，本文算法金字塔创建速度可以达到 ArcGIS 的 85.3 倍，特别是随着数据规模以及波段数的增大，优势越明显。影像数据快速切片算法，针对目前商业软件切片效率低无法利用分布式集群环境的问题，提出基于 MPI 的多进程并行切片方法，并且通过上层瓦片不断递归合成下层瓦片，这种方法支持断点，当切片过程遇到断电等意外情况时可以从断点处继续进行切片任务，并且基于上层瓦片来合成下层瓦片的策略可以极大提高切片的 IO 效率，加快瓦片缓存的生成速度。从实验结果来看，以 15.2*GB* 影像为例，本文算法在切片速度上可以达到 ArcGIS 的 9.1 倍，并且支持自动对影像进行投影变换，支持跨节点，配置简单，允许断点重续，稳定性高，相比 ArcGIS 来说不仅性能更加优异，而且具有更良好的扩展性与适用性。

第四章　分幅影像数据集的快速瓦片化

在实际生产实践中除了经常要面对单幅大影像瓦片化问题，大量小数据量的影像数据集瓦片化问题也是一个亟待解决的问题，而分幅影像数据集相对于普通的多幅影像其区别在于分幅影像在普通多幅影像数据集的基础上进行了一个分幅标准化的过程，并且数据集内各图幅间存在数据重叠区，其瓦片化过程更加复杂，因为一张瓦片可能需要从多张影像中获取数据，并且数据间还会有无效值数据干扰，因此并不能单纯地将数据抠出并进行拼接。影像分幅非常常见于国土测绘部门，其定义为按一定规格的图廓分割制图区域所编制的地图，简要来说就是指按一定规范将大范围的地图划分成尺寸适宜的若干单幅小地图，以便用于地图印刷及使用，常见分幅形式有经纬线分幅以及矩形分幅[80, 81]：

（1）　矩形分幅其图轮廓为矩形，以直线划分各相邻图幅，常用于局部地区的大比例尺平面图和中小比例尺挂图和地图集，我国大于 1:5 千比例尺的地形图采用矩形分幅这种模式[80]。其又可细分为拼接的矩形分幅地图（内分幅地图）和不拼接的矩形分幅地图（地图集），优点是各图幅之间接合紧密，易于日后镶嵌等相关操作；各图幅印刷面积相对均衡易于控制，可以充分利用当前纸张的印刷版面；图廓线可以避免对重要地物目标进行分割，易于保持重要目标的完整性。缺点主要是图廓线没有明确的地理坐标，地理位置不易精确描绘；整个制图区域只能一次投影，变形较大。矩形分幅其图幅一般为 $50 \times 50cm$ 或 $40 \times 40cm$，以横纵坐标的整公里整百米数作为图幅的分界线，$50 \times 50cm$ 图幅为当前最常用的规格。

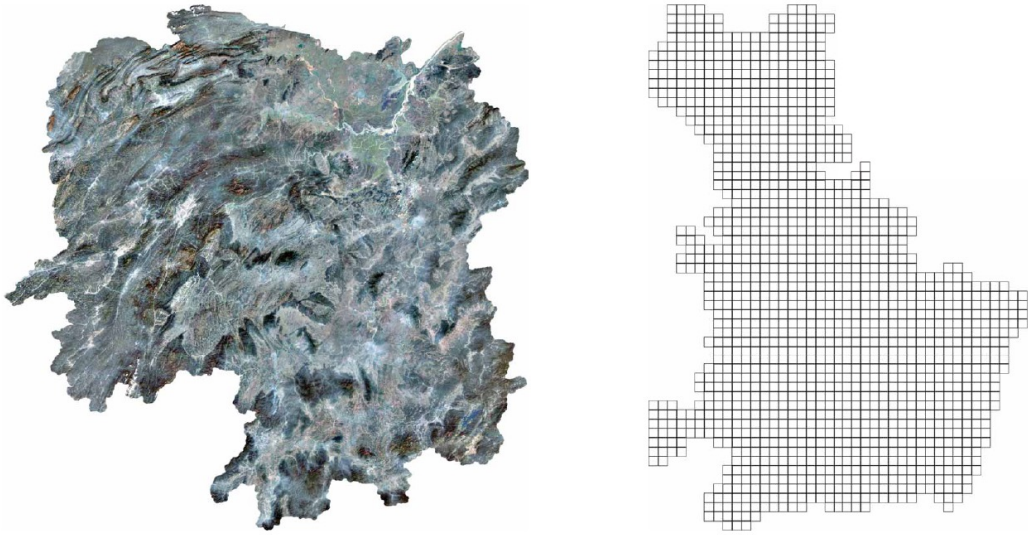
（2）　经纬线分幅其图廓为各边都是曲线的梯形，图廓线由经线和纬线组成，常用于基本比例尺地形图，是当前世界各国大区域的小比例尺分幅地图以及地形图所采用的主要分幅方式之一[80]。其主要优点是经纬度是全球性的统一标准，每个图幅都有明确的地理坐标，便于检索以及计算机发布，并且可以使用分带或者分块投影，便于控制投影误差，变形较小。同样它也不可避免地存在下列缺点，由于投影产生的形变，由经纬线构成的其图廓线不可避免地投影成了曲线，所以不利于后续拼接；随着纬度地升高，图幅面积不断缩小，不便于充分利用纸张；按一定的经差和纬差进行分幅也同样有可能会破会重要地理目标的完整性。

由于遥感影像根据其传感器以及平台的不同，其影像分辨率范围跨度较大，故其一景影像可能跨域多个图幅或小于一个图幅，如何快速取出相交区域，并快速拼合是一个难点，因此为了实现分幅影像快速瓦片化，就必须实现影像的并行分幅裁剪，对于边界图幅存在数据缺失的情况必须考虑裁剪多景影像的部分数据进行并行拼接，因此基于上述技术难点必须实现基于接图表的批量快速分幅输出技术、批量地图瓦片的并行融合技术以及影像数据的并行镶嵌技术。

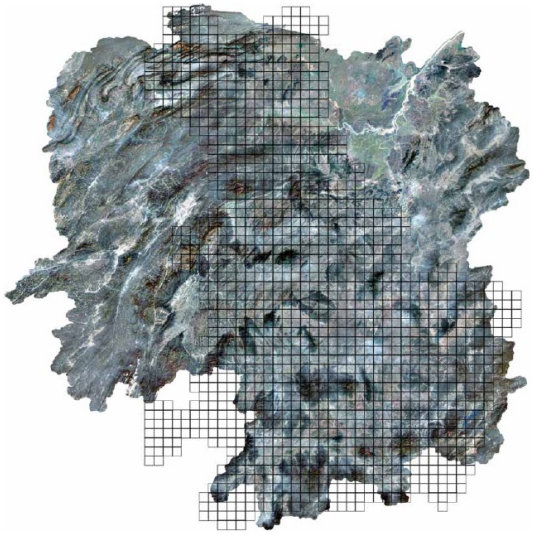
4.1 影像数据并行分幅输出算法

4.1.1 分幅算法描述

影像分幅输出算法主要是通过预先给定接图表，各影像根据接图表中划定的格网范围对数据集中的影像进行切割和拼接，使得最终输出的影像大小范围与接图表中的网格一一对应。以图 4.1为例进行说明，图 4.1(a)为遥感影像，图 4.1(b)为接图表文件，根据地理坐标将接图表叠加至遥感影像上，根据接图表中每个格网的范围对影像进行切割。



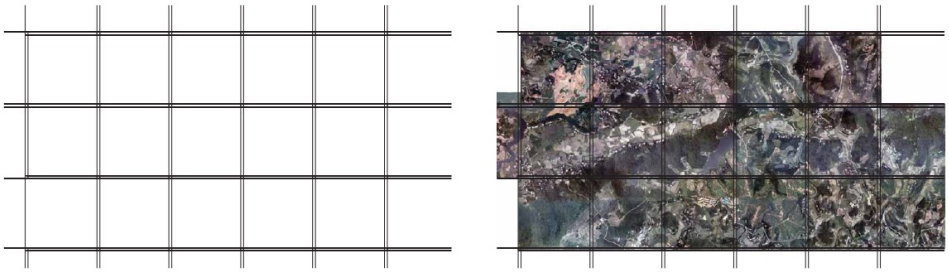
(a) 遥感影像　　(b) 接图表



(c) 接图表叠加至遥感影像

图 4.1　分幅输出示意图

接图表是一个矢量文件，其中有很多要素（Feature），如图 4.2所示，每一个方格就是一个要素，每个要素保存了其地理范围以及一些其他属性信息，而后将与该要素地理范围有相交的遥感影像找出，将其相交部分抠出，最后再拼合成一个跟该接图表文件中对应要素相同范围的影像文件。其中各进程任务划分采用图 3.19所示的车轮法，即每个进程读取一个要素，然后搜索与该要素范围有相交的影像，抠出相交部分，最后再把相交部分拼合成分幅文件进行输出。



(a) 接图表局部放大图　　(b) 分幅结果

图 4.2　分幅过程演示

4.1.2　并行分幅算法执行步骤

图 4.3 为整个并行分幅输出算法的流程示意图，首先读取所有待处理遥感影

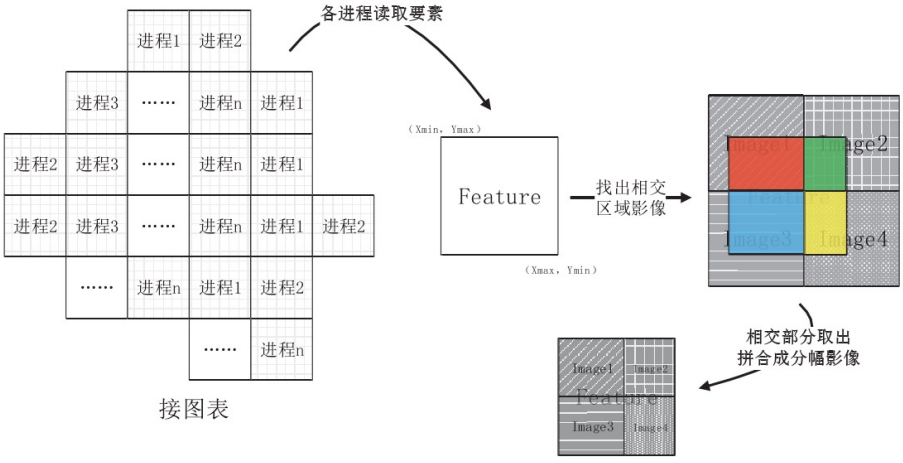


图 4.3　算法示意图

像的地理坐标，获取其四至范围，并基于其四至范围构建 R 树，而后采用车轮法各进程按照任务划分规则依次读取截图表中的要素，根据要素的四至范围创建一个相同大小空的分幅文件，而后将要素的地理坐标范围输入事先构建好的 R 树，查找与该要素地理范围有相交的影像，将影像与要素相交区域的数据取出，根据地理范围将相交数据写空分幅文件的相应位置。假设影像地理范围为 [*ImgMinX,*

ImgMinY, ImgMaxX, ImgMaxY]，以像素为单位长为 *width* 宽为 *height*；要素地理范围为 [*FeaMinX, FeaMinY, FeaMaxX, FeaMaxY*]，影像的空间分辨率为 *Resolution*，那么影像与要素相交部分 [*IntersectMinX, IntersectMinY, IntersectMaxX, IntersectMaxY*] 可由式 4.1计算得出。

$$
\begin{aligned}
IntersectMinX &= \begin{cases} ImgMinX & ImgMinX > FeaMinX \\ FeaMinX & ImgMinX < FeaMinX \end{cases} \\
IntersectMinY &= \begin{cases} ImgMinY & ImgMinY > FeaMinY \\ FeaMinY & ImgMinY < FeaMinY \end{cases} \\
IntersectMaxX &= \begin{cases} ImgMaxX & ImgMaxX < FeaMaxX \\ FeaMaxX & ImgMaxX > FeaMaxX \end{cases} \\
IntersectMaxY &= \begin{cases} ImgMaxY & ImgMaxY < FeaMaxY \\ FeaMaxY & ImgMaxY > FeaMaxY \end{cases}
\end{aligned}
\tag{4.1}
$$

计算出相交部分的地理范围后，则可通过式 4.2计算出相交区域对于接图表要素的相对位置，可通过式 4.3计算出相交区域对于遥感影像的相对位置

$$
\begin{aligned}
ShpStartX &= |IntersectMinX - FeaMinX| \\
ShpStartY &= |IntersectMaxY - FeaMaxY| \\
ShpWidth &= |IntersectMaxX - IntersectMinX| \\
ShpHeight &= |IntersectMaxY - IntersectMinY|
\end{aligned}
\tag{4.2}
$$

其中，*ShpStartX* 和 *ShpStartY* 表示相交区域左上角位置对于接图表要素左上角点的相对位置，*ShpWidth* 和 *ShpHeight* 表示相交区域的长宽

$$
\begin{aligned}
ImgStartX &= |IntersectMinX - ImgMinX| \\
ImgStartY &= |IntersectMaxY - ImgMaxY| \\
ImgWidth &= |IntersectMaxX - IntersectMinX| \\
ImgHeight &= |IntersectMaxY - IntersectMinY|
\end{aligned}
\tag{4.3}
$$

其中，*ImgStartX* 和 *ImgStartY* 表示相交区域左上角位置对于影像左上角点的相对位置，*ImgWidth* 和 *ImgHeight* 表示相交区域的长宽

而后将 [*ImgStartX, ImgStartY, ImgWidth, ImgHeight*] 转化成像素坐标以及像素范围，以便用于从影像中读取相应数据。由于从影像元数据信息中读取的分辨率

信息中其小数点后精度达不到分幅要求中小于一个像素误差的要求，因此在将地理坐标范围转换到像素坐标范围中，我们通过相对比例而不是直接的分辨率计算来达到精度提升的目的，计算过程如式 4.4所示，其中 *ImgRX* 和 *ImgRY* 为从影像上的读取位置，*ImgRWidth* 和 *ImgRHeight* 为从读取大小，将 *ImgRX*、*ImgRY* 以及 *ImgRWidth*、*ImgRHeight* 作为参数代入 GDALRasterIO 函数中即可将相交区域的数据从影像中读取出来

$$\begin{aligned} ImgRX &= \frac{ImgStartX}{ImgMaxX - ImgMinX} \times width \\ ImgRY &= \frac{ImgStartY}{ImgMaxX - ImgMinX} \times height \\ ImgRWidth &= \frac{ImgWidth}{ImgMaxX - ImgMinX} \times width + 0.5 \\ ImgRHeight &= \frac{ImgHeight}{ImgMaxY - ImgMinY} \times height + 0.5 \end{aligned} \tag{4.4}$$

同理将 [*ShpStartX, ShpStartY, ShpWidth, ShpHeight*] 转化为像素范围，而后将相交区域数据写入空分幅文件相应位置中。至此当前进程所对应的数据分幅完成。

4.1.3 实验与分析

实验采用一台超微计算机，操作系统采用 Linux Ubuntu 14.04.1LTS，文件系统采用磁盘整列，编译器采用 GCC4.4.6，MPI 采用 MPICH 3.0.4，详细硬件环境如表 4.1所示。

表 4.1 算法实验环境

类型	描述
CPU	Four Inter(R) Xeon CPU E5-4620,2.60GHz, 8 cores per CPU, Totally 64 virtual CPU cores
RAM	Totally 768GB, Samsung 32GB per each
File system	A disk array with 48TB
OS	Linux Ubuntu 14.04.1 X86_64

实验数据采用湖南宁远县地区高分遥感影像，原始数据集有 67*GB* 大小，在本文中通过在数据集中选取不同规模的数据构建子数据集，而后基于上述构建的不同规模的子数据集测试算法性能随进程数的变化情况，数据集详细情况见表 4.2。

实验采用 10 次测试结果的均值作为最终实验结果，从图 4.4中可以很清楚地看到随着进程数的增大，算法可以一直保持在较高的加速比。算法的正确性与稳定性得到了湖南省测绘院的确认，在湖南省测绘院的实际应用中最大测试数据达到 8*TB*，由于无法获得国土测绘部门所用的传统分幅软件故无法进行对比实验，

表 4.2　测试数据

名称	数据集大小	包含影像数	影像尺寸 (Pixel)	影像分辨率
Dataset0	1.1GB	13	5240 × 5240	0.2 米
Dataset1	3GB	36	5240 × 5240	0.2 米
Dataset2	5GB	61	5240 × 5240	0.2 米
Dataset3	8GB	97	5240 × 5240	0.2 米
Dataset4	10.7GB	130	5240 × 5240	0.2 米
Dataset5	20.7GB	251	5240 × 5240	0.2 米

但据湖南省测绘院相关数据处理员介绍，8*TB* 数据处理完共耗时 3 天，按照以往其工作流程需要耗时 3 个月，因此可以看到本文的并行分幅算法极大提高了国土部门在生产实践中的工作效率。

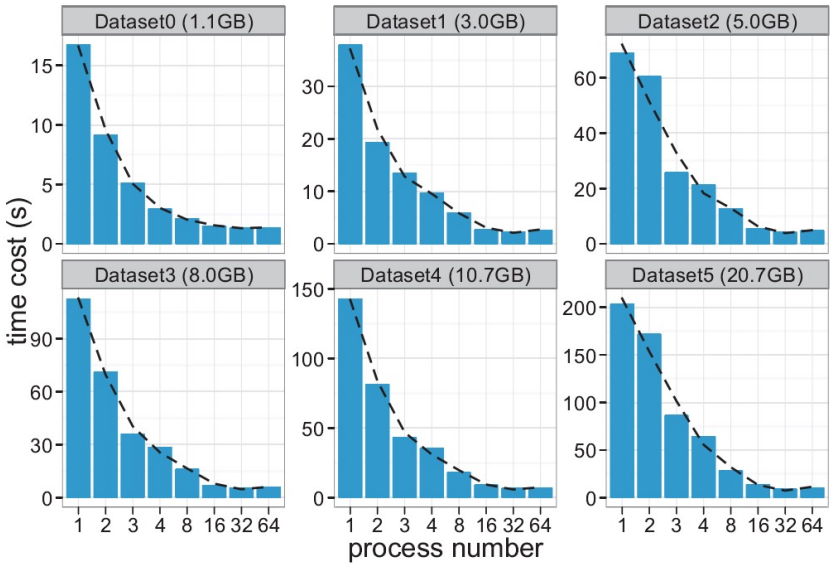


图 4.4　不同规模数据集下并行程度

图 4.5表示的是在 16 个进程下，随着数据集规模的增大，算法的耗时情况，可以看到随着数据量的增大，算法效率能够保持在一个比较高效的水平，算法具有较强的稳定性。图中有一个比较有意思的现象就是 Dataset4 和 Dataset5 的实验结果，虽然 Dataset5 在数据量上是 Dataset4 的两倍，但是算法耗时上却远远没有到两倍，这主要在于算法耗时不仅取决于测试的数据量，而且也跟数据自身跟接图表的匹配程度也有关系，当基于接图表对数据集进行拼接裁剪时，需要通过 R 树搜索与该接图表要素对应区域有相交的影像，如果相交的影像的比较多那就意味着 IO 访问的次数比较多，自然耗时也就长。Dataset5 数据集相比 Dataset4 数据集在增加的那部分数据上，其与对应的要素吻合较好，因此算法能够比较快速地

输出出分幅结果。

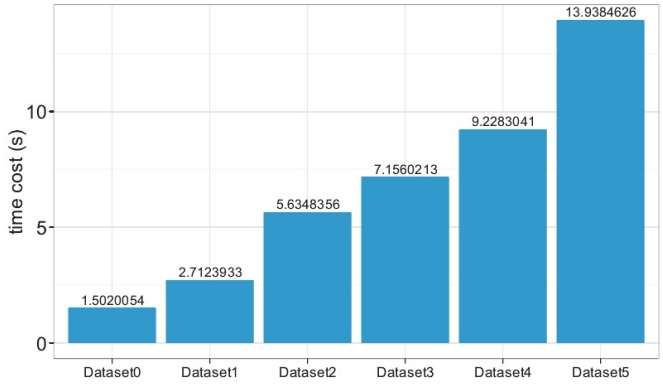


图 4.5　相同进程数下算法效率随数据规模增大的变化情况

4.2　基于 MPI 的分幅影像快速瓦片化技术

分幅输出的影像其结果数据集内影像的数量往往非常巨大，以 0.2 米分辨率的影像来说，一个中小规模的县往往就需要 1000 幅左右影像才能完全覆盖，全部数据量大致在 80GB 左右。并且如果存在多个接图表的情况下，两个接图表相临接的区域必然会存在无效值，如何从对方数据集内找到相应区域的数据并填补自己的无效值部分，是一个必须要解决的问题。这里我们从实际需求出发，提出了两种解决方案。一种是对所有数据集先进行拼接，在拼接的过程中不断利用有效值来覆盖无效值，最后拼合成一幅完整的影像，然后再对该幅影像进行切片。这是一种比较传统的方法，在分幅前进行拼接，而拼接往往比较耗时，但是这种方法有其特定需求，在制作某个城市或者县城的专题图时需要获取其掩膜图像并对掩膜图像进行瓦片化，而获取掩膜图像就必须先将各分幅结果影像进行拼接，而后再对拼接好的影像进行掩膜操作。还有一种方法就是直接对分幅结果影像进行瓦片化操作，如果存在多个接图表的情况下，那么以一个接图表为基准，依次寻找其他分幅文件夹相应层级下与该瓦片相同行列号的瓦片，因为瓦片行列号具有唯一性，相同层级下相同行列号代表着相同地理范围，因此可对这些相同行列号的瓦片进行融合，利用融合操作将有效值替换掉无效值。由于瓦片并行切片算法已由 3.2小节进行了详细阐述，故这里不做累述，只对影像并行镶嵌以及瓦片并行融合进行详细讲解。

4.2.1　影像数据并行镶嵌

遥感影像镶嵌是一个计算密集型和 IO 密集型的的过程，特别是其大量 IO 操作非常耗时，传统方法以及商业软件仅仅只用单机资源，当影像的数据量越来

越大后，其的性能会急剧下降，因此需要提出基于并行 IO 的遥感影像并行镶嵌算法。

本文提出的遥感影像并行镶嵌算法其共分为 5 个过程，如图 4.6所示: 第一步，

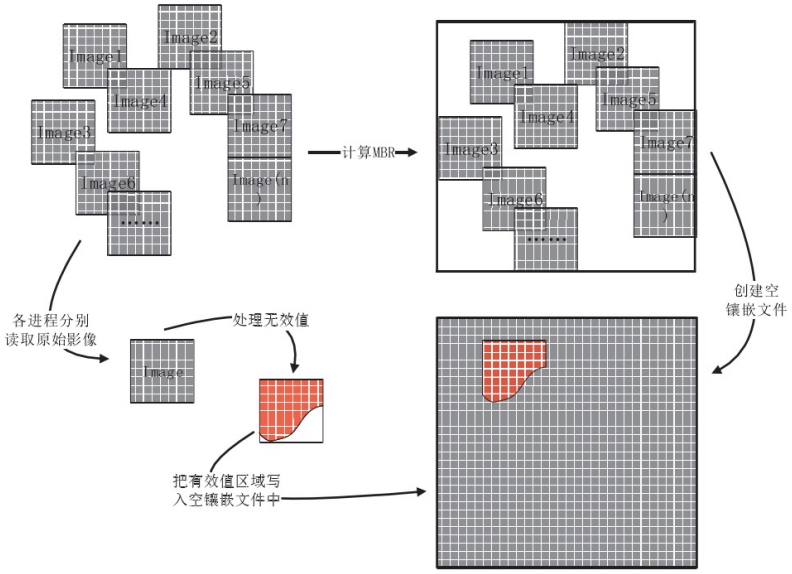
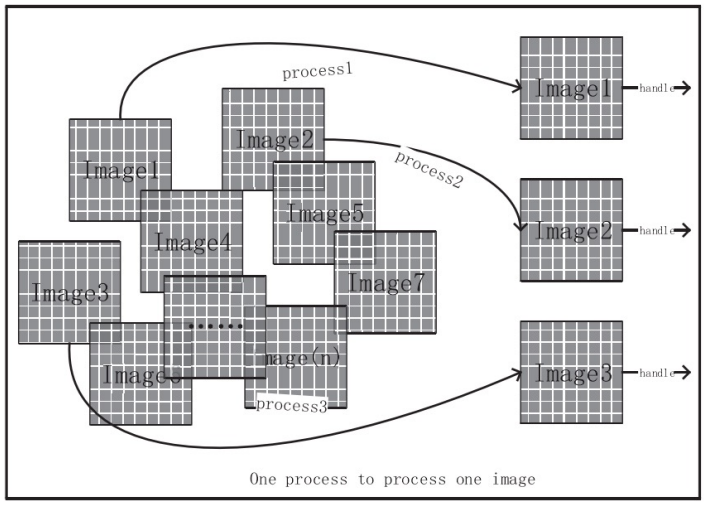


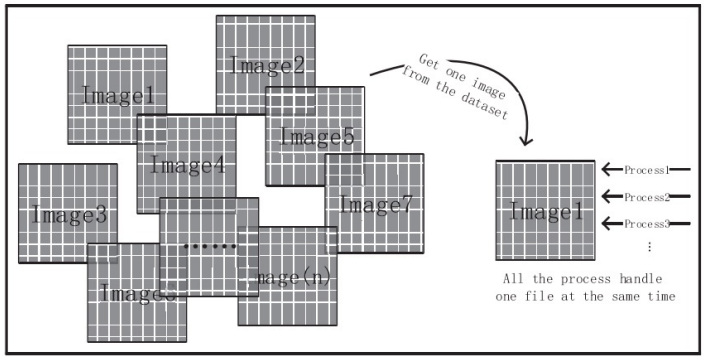
图 4.6　并行镶嵌算法流程示意图

读取所有待镶嵌影像的地理范围最小外包框（MBR）；第二步，计算覆盖所有待镶嵌影像地理范围的最小外包框；第三步，主节点基于上述计算所得最小外包框范围创建一个该大小的空的镶嵌结果文件，随后主节点将分辨率，波段数，以及其他必要的元数据信息写入这个空的镶嵌结果文件中，如果待镶嵌影像分辨率存在不一致的情况，那么镶嵌结果影像的分辨率采取所有待镶嵌影像中分辨率最小的那一个；第四步，利用 MPI 多进程并行方法给每个进程分配任务，每个进程根据预先制定好的任务划分规则各自独立互不影响地从待镶嵌影像中读取数据到内存。任务划分根据数据的具体情况分别采取以下两种方法，当每个待镶嵌影像的数据量不是很大的时候，我们采取的方法是每个进程独立负责一幅影像（图 4.7(a)），采用车轮法为每个进程分配影像，各进程之间影像不会有交叉，每个进程独自处理其分配得到的影像；另一种情况是每个待镶嵌影像的数据量都比较大的话，那么我们采用的策略式多个进程同时处理一幅图像，处理完后再同时处理下一幅(图 4.7(b))，即多个进程对同一副影像进行并行读写。第五步，如果待镶嵌影像的分辨率与镶嵌结果影像的分辨率不一致，那么将待镶嵌影像的分辨率重采样到镶嵌结果影像分辨率下，而后处理无效值，计算数据在镶嵌结果影像中的写入位置，最后各进程将有效值区域数据写入镶嵌结果影像的对应位置。

实验硬件环境在同样的如表 4.3所示的高性能集群上，共有 10 个节点，各节点间网络采用万兆网连接，并行文件系统采用 IBM 的 GPFS（General Parallel File



(a) 影像数据量较小的情况



(b) 影像数据量较大的情况

图 4.7　并行镶嵌任务划分规则

System）[82]。测试数据采用几个接壤省份的 GF-2 卫星高分影像数据，详细信息见表 4.4。

表 4.3　算法实验环境

Type	Details
CPU	Intel(R) Xeon(R) CPU E5-2640 v2, 16cores×12(2.0GHz) + 8cores×8(2.4GHz)
Memory	64G×12 + 16G×8
Internet	Infiniband
File system	GPFS(General Parallel File System) 19TB
Operation system	Linux CentOS6.3.X86_64

以 Dataset1 为例测试算法的并行程度，以 10 次测试结果的平均值作为最终实验结果，图 4.8为实验结果，从图中可以看到在 8 个进程以前，随着进程数的增加

表 4.4　并行镶嵌实验数据

Dataset name	Image information				
	Size	Extend(Pixel)	Band count	Pixel type	Resolution
Dataset1	5.4GB 5.0GB	54241 × 29702 38265 × 38376	3	Byte	1m
Dataset2	5.4GB 5.0GB 4.5GB	54241 × 29702 38265 × 38376 34364 × 39058	3	Byte	1m
Dataset3	5.4GB 5.0GB 4.5GB 5.2GB	54241 × 29702 38265 × 38376 34364 × 39058 34989 × 34851	3	Byte	1m

算法的效率提升很明显，8 到 32 个进程之间算法性能的增加幅度明显降了下来，这主要是算法已经逐渐达到了磁盘读写速度的峰值，算法的性能会趋于一个稳定的值。但是当进程数达到 64 个进程后，由于进程数太多，导致系统分配资源可能存在各进程读写竞争的情况，因此在一定程度上性能可能存在下降的现象。

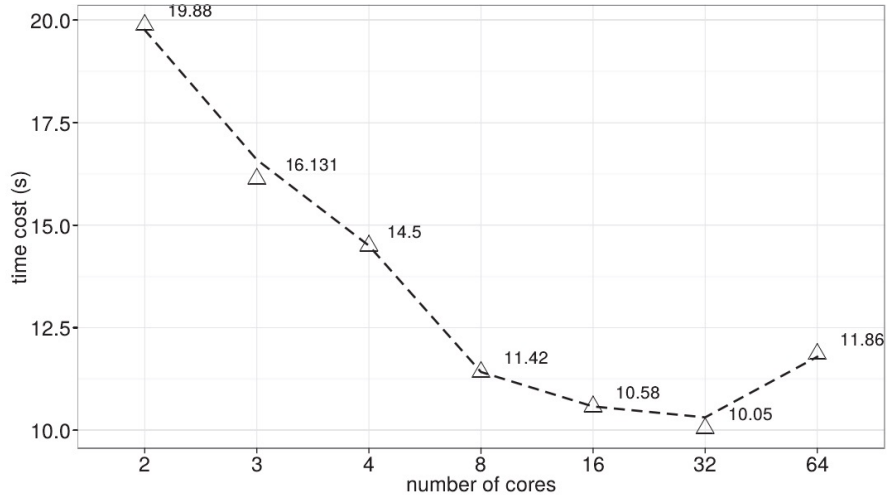


图 4.8　算法并行程度

ArcGIS 是当前全球最主流的商业 GIS 软件，实验采用 ArcGIS10.2 版本，其支持基于单机的多线程并行，为了保证实验对比性，ArcGIS 我们设置其并行参数因子为 16，本文算法设置进程数为 16。GDAL 是遥感业界非常知名的一个功能强大的地理栅格数据转化库，其有一个影像镶嵌工具，但不支持并行。图 4.9是实验的对比结果，其中 paraMosaic 是本文算法的名字，从实验结果图中可以很清晰地看到本文算法相对于 ArcGIS 和 GDAL 在速度上有数量级的提升。

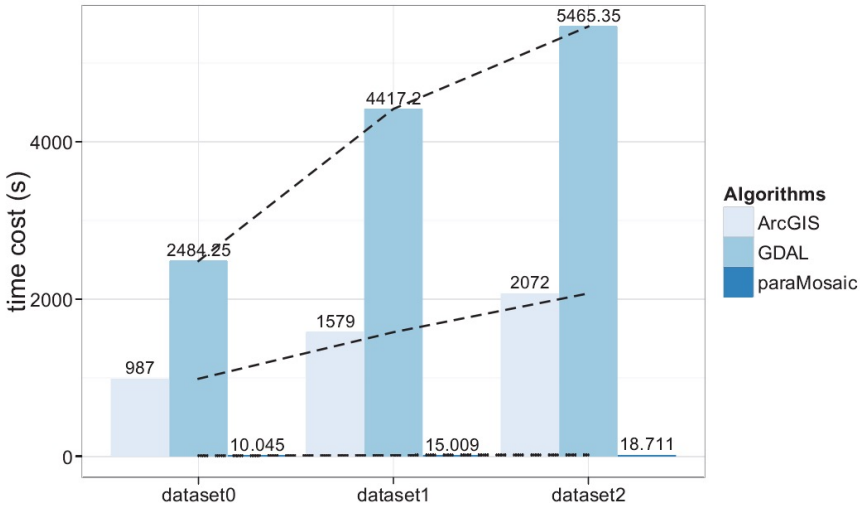


图 4.9　各镶嵌算法对比实验

4.2.2　瓦片并行融合

瓦片融合是为了解决掩膜过的图像或者是接图表边界图像存在无效值的问题，特别是当无效值区域不规则的时候处理难度相对较大。其是在传统先镶嵌后切片的基础上提出来的一种不需要镶嵌直接进行切片的方法，这种方法减少了镶嵌的大量 IO 操作，因此如果没有特殊需求的话，这种方法是分幅数据瓦片化的首选方法。如果不处理无效值的话，两幅相邻区域的影像分别进行瓦片切片后会存在无效值区域把有效值区域覆盖了的问题，如图 4.10所示，展示的是一个瓦片融合的实例，黑色为无效值区域，融合结果把两幅影像的有效值区域接合到了一幅影像中。

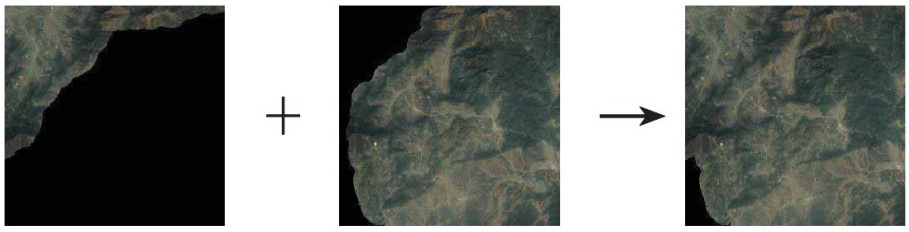


图 4.10　瓦片融合实例

算法实现同样采用 MPI 并行架构体系，进程的任务划分同样采用图 3.17所示的车轮法方式。图 4.11为瓦片并行融合的示意图，具体描述如下：1、任务划分，各进程根据车轮法依次从第一幅影像的瓦片集合中读取瓦片。2、瓦片查询，各进程从第二幅影像的瓦片集合相应层级下查找刚才读取的第一幅影像的相同行列号的瓦片。3、无效值处理，由于两幅相同行列号的瓦片对应的地理范围是一致的，

所以处理无效值时可以直接基于像素坐标系下进行，即不断遍历第一幅影像瓦片的像素点，如果发现像素是无效值点，那么就用第二幅影像瓦片对应的位置上的像素值替换第一幅瓦片的无效值，不断遍历直到所有的无效值都处理完为止。

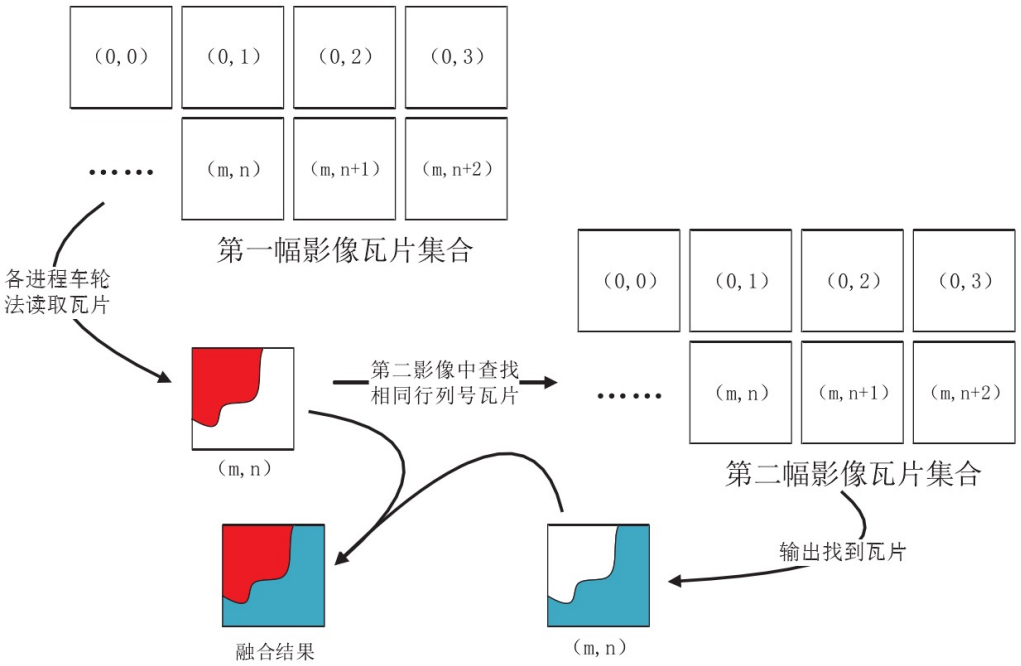


图 4.11　瓦片融合流程示意图

由于目前并没有发现其他有关瓦片融合的算法，故没有进行性能对比实验，只进行了算法正确性验证实验，实验数据采用湖南省常德市几个存在接壤的行政县分幅影像的瓦片数据集，通过人工目视判断融合结果是否正确。经过项目验收，算法准确性以及性能都能很好满足生产工作需要。

4.3　基于分布式的分幅影像快速瓦片化技术

传统 MPI 共享外存方式的并行架构体系由于其存储系统难以扩展，所以 IO 会成为其性能提升的最大瓶颈，并且由于其提供接口位于操作系统底层，因此开发难度大学习成本高，可扩展性差。而现阶段采用分布式文件存储的集群系统就可以很好的解决 IO 瓶颈的问题，并且其组成灵活，成本低廉，配置简单，用户可以按需配置相应数量的机器，性能过剩的机器还可应用于其他用途。

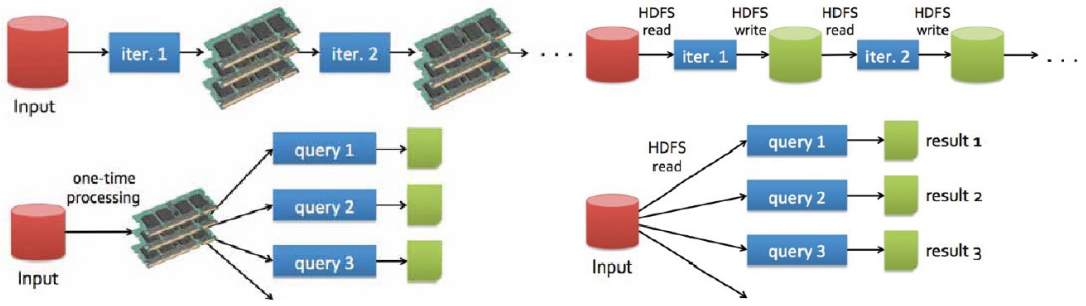
现阶段主流的分布式集群系统主要有 Hadoop，Storm 以及 Spark。Hadoop 是一个分布式系统基础架构，其由 Apache 基金会所开发，用户可以在不了解分布式底层细节的情况下，充分利用集群的威力进行高速运算和存储，进行分布式程序开发[84]。Hadoop 设计组成分为两个部分：HDFS 和 MapReduce[85, 86]。HDFS（Hadoop Distributed File System）是一个分布式存储系统，具有高容错的特点，并

且可以部署到低廉的硬件上，并且支持以流的形式访问文件系统中的数据[87, 88]。MapReduce 框架则为海量数据提供了计算支持，MapReduce 通过把对数据集的大规模操作分发给网络上的每个节点来实现可靠性；每个节点会周期性地返回其所完成的工作和最新状态[89]。

Storm 是由 Twitter 开发并开源的一个分布式、容错的实时计算系统，它主要被用于“流处理”当中，Storm 可以方便地在一个计算机集群中编写与扩展复杂的实时计算，Storm 用于实时处理就好比 Hadoop 用于批处理[90]。Storm 有许多应用领域，包括实时分析、在线机器学习，信息流处理、连续性计算、分布式 RPC、ETL 等。当前使用 Storm 的公司有：Twitter、雅虎、Spotify 等。

4.3.1 大数据计算平台 Spark

Spark 是一个集群计算平台，用来实现通用而快速的分布式计算。在速度方面，Spark 扩展了 Hadoop 的 MapReduce 计算框架，而且支持更多的高效计算模式，包括实时流处理以及交互式查询[91]。如图 4.12所示，Spark 相比 Hadoop 的一



(a) Spark 计算流程　　(b) Hadoop 计算流程

图 4.12　Spark 与 Hadoop 对比

个主要特点就是能够在内存中进行计算，因而更快[92]。Spark 集成了多种分布式平台，包括批处理、迭代算法、交互式查询、流处理。通过在一个统一的框架下支持这些不同的运算，Spark 可以简单而低耗地把各种处理流程整合在一起，Spark 的这种特性大大减轻了原先需要对各种平台分别管理的负担。

Spark 提供了非常丰富的接口，除了可以使用 Python、Java、Scala 和 SQL 提供的简单易用的 API 以及内建的丰富的程序库以外，Spark 还能密切配合其他大数据工具使用。例如，Spark 可以直接运行在 Hadoop 集群上，访问任意的 Hadoop 数据源等。

Spark 平台各个组件紧密集成，Spark 的核心是一个计算引擎，其对运行在多个工作机器或者是一个计算集群上的很多计算任务组成的应用进行调度、分发以及监控[92]。Spark 的各个组件以及与其他大数据平台的关系如图 4.13所示[93, 94]，

下面简单介绍其各组件：

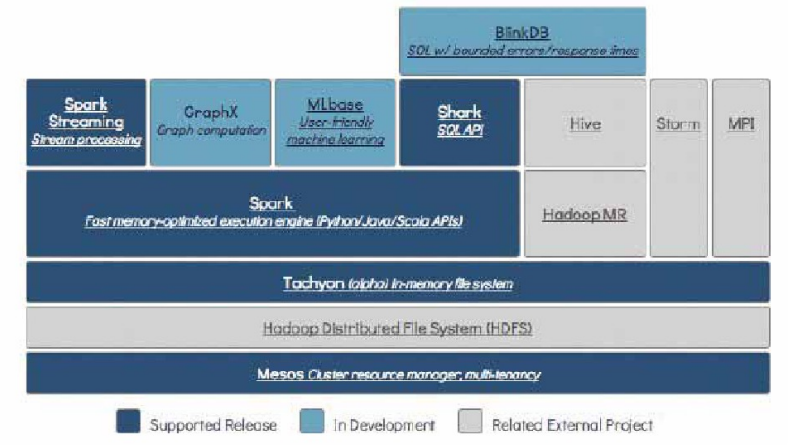


图 4.13　Spark 组件与其余大数据技术关系

（1）　Spark Core 包括了 Spark 的基本功能，包含任务调度、错误恢复、IO、内存管理等模块。Spark Core 中还包括了对 RDD（resilient distributed dataset，弹性分布式数据集）的 API 定义。RDD 是 Spark 主要的编程抽象，其主要定义为分布在多个计算节点上可以并行操作的元素集合。Spark Core 提供了创建和操作这些集合的多个 API。

（2）　Spark SQL 是 Spark 用来操作结构化数据的程序包。通过 Spark SQL 可以使用 SQL 或者 Apache Hive 版本的 SQL（HQL）来查询数据。Spark SQL 支持多种数据源，比如 Hive 表、Parquet 以及 JSON 等，其除了为 Spark 提供了一个数据查询接口，Spark SQL 还将传统 SQL 和 Spark RDD 数据编程的方式进行了结合，无论是使用 Scala、Java 还是 Python，开发者都可以无缝嵌套使用 SQL 语言以及复杂的数据分析接口。在 Spark SQL 发布之前，University of California, Berkeley 曾尝试修改 Apache Hive 以使其运行在 Spark 上，即是曾经的 Shark，由于 Spark SQL 相比 Shark 更加与 Spark 引擎结合紧密，现在 Shark 已经被 Spark SQL 取代。

（3）　Spark Streaming 是 Spark 处理实时数据流式计算的组件。比如服务器中的网页服务器日志分析统计，或是网络服务中用户提交的状态更新组成的消息队列。Spark Streaming 提供的数据流 API 与 Spark Core 中的 RDD APP 深度集成，交互性强，易于开发。

（4）　MLib 是 Spark 提供的机器学习的程序库，其提供了多种机器学习算法，包括回归、分类、聚类、协同过滤等，还提供了模型评估、数据导入等功能。

（5）　GraphX 是 Spark 用来操作图的程序库，可以进行并行的图计算，GraphX 扩展了 Spark Core 的 RDD API，可以用来创建一个顶点和边都包含任意属性的有向图，并支持对图的各种操作以及一些常用图算法。

针对于分幅数据集多和碎的特点，并且其计算模式多为批处理操作，数据和数据间独立性大，因此适合于当前主流的大数据计算框架。通过比较相关的大数据平台，Spark 无论从速度还是编程实现角度都是最佳选择，并且 Spark 支持 HDFS 存储，可以解决栅格影像处理中 IO 瓶颈这一难题。

4.3.2 Geotrellis 描述

Geotrellis 是一个开源的地理空间数据处理框架，Geotrellis 采用 Scala 语言编写，用来在 web-scala 以及 cluster-scala 上支持地理空间数据处理，Geotrellis 主要用来解决栅格影像处理中的三个核心问题[95]：

- 创建一个高度可靠的地理空间数据处理网络服务
- 创建一个地理数据批处理服务，使得能够在分布式环境中处理大规模数据集
- 实现地理数据的并行处理，使得能够充分发挥多核计算机的优势

Geotrellis 是由 Azavea 公司于 2011 发布并开源，其通过集成 Spark 来实现 cluster-scala 批处理操作，其 web-scala 部分已经十分成熟，在一些简单栅格计算（+，-，*，/，等）以及少数复杂栅格和矢量计算如 Kernel Density 和 Cost Distance 都可以达到亚秒级的响应[96]。

Geotrellis 主要实现了三大功能：1、地理数空间数据读写接口（栅格 / 矢量）2、栅格影像操作（地图计算器）3、网络服务创建工具。如图 4.14所示为 Geotrellis 数据处理流程，Geotrellis 提供了在分布式环境下高可靠性地处理大量小数据集的能力，Geotrellis 采用 Akka 框架来实现分布式处理的 Actor 模型，Geotrellis 有 Local、Focal、Zonal 等栅格操作以及少量矢量和网络操作，Geotrellis 会自动对地理处理模型进行并行优化，可以帮助开发者快速构建简单，标准的 REST 服务[97]。

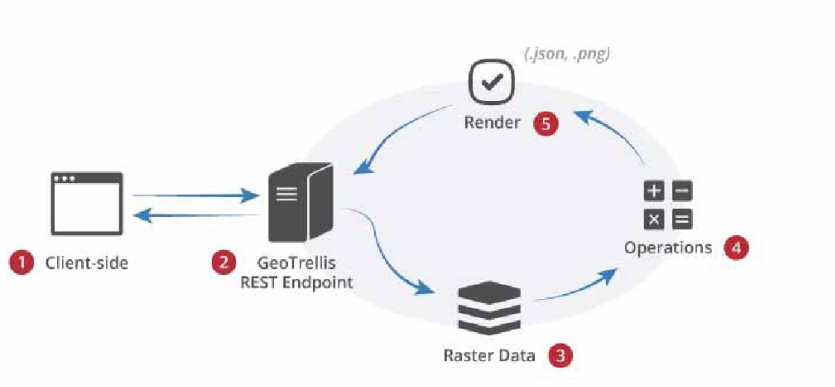
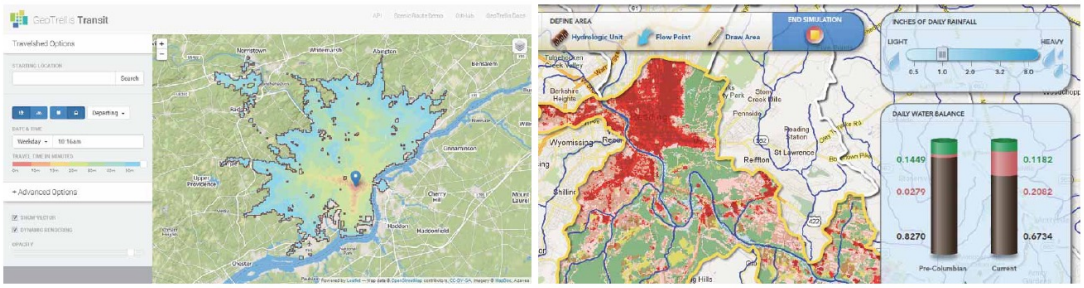


图 4.14　Geotrellis 数据处理流程

目前基于 Geotrellis 实现的成功案例已经有很多，例如基于 William Penn 基金支持的可持续交通网络应用，以及为 Stroud 水研究中心研发的流域分析模型

等[98, 99]，如图 4.15所示。



(a) 可持续交通应用　　(b) 流域分析应用

图 4.15　基于 Geotrellis 实现的案例

4.3.3　算法流程步骤

本文算法基于 Geotrellis 提供的 API 接口，分为 IO 以及处理两大部分，具体描述如下：

第一步：*数据存储*，把遥感影像数据集分布式存储到 HDFS 中。

第二步：*初始化* Spark 环境，设置主节点 IP、驱动器节点最大内存、执行器节点最大内存等。

第三步：*数据读入* Spark，利用 Geotrellis 提供的 Spark 影像 IO 函数 *hadoopMultibandGeoTiffRDD* 将遥感影像读入 Spark 的 RDD 中。

其底层实际采用新版的 Hadoop API 接口“newAPIHadoopFile”，其一共四个输入参数，第一个参数为影像数据集的文件路径或者目录路径，当为目录路径时会自动搜索该目录下以及所有子目录下的影像文件；第二个参数为格式累，代表输入的格式，Geotrellis 一共实现了四种格式，分别为 GeotiffInputFormat（单波段 Geotiff 影像）、TemporalGeoTiffInputFormat（带时间戳的单波段 Geotiff 影像）、MultibandGeoTiffInputFormat（多波段 Geotiff 影像）以及 TemporalMultibandGeoTiffInputFormat（带时间戳的多波段 Geotiff 影像）。第三个参数为存储影像 RDD 的 Key 值类型，在 Geotrellis 中把影像的地理范围作为 RDD 的 Key。其有两种类型，一种为 ProjectedExtent（基于投影坐标的地理范围），另一种为 TemporalMultibandGeoTiffInputFormat（带时间戳的投影坐标范围）。最后一个参数为 RDD 的 Value 类型，存储的是在 Spark 数据切分后每一块的影像数据，其也有两种类型 Tile（单波段影像块）以及 MultibandTile（多波段影像块）。

第四步：*设置切分准则*，包括瓦片大小，以及瓦片投影坐标系。并获取影像的元数据信息。

第五步：影像切分成瓦片 RDD，根据切分准则，把影像分布式存储成多个 RDD，每个 RDD 的 Key 为瓦片的行号以及列号组成的元组，在 Geotrellis 中对应的类型为 SpatialKey；Vaule 对应为该瓦片的像素信息，在 Geotrellis 中以 Tile（单波段）、MultibandTile（多波段）类型保存。如图 4.16所示，当一个瓦片跨越多个影像时，Geotrellis 会自动从多张影像中提取数据来组建瓦片的 RDD。

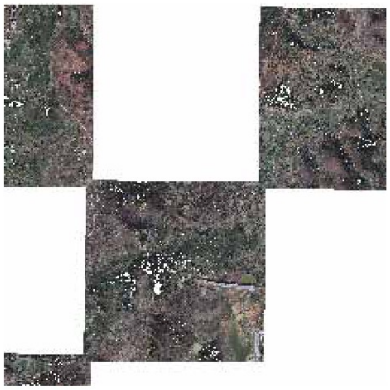


图 4.16　瓦片跨越多张影像情况

第六步：投影变换，如果原始影像投影跟瓦片投影不一致，将原始影像投影变化至瓦片投影坐标系下。

第七步：建立金字塔，依次将每层金字塔的瓦片 RDD 渲染成“jpg”或者“png”格式的图片然后输出到 HDFS 文件系统中。

4.3.4　实验与分析

实验采用 10 台机器组成的 Spark 集群进行测试，Spark 版本号为 1.6.1；Hadoop HDFS 版本为 2.6.0，HDFS 总容量 14TB；Geotrellis 版本为 2.10-0.10.0。集群内节点分别有两种类型，每个类型详细信息见表 4.5。

表 4.5　Spark 集群信息

CPU	总核数	内存（GByte）	操作系统	机器个数
Intel(R) Core(TM) i5-4570 @ 3.20GHz	4	14.4	Ubuntu	5
Intel(R) Xeon(R) E5-2650 @ 2.30GHz	40	256	Ubuntu	5

实验数据采用湖南省某县城的高分辨率遥感影像标准分幅数据集，其中数据集中的每个影像分辨率都为 0.2m，以像素为单位长宽都为 5201 × 5201，像素类型为 Byte，投影坐标系都为中国大地 2000-高斯克吕格投影坐标，取其中 10GB 数据分别组建 1 ～ 10GB 不同规模大小的 10 个数据集进行测试，通过利用不同数量

的相同大小的影像构建不同规模的测试数据集可以减少因数据的原因给实验结果带来的误差。图 4.17为为算法在表 4.5所示的 Spark 集群环境下，对上述所述的 10 个不同数据规模数据集进行测试得到性能曲线。横坐标为数据集的大小，纵坐标为算法耗时，可以看到在集群中每个节点内存都足够充足的情况下随着数据量的增大，算法耗时大致能够保持一个线性递增的变化，算法性能可以一直保持在较高的水平。

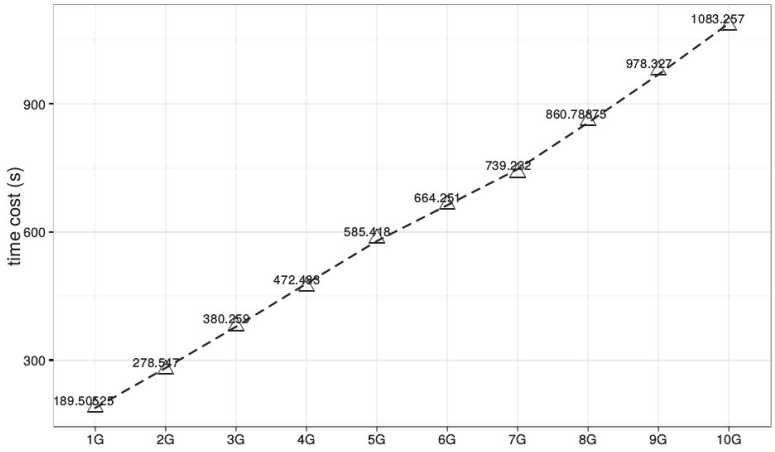


图 4.17　不同数据规模下算法性能

4.4　本章小结

本章主要阐述的是分幅影像数据集快速瓦片化技术，其中简单介绍了分幅影像的概念，以及分幅影像相对于其他影像瓦片化的难点，而后针对这两点提出了分幅影像快速瓦片化的实现方案。基于分幅影像快速瓦片化，本文提出了两大算法：1、影像数据并行分幅输出算法 2、分幅影像快速瓦片化算法。分幅输出算法采取 MPI 多进程的方式，分幅影像快速瓦片化方法分别探讨了传统 MPI 以及 Spark 分布式两种实现方式。其过程主要是首先采用并行分幅输出算法基于接图表对原始影像进行分幅，而后再用分幅影像快速瓦片化算法将分幅结果进行瓦片化。

并行分幅输出算法采用的是车轮法进行各进程任务分配，通过利用每个影像的地理范围 MBR 来建立 R 树索引，实现每个进程可以快速获取其对应数据，通过计算接图表与对应数据地理范围相交的部分，而后取出相交的部分最后拼合成完整的分幅结果，分幅过程中各进程互相独立，并行程度高，算法具有较强的稳定性与扩展性。

基于 MPI 的的分幅影像快速瓦片化方面主要实现了分幅数据的并行镶嵌以及瓦片的并行融合来解决单张瓦片可能跨越多个分幅数据的情况。基于分布式的分幅影像快速瓦片化实现方法主要是利用 Spark 分布式计算平台，采用开源项目

Geotrellis 提供的 API 接口，通过将所有影像读取成一个完整的 Spark RDD，然后对这个完整的 RDD 进行切片和渲染，类似于在内存中完成了影像数据集的拼合以及无效值的去除工作。

后期研究主要是在 Spark 平台上引入 Yarn 或者 Mesos 等资源调度系统，当集群中某些节点拖累性能时，可以实现算法在指定的高性能节点上运行，不会因为个别节点而影响到整个集群的性能。

第五章　总结与展望

随着航天技术的发展，传感器精度的不断提高，遥感影像的时间与空间分辨率都得到了大幅提高，这也导致了单幅遥感影像数据量以及分幅影像数据集的数据量大大增加，传统算法以及现有瓦片化工具在处理单幅大影像以及批处理大量影像数据集上性能都完全不能满足应用需求。随着当今高性能计算以及大数据技术的蓬勃发展，如何结合高性能计算技术以及瓦片化技术，利用并行技术来提高传统影像切片算法的性能是当今 WebGIS 可视化中一个比较热的研究方向，本文主要基于 MPI 在共享外存的集群以及基于 Spark 在分布式环境上分别设计并实现了相应的瓦片切分算法，通过与传统算法进行性能对比，进一步阐述了本文算法的优势以及当前存在的不足。

5.1　主要研究成果

本文针对高性能影像数据瓦片化关键技术从单幅大规模影像以及大规模分幅影像数据集上进行了多个方面的研究，主要研究成果以及创新点总结为，主要有如下两点：

第一点，单幅大规模影像快速瓦片化技术

针对单幅大规模影像数据量大，IO 耗时严重，计算复杂度高的特点，本文提出了一套地理空间数据并行 IO 的处理机制，一种数据并行重采样方法，以及一种基于 BSQ 结构的存储方法，通过整合上述技术实现了一种基于 MPI/OpenMP 的多线程与多进程混合并行模式的影像金字塔并行构建算法，该方法采用多进程与多线程结合的方式，当数据量过大而用户基于 MPI 为算法分配的进程总数过少时，算法会自动利用 OpenMP 多线程技术在每个进程下细分多个线程，并通过计算得到每个进程下合理的细分线程总数，不仅解决了在处理大规模数据时常见的类型溢出现象而且充分发挥了集群的性能极大提高了算法的执行效率，为 Web 浏览器在线浏览地图时 Mapnik 能够进行实时切片提供了保证，以 115.9*GB* 测试数据为例，实验表明本文提出的并行构建金字塔算法可以达到 GDAL 金字塔构建算法的 1893.31 倍，ArcGIS Server 切片工具的 85.3 倍，赫高进等人提出的基于 MPI 的金字塔并行构建算法的 10.28 倍，并且随着数据量的继续增大，以及处理影像的波段数持续增加，本文提出的并行构建金字塔算法性能优势会更加明显。

针对预处理的情况，本文改进了传统切片流程中先建影像金字塔，而后基于影像金字塔进行逐层切片形成瓦片金字塔的步骤，提出一种并行切片方法，采用行划分的方式多进程并行切出影像最高层瓦片，而后基于最高层通过相邻四瓦片合成下一层瓦片的方式，逐步合成底层金字塔的瓦片，最终生成瓦片金字塔。这

种方式可以省去预先构建影像金字塔的时间，通过并行切出最高层瓦片后，底层瓦片通过上层瓦片逐级合成的过程中，各进程可以快速连续的读写，极大提高了IO 效率，有效地减少了瓦片切片耗时。通过本文的测试数据进行实验，与传统ArcGIS 切片方法进行对比，本文算法在切单层瓦片上可以达到 ArcGIS 的 9 倍以上，如果采用瓦片合成的方法来进行瓦片金字塔构建，其速度可以继续提高 5 倍左右。

第二点，大规模分幅影像数据集的快速瓦片化技术

针对国土部门非常常见的大规模分幅影像数据集快速发布的需求，大规模分幅数据集针对与单幅大规模数据量影像其区别在于数据集内影像数量多，但是每幅影像的数据量不大，并且相邻图幅影像有重叠区域，针对分幅影像数据集的特点本文分别提出了影像数据集并行分幅算法以及分幅影像快速瓦片化算法，并且实现了全部流程的自动化处理。其中并行分幅算法主要用于将影像数据集基于接图表快速生成标准的分幅影像数据集，而分幅影像快速瓦片化算法主要是在上一步分幅影像数据集的基础上，进行切片形成分幅影像数据集的瓦片集合，其中主要难点在于相邻图幅数据间相交区域的无效值处理，所以针对分幅影像数据集瓦片化问题本文通过利用现有技术提出了基于传统 MPI 的实现方法以及基于 Spark 的分布式瓦片化方法，基于 MPI 实现的分幅数据集瓦片方法，本文主要实现了影像数据并行镶嵌算法以及分幅瓦片并行融合算法，通过这两个算法来处理相邻图幅影像重叠区域间可能存在无效值的问题，其中影像数据并行镶嵌算法是将所有分幅影像并行拼接成一个完整的影像，在拼接的过程处理掉无效值问题，然后再对该拼合后的影像进行瓦片化；而分幅瓦片并行融合算法主要是先分别对每一个分幅影像进行并行独立切片，得到每个分幅影像的瓦片数据集，因为在瓦片坐标系下，相同行列号的瓦片代表的是相同区域，因此可以通过将所有行列号相同的瓦片进行融合来去除瓦片中存在的无效值部分。

基于 Spark 的瓦片化实现方法本文主要采用 Geotrellis 开源项目提供的接口，通过将其提供的串行瓦片化接口在 Spark 上进行分布式扩展，进而得到了本文所提出的分布式瓦片化算法，针对与 Geotrellis 原始提供的接口本文不仅将其进行了分布式扩展而且还在分布式的基础上进行了多波段扩展，使其具有处理多波段数据的能力。通过利用不同数据量的数据集对算法性能进行了测试，实测数据表明在内存充足的情况下，随着数据量的增大，本文算法基本可以一直稳定保持在每 *GB* 数据切瓦片速度在 189 秒左右。

本文并行分幅算法已经应用到湖南省国土测绘部门，根据应用单位的反馈，在处理 8*TB* 影像数据分幅的工作上本文算法将其传统工作流程 3 个月的工作量提升到 3 天完成，极大提高了数据分幅处理的工作效率。

5.2 下一步研究工作

本文中有些工作仍然存在一些不足，前期的影像并行构建金字塔算法，以及分幅影像瓦片化算法的实现环境都是在共享存储的高性能集群上，这种方法其性能可扩展性较差，最终性能会受限于磁盘 IO 的速度，下一步工作主要是尝试能否将算法的存储环境由集中式存储转换到分布式存储。

在基于 Spark 实现的分布式海量影像瓦片化技术中，借助了开源项目 Geotrellis 提供的 API 接口，因此对于其是如何实现将遥感影像转换为 Spark RDD 的原理，不是特别明了，下一步工作中，这是必须解决的问题，只有在吸收其实现技术的基础上才能做相应的优化和改造，遥感影像的分布式 IO 也是影像分布式处理的一个难点。由于 Geotrellis 开源项目还在不断发展，目前其自身还是有许多问题，比如其不支持读取数据量大于 2*GB* 的单个影像，这就限制了其处理单个大规模影像，只能处理海量小规模影像，这是下一步工作中一个比较迫切需要的问题。

由于瓦片数据具有单个文件数据量小，但是数量多的特点，因此如何利用 GFS 以及 TFS 等小文件管理系统来加快瓦片文件的存储以及检索效率也是后续工作应该考虑的部分。

本文主要探讨的是栅格数据高性能瓦片化技术，但是在地图数据中不仅有栅格数据而且还有矢量数据，矢量数据快速绘制以及可视化也是一个重要的研究方向，因此下一步工作会考虑矢量数据并行瓦片化技术的相关研究。

致　谢

丙申年秋，星城九月，天朗气清，丹桂飘香，秋雁齐飞；授业即毕，离日且近，伏案奋笔，字斟句酌，思辨慎取，尽付三年所学以成此文；虽文拙理浅，然兢业律己之心，不敢懈怠，此情悠悠，天地可鉴。书山有路，学海无涯，师恩浩荡，虽结草陨首难以报之万一，是故以情造文，铭而敬谢。

桃花又谢春红，忆吾以弱冠之龄负笈于科大，迩来春秋有三；蹉跎岁月，荏苒时光，户牖之外，草长莺飞，景致阙如昨日；日月不淹，冬夏往替，初入科大，校门之巍峨，列兵之庄严，历历在目，敬畏之心溢于言表。清晨号角吹寒，雄声如虎啸，每日修习，如醍醐灌顶，心激荡非凡。煌煌科大，军中清华，国之重器，西挹岳麓之紫气，北来湘水之浩渺，学风鼎盛，俊才星罗，将星闪耀，荟萃四方名士。创新不辍，启迪华夏，弦歌继响，薪火相传，闻名遐迩，厥功至伟，广厦栋宇，气势恢弘，楼台曲水，嘉树繁阴。诚朴雄伟，巍巍大学之道尽在其中，励学恭行，赫赫诲人之德皆出其里，金声震于华夏，威名扬于神州，热血男儿，无不争先以此为身效祖国、成功立业之基石，余亦不例外。

吾生性驽钝，惟勤力自勉，幸逢恩师不以鄙陋，扶泥淖以大道，登幽谷以绝顶，得以攀科大之桂，惶恐入榜。少年无知，懵懂离家，学子有幸，忝列师门，上天眷顾，从不敢忘；驽马十驾，功在不舍，立雪囊萤，闻鸡看剑；科研之路，且理且实，望书卷洋洋，恨不能遍览所藏，听名师讲堂，叹不能窥尽所知一二，惟此平生，披光履尘，不教此剑，负此乾坤。

恩师李公，单名军字。先生亦父亦师授吾地理信息之道，图像处理之法；先生温蔼和风，平易严谨，博通古今，著满中外；虽年逾不惑，但不倦如故，潜心治学，心无旁骛，研欲深广。吾初学无法，学浅识薄，倘有管窥蠡测，坐井观天之嫌，先生不以愚钝，苦心孤诣，鱼渔双授，循循善导，切切叮咛，开我茅塞，吾屡屡如梦初醒；犹忆当年，乍入门墙，时先生漂洋过海，赴加国访问，仍每每惦念吾等日用之难，生活学习之需，关怀备至，资助颇厚。求学之路多艰辛，每遇彷徨失意，心绪不宁，欲知难而退之际，先生良言开导，宽言相慰，激以励志，敛欲克己，厚积薄发，终有高翔九天，清鸣云中之时；桃李不言，下自成蹊，细雨之情，润物无声而草木深秀，每念于此，倍感其恩。先生如明灯，导吾以正途，言传身教，使吾得以在学术殿堂大展身手，敢于勇跃 SCI 之门。古语有云：“艨艟巨舰，非舵桨导引之助不能乘风破浪；北冥鲲鹏，非长风托举之力不能垂翼九天”，吾今之小成，尽皆恩师教育指引之功，于今毕业，先生荐吾以工作，广吾人脉，助吾

仕途，恩师之慧，不一而足，受益匪浅，不能言尽，虽只字片语，感谢无疆。

吾师陈荦，颇类古之良师，治学严谨，眼界高远，神思敏捷，善规划决断，虽每日政务繁杂，然轻重缓急，井然有序，丝毫不差；虽少有空暇，然则为政闲时，仍不忘拔冗垂询诸弟子之学业，常以己之饭卡资吾等食资，先生之德，大为拜服。先生处事，事无巨细，未动先谋，心无旁骛，动若脱兔，律己日严，宽则待人，此亦为吾日后操行之典范。曾记当时，初入师门，先生即委以重任，是以勤学自勉，思唯速决，毋负所望，然好事多磨，欲速不达；先生不厌其烦，每必躬亲，探讨琢磨，同至进退，常至子夜；至于遣词造句，缀词成文，先生亦反复锤炼，不辞琐碎，字斟句酌，数易其稿，但求文顺气通，简练明了，观其批阅，独具匠心，深自叹服。与先生相处，乃明聚沙成塔，滴水穿石之理，所获处世为人，修身齐家之法，吾必以此为则，行之以诚；今离日在即，谆谆教诲，不复再闻，然授业之恩，不敢稍忘，唯期终有所成，来日后报。

先生吴秋云，导吾于狭路，示吾以通途，乃吾学术之引路人，若无先生吾有眼如盲。课题之确定，文章之撰写，研究方法，及整体结构设计，皆得先生悉心教导。先生睿智幽默，儒雅谦和，与公相处，其乐融融，此诚人生一幸事。翩翩鸿儒，微言大义，不偏不倚，劲竹强风，上善若水，夫唯不争，澹淡名利，浆饮陋巷。谢先生荫护栽培，非先生之功，吾不能得安身立命之本，不能明人生得失之幽。

熊伟先生，望之若严，亲之和蔼，才华横溢，深耕数据库，蔚为大家；授学之道，探幽索微，深入浅出，妙趣横生，学生无不如浴春风，与公探讨学问，其乐无穷，此诚良师亦友也。

吴烨师兄，亦师亦兄，飞扬脱脱，技术达人，但有所问，必可得解，日后必为DBRG之中坚；想当时，与君共调代码至子夜，虽任务繁重，但从不觉辛劳，每日充实非常，今之回望，仍念之不已。

先生景宁，年少之时，背井离乡，远渡重洋，赴美留学，历尽艰难，待功成名就，毅然回国，景行余风叫人敬之无穷。先生重德惜才，身体力行，懿德雅亮，宽广贤明，门生桃李天下，为当世之名师；为建HIGIS平台，扩学术之影响，先生倾一己之力，斡旋中外，经纬天地，协同僚，带弟子，奔走呼号，殚精竭虑，如此今之方有我辈钻研习艺之良器佳境，盖后人乘凉亦不忘前人栽树也。先生伯乐也，遥忆当年，学士即毕，幸蒙赏识，喜托龙门，嘘寒问暖，寄于厚望，吾尚无剖符丹书之功，反受此殊遇，感激涕零，惟奋起而后进，以报先生知遇之恩。

师门济济，手足侪辈，亦有师钟志农、伍江江、陈浩、陈宏盛、郭裕兰、杜春、宋宝泉等无私教导，言传身教，淳淳教诲如春风拂面，雨露滋润似久旱甘霖；更欲备述各师长哺育之恩，然恩长笔短，不能一一言尽，在此唯有再次俯身拜谢。

师门之同窗少年者，皆四海英才，五湖栋梁，得此广助，幸何如之。三载相交，情义笃深，亲同手足；研习间，指点江山，激昂文字，谈人生理想，评教育得失，叹历史兴替，争学问于庭前，访风景于崇阿，情意暖暖，其乐无穷。博士杨公岸然，谓之奇才，无所不通，无所不晓，琴棋书画皆造诣颇深；君出吾于幽谷，迁吾于乔木，博吾孤陋，厚吾基石，习谈之间，屡获金石朱玉之言，如醍醐灌顶，与君辩疑，妙趣横生，别有洞天；于青春之时与公相识，幸甚至哉，待功成之时，一举十觞，与君畅饮。师兄郝队长，少年英气，章灼神采，志存高远，乃吾 MPI 并行编程之领路人，金字塔研究之先驱；初入此门，茫然前路，常与公探讨于前，互为解难，切磋学艺，畅谈人生；公于吾先辞，惟愿他日再会，再度把酒言欢。师弟小超，古道热肠，但有所托，必施援手，勤奋好学，自立自强，来日必有所成；愿君再接再厉，勇创辉煌。同级者郭宁，温和敦厚，忠正纯良，学术有成，日后必为学术大拿。马梦宇，溯出衡水，学霸气韵，无人可比，胸襟绰约，慷慨大方，与公相谈，常有新得，多有广益，在此谨表谢枕。伍送兵，相交莫逆，号永州十霸，少年义气时，挥斥方遒，金戈铁马，赤血黄沙，王侯将相，其志无匹。王晨，文艺才子，不拘小节，舞蹈乐理，样样皆通，吾羡之无穷。同舍王崇辉、楼康威、陆旭三人，砥砺三载，情深谊笃，今入江湖，青山不老，绿水长存，不诉离殇。其余同窗者，益予者多矣，窃愿与诸位功业共勉。人生苦短，得遇诸公等金兰之交，于今将相隔千里，情深意长，一语难表，惟寄此情于心，再起征程。

“慈母手中线，游子身上衣，临行密密缝，意恐迟迟归”，椿庭严，萱堂慈，严正行，慈养心。襁褓之时，体弱多病，如草上珠，朝不保夕，双亲守候无稍息，咽苦不颦，个中艰辛，实非言语所能尽述；而后违远慈思，拜辞桑梓，千里孤学，闻道远行，慈母手线，怜儿夜寒，儿在远游，亲念他乡；时至今辰，业已即成，然茕茕孑立，双亲尤系，夙兴夜寐，时忧仕途，常念婚姻，未得一日清福；吾为独子，不能久伴，亦难常叙天伦，享儿孙绕膝之福，本欲远渡重洋，再行深造，恐子欲养而亲不待，难报双亲之恩情，不禁潸然；鱼传尺素，报答椿首，书不尽意，情更拳拳，不图天降大任，只求福满之家，以报滋养恩，皇天后土，实所共鉴。

巍巍岳麓，浩浩湘水，筵席终散，天各一方；歧路沾巾，终儿女态，看明朝蓬勃，登高以歌，直上青天摘明月！收笔之际，引《三国》诗一首，畅舒胸臆：

英雄露颖在今朝，一试矛兮一试刀。

初出便将威力展，三分好把姓名标。

石中金於于国防科大

参考文献

[1] CHRISTOPHER B. J and J.MARK WA. Map generalization in the Web age [J]. International Journal of Geographical Information Science, 20 Feb 2007.

[2] 胡祥云，胡祖志，钟宏伟，等. 科学可视化及其在地学中的应用 [J]. 工程地球物理学报，2004，1(4):358-362.

[3] Turner N, Fernandez C, Lessin M. Image tile server: US, US8244770[P]. 2012. OGC.OpenGIS Web processing service.OGC implementation specification,open geospatial consortium. 2007.

[4] OGC. Open GIS Web processing service. OGC implementation specification, open geospatial consortium. 2007.

[5] 刘晨鑫. WebGIS 设计原理与实现研究 [J]. 通讯世界，2014(21):239-239.

[6] 苗聪. 基于 WebGIS 的公交信息服务系统设计与实现 [D]. 东南大学，2006.

[7] Bober M. Method for efficient coding of shape descriptor parameters: U.S. Patent 7,447,373[P]. 2008-11-4.

[8] Iodice D M, Bell D M. Remote image exploitation display system and method: U.S. Patent 5,613,051[P]. 1997-3-18.

[9] Migdal C J, Foran J L, Jones M T, et al. Method and system for providing exture using a selected portion of a texture map: U.S. Patent 5,760,783[P]. 1998-6-2.

[10] KanedaK.Imageprocessingdeviceandmethodandmemorymedium:U.S.Patent ,714,682[P]. 2004-3-30.

[11] Barclay T, Gray J, Slutz D. Microsoft TerraServer: a spatial data ware¬house[C]//ACM SIGMOD Record. ACM, 2000, 29(2):307-318.

[12] Rasmussen J E, Rasmussen L E, Taylor B S, et al. Digital mapping system: U.S. Patent 7,158,878[P]. 2007-1-2.

[13] McAvoy J, Wendland L, Nguyen B, et al. Customized wall map printing system: U.S. Patent 7,274,378[P]. 2007-9-25.

[14] Brichford C, Rowe E R W, Lynch K, et al. Rendering hypertext markup language content: U.S. Patent 8,627,216[P]. 2014-1-7.

[15] Jing F, Zhang L, Li M J, et al. User interface for viewing clusters of images: U.S. Patent 7,644,373[P]. 2010-1-5.

[16] Jing F,Zhang L,Ma W Y.Identifyingsightforalocation:U.S.Patent7,707,208[P]. 2010-4-27.

[17] Jing F, Zhang L, Ma W Y. User interface for displaying images of sights: U.S. Patent 7,657,504[P]. 2010-2-2.

[18] Lafon S. Efficient rendering of panoramic images, and applications thereof: U.S. Patent7,843,451[P].2010-11-30.

[19] ZhuJ,FilipD,VincentL.Three-dimensionalannotationsforstreetviewdata:U.S. Patent8,072,448[P].2011-12-6.

[20] Dayan T, Ross M. Sharing geographical information between users: U.S. Patent 9,436,666[P].2016-9-6.

[21] Jones M T, Rohif J, McClendon B A. Dynamic view-based data layer in a geo¬graphicinformationsystem:U.S.Patent8,350,849[P].2013-1-8.

[22] SurazakovAB,AizenVB.Estimatingvolumechangeofmountainglaciersusing SRTMandmap-basedtopographicdata[J].IEEETransactionsonGeoscienceand RemoteSensing,2006,44(10):2991.

[23] Thorvaldsdóttir H, Robinson J T, Mesirov J P. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration[J]. Brief¬ingsinbioinformatics,2013,14(2):178-192.

[24] Maiellaro N, Varasano A. Multimedia Interactive Map for CH Presentation[C]//Euro-MediterraneanConference.SpringerInternationalPublish¬ing,2016:178-190.

[25] IriberriGutiérrezC.Predictiveanalysisforcachegenerationinquadtreebasedgeo visualizations[D].,2015.

[26] FollumML,TavakolyAA,NiemannJD,etal.AutoRAPID:AModelforPrompt Streamflow Estimation and Flood Inundation Mapping over Regional to Conti¬nentalExtents[J].JAWRAJournaloftheAmericanWaterResourcesAssociation, 2016.

[27] LeeKW,ChoiJY. ImplementationofTileSearchingandIndexingManagement Algorithms for Mobile GIS Performance Enhancement[J]. Journal of The Korea InternetofThingsSociety,2015,1(1):11-19.

[28] LangevinS,JonkerD,BirkK,etal.GlobaltoLocalPatternofLifeAnalysiswith Tile-BasedVisualAnalytics[J].

[29] Behm A. Efficient Retrieval and Ranking of Maps for Geographic Queries[J]. 2016.

[30] Liu Z, Jiang B, Heer J. imMens: Realtime Visual Querying of Big Data[C]//Computer Graphics Forum. Blackwell Publishing Ltd, 2013, 32(3pt4):

[31] Haklay M, Weber P. Openstreetmap: User-generated street maps[J]. IEEE Perva¬siveComputing,2008,7(4):12-18.

[32] BieleckiW,SkotnickiP. TileMergingTechniquetoGenerateValidTiledCodeby Means of the Transitive Closure of a Dependence Graph[C]//International Multi-Conference on Advanced Computer Systems. Springer International Publishing, 2016:315-327.

[33] GuevaraEscobedoJ.Embebedwaveletimagereconstructioninparallelcomputa¬tionhardware[D].U

[34] Schwambach V, Cleyet-Merle S, Issard A, et al. Image tiling for embedded ap¬plicationswithnon-linearconstraints[C]//DesignandArchitecturesforSignaland ImageProcessing(DASIP),2015Conferenceon.IEEE,2015:1-8.

[35] 曹冬冬, 赵永华, 赵莲. 基于瓦片算法的并行 QR 分解及其实现 [J]. 科研信息化技术与应用,2016,7(2).

[36] 陈欢. 地理矢量数据快速可视化技术研究 [D]. 国防科学技术大学,2013.

[37] 刘晰, 张轶, 杨军, 等. 利用并行技术的海量数据瓦片快速构建 [J]. 测绘科学, 2016, 41(1):144-150.

[38] 周郑芳, 易磊, 王惠, 等. 遥感影像并行服务系统设计与实现 [J]. 测绘科学技术学报,2015(1):42-46.

[39] 杨子煜, 严明, 赵鹏. 基于多核阵列体系结构的嵌套循环并行优化 [J]. 计算机工程与科学,2009,31(a01):125-128.

[40] 刘义. 大规模空间数据的高性能查询处理关键技术研究 [D]. 国防科学技术大学,2013.

[41] 陈小潘, 渠涧涛, 赵亚萌, 等. 基于双缓冲队列的海量地形数据并行处理方法 [J]. 郑州大学学报 (工学版),2016,37(3).

[42] 殷君茹, 侯瑞霞, 唐小明, 等. 基于瓦片金字塔模型的海量空间数据快速分发方法 [J]. 吉林大学学报: 理学版,2015,53(6):1269-1274.

[43] 邓雪清. 栅格型空间数据服务体系结构与算法研究 [D]. 中国人民解放军信息工程大学,2003.

[44] 杜波. 基于 MapReduce 的栅格地图切片系统 [D]. 西安电子科技大学,2014.

[45] 张锦林. 海量遥感影像瓦片金字塔并行构建工具的设计与实现 [D]. 华中科技大学,2014.

[46] 刘坡, 龚建华. 大规模遥感影像全球金字塔并行构建方法 [J]. 武汉大学学报 (信息科学版),2016,41(1):117-122.

[47] 刘振东. 大规模三维地形高效可视化方法研究 [D]. 中国测绘科学研究院, 2015.

[48] 原发杰. 一种新的海量遥感瓦片影像数据存储检索策略 [D]. 电子科技大学, 2013.

[49] 李晶. 三维数字地球构建关键技术研究 [D]. 电子科技大学,2012.

[50] 李德仁, 朱欣焰, 龚健雅. 从数字地图到空间信息网格——空间信息多级网格理论思考 [C]. 中国地理信息系统协会年会，2003:642-650.

[51] 李德仁, 童庆禧, 李荣兴, 等. 高分辨率对地观测的若干前沿科学问题 [J]. 中国科学: 地球科学，2012,42 (6) :805-813.

[52] 许虎, 聂云峰, 舒坚. 基于中间件的瓦片地图服务设计与实现, 地球信息科学学报 [J], 2010,12 (4):562-567.

[53] 霍亮, 杨耀东, 刘小勇等. 瓦片金字塔模型技术的研究与实践 [J]. 测绘科学,2012,37 (6):144-146.

[54] 关雷, 刘蕾, 郭慧宇. 基于瓦片技术的高分辨率遥感影像快速访问技术在测绘生产中的应用研究 [J]. 测绘与空间地理信息, 2016, 39 (2):78-79.

[55] 刘世永, 吴秋云, 陈荦, 等. 基于高层级地图瓦片的低层级瓦片并行合成技术 [J]. 地理信息世界, 2015, 22 (6):51-55.

[56] 李长春, 蔡伯根, 上官伟, 等. 基于 Web 墨卡托投影的地图算法研究与实现 [J]. 计算机应用研究, 2012, 29 (12):4793-4796.

[57] 刘镇. 遥感影像瓦片金字塔模型 [J]. 科技创新导报, 2008 (6):199-200.

[58] Yu, L. Accelerating image pyramid on gpus [J]. *Advanced Materials Research, 2014, vols.926-930*, May, pp. 403–406.

[59] Liu, P., Gong, J. Parallel construction of global pyramid for large remote sensing images [J]. *Geomatics & Information Science of Wuhan University, 2016,* **41**(5), Jan, pp. 117–122.

[60] Wei, X., Lu, X., Sun, H. Fast View of Mass Remote Sensing Images Based-on Image Pyramid. Intelligent Networks and Intelligent Systems [C]// *2008. ICINIS '08. First International Conference on IEEE*., 2008, pp. 461–464.

[61] Deng, X. Q. Research on service architecture and algorithms for grid spatial data [J]. *Acta Geodaetica Et Cartographic Sinica., 2003,* **32**(4), pp. 361–362.

[62] Adelson, E. H., Anderson, C. H., Bergen, J. R., Burt, P. J., Ogden, J. M. 'Pyramid methods in image processing [J]. *Rca Engineer*. 1983.

[63] Kang, J. F., Zhen-Hong, D. U., Liu, R. Y., and Fang, L. Parallel image resample algorithm based on gpu for land remote sensing data management [J]. *Journal of Zhejiang University., 2011,* **38**(6), pp. 695-700.

[64] Cheng, C. Q., Zhang, E. D., and Wan, Y. W, et al. Research on remote sensing image subdivision pyramid [J]. *Geography and Geo-Information Science., 2010,* **26**(1), pp. 19-23.

[65] Yang, J., Zhang, J. Parallel performance of typical algorithms in remote sensing-based mapping on a multi-core computer [J]. *Photogrammetric Engineering & Remote Sensing., 2015,* **81**(5), pp. 373-385.

[66] Yang, J., Zhang, J., and Huang, G. A parallel computing paradigm for pan-sharpening algorithms of remotely sensed images on a multi-core computer [J]. *Remote Sensing., 2014,* **6**(7), pp. 6039-6063.

[67] Wang, L., Ma, Y., Zomaya, A. Y., Ranjan, R., and Chen, D. A parallel file system with application-aware data layout policies for massive remote sensing image processing in digital earth [C]// *IEEE Transactions on Parallel & Distributed Systems., 2015,* **26**(6), pp. 1497-1508.

[68] Chu, B., Jiang, D. L., and Cheng, B. Large scale mosaic using parallel computing for remote sensed images [J]. *Applied Mechanics & Materials., 2014, vols. 556-562*, pp. 4746-4749.

[69] Zheng, G. Research and implementation of fast generation of massive remote sensing image in Pyramid [D]. *East China Normal University*. 2012.

[70] 刘义, 陈荦, 景宁, 等. 利用 MapReduce 进行批量遥感影像瓦片金字塔构建 [J]. 武汉大学学报: 信息科学版, 2013, 38 (3):278-282.

[71] The Message Passing Interface (MPI) standard [EB/OL]. http://www.mcs.anl.gov/research/projects/mpi/.

[72] 赫高进, 陈荦, 熊伟, 等. 基于 MPI 的大规模遥感影像金字塔并行构建方法 [J]. 地球信息科学学报, 2015, 17 (5):515-522.

[73] ArcGIS, E. S. R. I., 2013. 10.1 Desktop help [DB/OL]. ESRI, Redlands, CA. ArcGIS, 10.

[74] Adobe Developers Association. 26.TIFF Revision 6.0 [DB/OL]. Mountain View, Cal.: Adobe Systems. 1992.

[75] Qin, C. Z., Zhan, L. J., Zhu A. How to apply the Geospatial Data Abstraction Library (GDAL) properly to parallel geospatial raster I/O? [J]. *Transactions in GIS., 2014,* **18**(6), pp. 950-957.

[76] Liu, O. Parallel access methods for geographic raster data [J]. *Computer Science*. 2012.

[77] OpenMP Architecture Review Board [EB/OL]. https://zh.m.wikipedia.org/wiki/OpenMP.

[78] 王慧, 申家双, 陈冬阳, 邓雪清. 一种高性能的大区域遥感影像管理模型 [J]. 2006, 26 (3):71-74

[79] 殷福忠, 孙立民. 基于瓦片金字塔技术的地图发布平台开发研究 [J]. 2010, 33 (5):16-20

[80] 地图分幅 [EB/OL]. http://baike.baidu.com/link?url=M_iN6Oh2hX2i1 29ubZG-GcdGtldhPHyDKBCMS2RUyaNRebZMgGKBGfbigMXw8aWqKdpqwzrnGhw69rONyvMjnWq.

[81] 地图分幅和编号 [EB/OL]. http://blog.sina.com.cn/s/blog_508efd540100bt6k.html.

[82] IBM General Parallel File System [EB/OL]. https://en.wikipedia.org/wiki/IBM_General_Parallel_File_System.

[83] Nature. Big Data [EB/OL]. 2008, http://www.nature.com/news/specials/bigdata/index.html.

[84] White T. Hadoop: the definitive guide [J]. O'reilly Media Inc Gravenstein Highway North, 2010, 215(11): 1-4.

[85] Apache. Hadoop [EB/OL]. http://hadoop.apache.org.

[86] Wiki H. Apache Hadoop [EB]. http://wiki.apache.org/hadoop/Frontpage.

[87] Shvachko, K., Kuang H., Radia S, et al. The Hadoop Distributed File System [C]// IEEE, Symposium on MASS Storage Systems and Technologies. IEEE Computer Society, 2010:1-10.

[88] Apache. HDFS Architecture Guide [EB]. http://wiki.apache.org/hadoop/Frontpage.

[89] Oracle. MapReduce [EB/OL]. https://blogs.oracle.com/datawarehousing/entry/in-database_map-reduce.

[90] 龙少杭. 基于 Storm 的实时大数据分析系统研究与实现 [D]. 上海交通大学, 2015.

[91] Zaharia, M., Chowdhury M., Franklin M. J, et al. Spark:cluster computing with working sets [J]. 2010:10-10.

[92] 胡俊，胡贤德，程家兴. 基于 Spark 的大数据混合并行计算模型 [J]. 计算机系统应用. 2015 (4):214-218.

[93] Apache. spark [EB/OL]. http://spark.apache.org.

[94] Apache. spark [EB/OL]. http://en.wikipedia.org/wiki/Apache_ Spark.

[95] Eclipse. Introducing GeoTrellis [EB/OL]. http://www.eclipse.org/community/eclipse_ newsletter/2014/march/article4.php.

[96] Stal, M. Azavea Announces Release of Geotrellis under GPLv3 License [J].

[97] Geotrellis [EB/OL]. http://github.com/geotrllis/geotrellis.

[98] Geotrellis transit [EB/OL]. http://transit.geotrellis.com/travelshed.html.

[99] Geotrellis watershed [EB/OL]. http://wikiwatershed.org/.

作者在学期间取得的学术成果

发表的学术论文

[1] 刘世永，吴秋云，陈荦，李军. 基于高层级地图瓦片的低层级瓦片并行合成技术 [J]. 地理信息世界, 2015, 06: 51-55.

[2] 刘世永，陈荦，熊伟，吴烨，李军. 一种基于 MPI 的大规模栅格影像并行瓦片化算法 [J]. 计算机工程与应用, 2016. (已录用待刊)

[3] Liu, S.Y., Chen L., Li J. An MPI + OpenMP Hybrid Parallel Paradigm for Pyramid Building Algorithms of Large-scale Remotely Sensed Images[J]. ISPRS International Journal of Geo-Information, 2016. (SCI 已投)

研究成果

[1] 刘世永，李军，吴秋云，等. 一种并行方式栅格影像切片方法: 中国, CN20161006 6304.7，2016. (中国专利公开号)

[2] 陈荦，吴秋云，刘世永，等. 一种混合并行方式的栅格影像金字塔构建方法: 中国，CN201610018294.X，2016. (中国专利公开号)

作者在学期间参加的与本课题相关的科研项目

[1] 国家高技术研究发展计划（863 计划）课题“面向新型硬件架构的复杂地理计算平台”，项目号：2011AA120305，2011AA120306. 负责高性能影像并行处理算法研发。

[2] 国家高技术研究发展计划（863 计划）课题“高性能 GIS 关键技术与软件系统”，项目号：2015AA123901. 负责高性能瓦片化算法研发。