网络出版时间：2017-02-16 09:56:58
网络出版地址：http://www.cnki.net/kcms/detail/11.2127.TP.20170216.0956.012.html

基于 MPI 的大规模栅格影像并行瓦片化算法

刘世永， 陈荦， 熊伟， 吴烨， 李军
LIU Shiyong, CHEN Luo, XIONG Wei, WU Ye, LI Jun
国防科学技术大学 电子科学与工程学院,长沙 410073
School of Electronic Science and Technology,National University of Defense Technology, Changsha 410073, China

Parallel raster image tile algorithm based on MPI. Computer Engineering and Applications

Abstract：The map tiles technology are widely used in WebGIS solutions of the current mainstream GIS software and the Internet map applications, tile processing service are the key technology to realize the fast and seamless browsing on WebGIS. Traditional algorithms and commercial GIS software cannot meet the demand of fast processing large data size raster image tiling. In view of these shortcomings and problem, this paper summarizes the current parallel tiling technology and then puts forward a new parallel raster image tile method which named “ParaTile”. ParaTile adopts the shared disk parallel technology, using multi-process to divide the source raster image. Each process of ParaTile is read and written independently to the area that it is owned based on MPIparallel IO function. Finally, each process saves the output tile results independently according to the TMS (Tile Map Service) or the standard made by Google Tile. Experiments were tested by different level size of the remote sensing image, the result shows that no matter in speed or stability ParaTile algorithm has a great improvement in the same parallel level compared to ArcGIS software, especially the larger size of data the more advantages are.

Key words：raster image; parallel; tile; MPI;

摘 要：当前主流 GIS 软件以及互联网地图应用在 WebGIS(网络地理信息系统)解决方案中都广泛采用地图切片（又称瓦片），切片处理服务是实现影像在 WebGIS 上快速无缝浏览的关键技术。针对目前传统算法以及商业 GIS 软件在大数据量栅格影像快速瓦片化方面的不足，本文提出一种名为 ParaTile 的高效栅格影像快速瓦片化方法，ParaTile 基于 MPI 共享外存的并行技术，利用多进程对原始栅格影像进行数据划分，每个进程对其所划分的区域进行独立读写和计算，而后再按照 TMS 或者 Google Tile 定义的标准将瓦片进行编码输出。实验采用不同级别大小的遥感影像进行测试，结果表明 ParaTile 在面对不同规模的数据时无论从速度还是算法稳定性上都较现有算法和工具具有显著优势，特别是当数据量越大时，这种优势愈加明显。

关键词：栅格影像；并行；瓦片化；MPI；
doi:10.3778/j.issn.1002-8331.1608-0322 **文献标志码：A** **中图分类号：TP391**

1 引言

随着卫星传感器技术以及无人机航拍技术的快速发展，遥感影像的空间和时间分辨率都有大幅度地提高，单幅遥感影像文件的数据量也急剧增加[1]。当前主流 GIS 软件以及互联网地图应用在 Web GIS(网络地理信息系统)解决方案中都广泛采用地图切片（Tile，又称瓦片）的发布策略[2]，这种方法通过预先将原始影像按照一定的切分规则切割成一张张大小相同的瓦片，以加载小数据量瓦片的方式达到在网络带宽受限的条件下实现影像的高效显示[4]。但是目前的商业 GIS 软件其地图切片的生成以及发布成本较高，操作较复杂，以行业内知名的 ArcGIS 系列软件为例，要将栅格影像进行切片不仅要安装 ArcGIS Desktop，而且还要安装庞大的 ArcGIS Server，并且操作十分复杂繁琐，极大增加

基金项目：国家 863 项目“高性能 GIS 关键技术和软件系统”(No.2015AA123901)。
作者简介：刘世永(1993－),男,硕士,研究领域为高性能影像数据瓦片化关键技术;**陈荦**(1973－),男,博士，教授，研究领域为高性能地理信息系统；**熊伟**(1976－),男，博士，副教授， 研究领域为内存数据库。E-mail: shiyongliu@nudt.edu.cn

时间和物力成本。最重要一点是随着单幅遥感影像文件的分辨率以及数据量的急剧增加，其相应的切片数量会呈现几何级数式的增长，传统的串行算法以及商业 GIS 软件是通过单机预先切好瓦片，再对外提供，这种传统切片技术计算资源利用低下，并且没有错误恢复机制，一旦切片过程中出现故障，整个切片工作得推倒重来，无法在原有的进度上继续进行。因此在硬件性能高速发展的情况下，如何利用高性能计算技术，方便快捷并且高效快速地进行栅格影像切片，已经成为 WebGIS 地图应用中快速可视化方面必须要解决的重要问题。

目前栅格影像并行切片方法主要有以下几种思路：一种是 CPU（Central Processing Unit，中央处理器）+GPU（Graphic Processing Unit，图形处理单元）的方式，利用 GPU 的计算能力进行并行加速，这种方法并行能力受限于 GPU 硬件的能力，并行程度有限，而且会提高系统架构成本，例如刘帅等人提出了一种基于 CPU+GPU 的地图切片的快速生成方法[5]；另一种是利用分布式集群系统，将切片任务划分为多个子任务，各子任务在多个分布式节点上同时进行，这种方法可以比较方便将原有的串行方法并行化，但是当数据规模比较大的时候，前期的数据分布式存储以及后期的结果合并都比较耗时[6]；再一种是基于多线程并行技术，各线程将任务划分成多个子任务，每个子任务同时进行，这种方法可以充分利用本机计算资源，但是由于线程间并行属于细粒度并行，各线程共享父进程的内存空间，容易造成系统的不稳定，并且这种方法可扩展性不是很好，这种方法目前是 ArcGIS 采用的策略；还有一种是利用并行技术进行海量小影像的瓦片化的批处理操作，这种方法任务并行划分简单，但是局限性大[8]。目前基于 MPI（Message Passing Interface），消息传递接口)的多进程方式进行单个大规模栅格影像的并行切片研究较少，这种方法利用共享外存的高性能集群，可扩展性强，稳定性强，可以充分利用多机计算资源，由于集群之间共享外存，所以数据无需提前分布式存储，可以极大地提高影像切片的效率。

2 栅格影像瓦片化基本原理

2.1 瓦片切分模型

瓦片最早是由 Google Map 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[9]，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、高德的图等，国外有 Google Map、Bing Map 等[11]。如图 1 所示，示意图以一张世界地图为例阐释 WebMercator 投影方式，假设地球被套在一个圆柱中，赤道与圆柱相切，然后在地球中心放一盏灯，把球面上的图形投影到圆柱体上，再把圆柱体展开，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。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，为了方便，web 墨卡托投影忽略了墨卡托投影中南北两级变形较大的区域，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，其大地坐标范围为[-180°,-85.051128779 8°,180°,85.051128779 8°]，投影坐标范围为[-20037508.3427892m,-20037508.3427892m,20037508.3427892m,20037508.3427892m][12]。

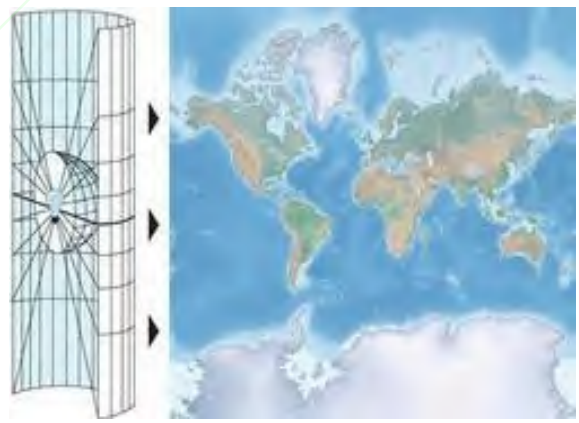

图 1 WebMercator 投影示意图

瓦片切分则是基于上述所说的 WebMercator 投影坐标系统，将栅格影像进行不同分辨率的切分，每个分辨率对应 WebGIS 进行缩放操作时相应的层级[13]。以图 2 为例，*level* 表示瓦片的级别，*tileszie* 为瓦片的长宽，则每一级别共有 4^{level} 个 $tileszie \times tileszie$ 大小的瓦片，因此瓦片级别和其对应的空间分辨率 *Resolution* 满足公式(1)：

$$Resolution = (2 \times \pi \times R) / (tilesize \times 2^{level}) \quad (1)$$

其中 R 为地球半径，瓦片划分采用四叉树的方式进行，即以赤道和本初子午线的交点作为中心，不断对地图进行四分，直到每个格网大小为 *tilesize*

×*tilesize* 为止[13]。如图 2 所示，0 级世界地图由一个瓦片表示，1 级世界地图应由 4 个瓦片表示，2 级 16 个，以此类推。因此当对一个普通栅格影像进行切片时，首先根据栅格影像的分辨率找到与其分辨率最接近的瓦片级别，而后通过上述所述的世界地图切分规则，计算影像所在该层世界地图瓦片中的位置，进行切片。

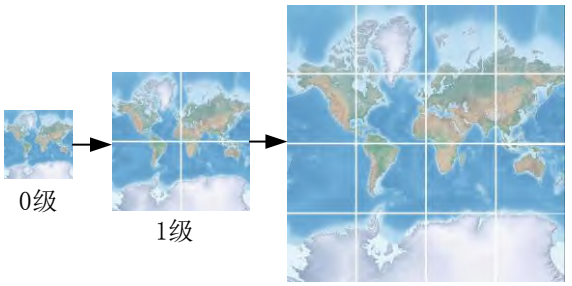

图 2 瓦片分级模型

2.2 瓦片切分模型

当地图被划分为 4^{level} 个瓦片后，需要对瓦片进行编码，存储在文件系统的相应区域以便浏览器前端能够快速访问，瓦片编码以瓦片在瓦片坐标系内的行列号为基础进行编制而成。瓦片编码是基于 WebMercator 投影坐标系进行的，如图 3 所示，瓦片坐标系的原点位于左上角，瓦片存储在文件系统内分为三级目录其中第一级为瓦片级别（*level*），第二级为瓦片列号（*tx*），第三级为瓦片行号（*ty*）。以字符串“root/level/tx/ty.png”作为瓦片图片的唯一编码，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,其中 root 为瓦片文件的根目录。

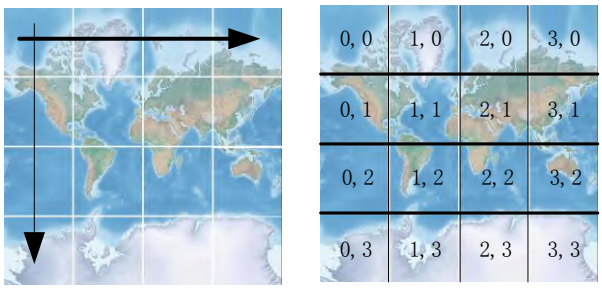

图 3 瓦片编码示意图

3 并行切片任务划分

并行瓦片化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。在进行任务划分前，需要先将原始影像投影变换到 WebMercator 坐标系得到投影变换结果影像，而后在这个坐标系下对投影变换结果影像进行数据划分。本文采用一种循环划分的分配策略，具体划分方法如图 4 所示，其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，各进程所分配得到的所有瓦片构成该进程的任务池。

可以通过如下公式计算得到每个进程任务池内的每个瓦片的行列号：假设进程的进程号为 *i*，*n* 表示进程总数，[*tminX,tminY,tmaxX,tmaxY*]为影像所覆盖的瓦片的行列号范围，则其对应瓦片行列号[*tx,ty*]满足(2)：

$$\begin{aligned} tx &= tminX + \left(pos\%twidth\right) \\ ty &= tmaxY - \left\lfloor pos/twidth \right\rfloor \end{aligned} \tag{2}$$

其中 *twidth* 和 *theight* 为 x 和 y 方向瓦片个数，*tcount* 为瓦片总数，解算方法如式(3)所示。

$$\begin{aligned} &twidth = tmaxX - tminX \\ &theight = tmaxY - tminY \\ &pos = j \times n + i, j \in [0, 1\cdots, k], \left\lfloor k = \frac{tcount}{n} \right\rfloor \\ &tcount = twidth \times theight \end{aligned} \tag{3}$$

假设在 WebMercator 投影坐标系下影像地理范围为[*ominX,ominY,omaxX,omaxY*]，其中 *ominX*，*omaxY* 为左上角点坐标，*omaxX,ominY* 为右下角点坐标。那么瓦片的行列号范围[*tminX,tminY,tmaxX,tmaxY*]可通过影像的地理范围计算得出，具体解算方法如式(4)所示。

$$\begin{aligned} tminX &= \left\lceil \frac{ominX + originShift}{Resolution \times tilesize} \right\rceil - 1 \\ tminY &= \left\lceil \frac{ominY + originShift}{Resolution \times tilesize} \right\rceil - 1 \\ tmaxX &= \left\lceil \frac{omaxX + originShift}{Resolution \times tilesize} \right\rceil - 1 \\ tmaxY &= \left\lceil \frac{omaxY + originShift}{Resolution \times tilesize} \right\rceil - 1 \end{aligned} \tag{4}$$

其中 *originShift*=(2×π×6378137)/2 即地球周长的一半，*Resolution*=(2×π×6378137)/((*tilesize*×2^{level}))，

tilesize 为瓦片边长大小。⌊ ⌋表示向下取整，⌈ ⌉表示向上取整。

如果当 *tcount%n*≠*0* 时，进程号 *i*<*tcount%n* 的部分进程还需处理瓦片行列号[*tx2*,*ty2*]满足以下条件的瓦片：*tx2=tminX+(pos2%twidth)*；*ty2= tmaxY-⌊(pos2/twidth)⌋*，其中 *pos2=tcount-tcount%n+i*。

因此根据上述所述的任务划分方法，只需循环 $\left\lceil \frac{tilecount}{n} \right\rceil$ 次就可以处理完所有的切片任务。

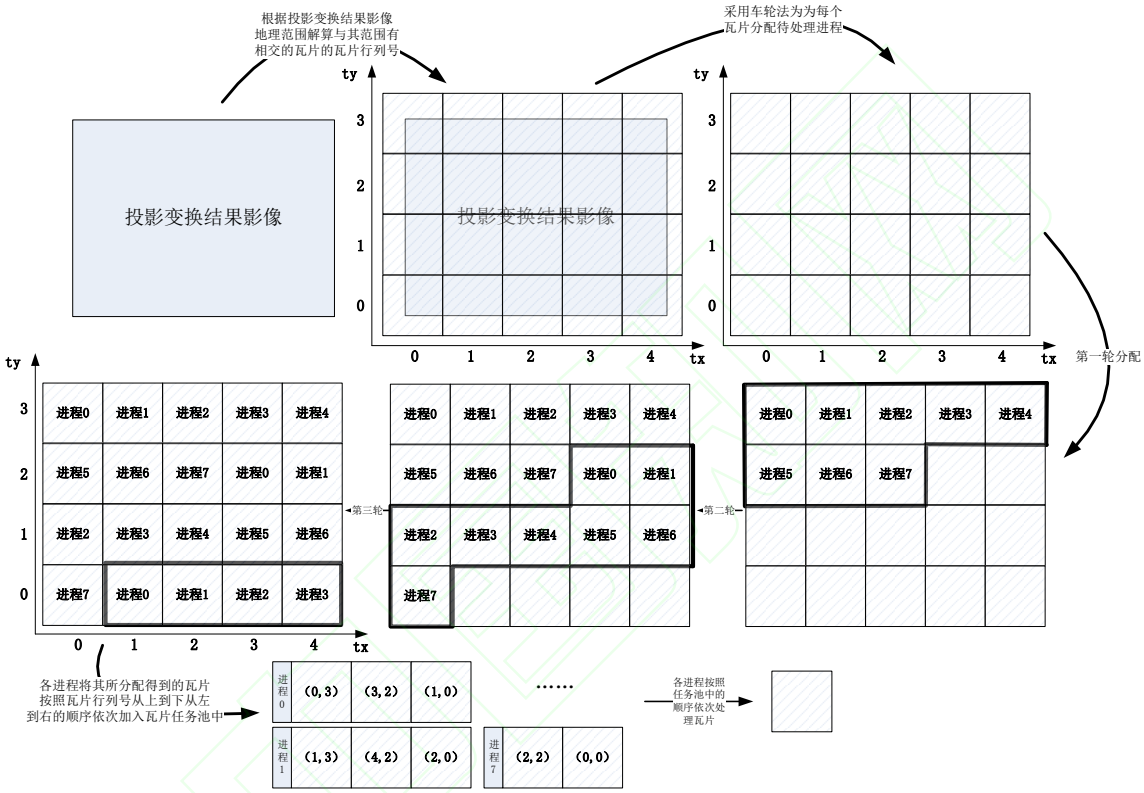

图 4 并行切片任务划分

4 数据并行读取

当每个进程的任务被分配好后，各进程依次遍历其所属的瓦片，根据瓦片的行列号解算其在原始栅格影像的数据读取位置。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 *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该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(5)所示。

$$\begin{aligned} gminX &= tx_i \times Resolution-originShift \\ gminY &= ty_i \times Resolution-originShift \\ gmaxX &= (tx_i+1) \times Resolution-originShift \\ gmaxY &= (ty_i+1) \times Resolution-originShift \end{aligned} \tag{5}$$

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，具体解算过程如下：以米为单位假设投影变换结果影像的地理范围为[*gimgminX*,*gimgminY*,*gimgmaxX*,*gimgmaxY*]，空间分辨率为 *Geores*，那么投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* 和 *rysize* 可由式(6)计算得到。

$$
\begin{aligned}
rxsize &= \frac{gistmaxX-gistminX}{Geores} \\
rxsize &= \frac{gistmaxY-gistminY}{Geores} \\
rx &= \begin{cases} 0, gistminX = gimgminX \\ \dfrac{gistminX-gimgminX}{Geores}, gistminX \neq gimgminX \end{cases} \\
ry &= \begin{cases} 0, gistmaxY = gimgmaxY \\ \dfrac{gimgmaxY-gistmaxY}{Geores}, gistmaxY \neq gimgmaxY \end{cases}
\end{aligned} \tag{6}
$$

5 数据并行写入

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(7)所示：

$$
\begin{aligned}
wxsize &= \frac{rxsize \times tilesize}{(gmaxX-gminX)/Geores} \\
wysize &= \frac{rysize \times tilesize}{(gmaxY-gminY)/Geores}
\end{aligned} \tag{7}
$$

利用最邻近内插法将栅格数据大小从 $rxsize \times rysize$ 重采样到 $wxsize \times wysize$。

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，回到第七步，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 (Geospatial Data Abstraction Library, 地理空间数据抽象库)[15] 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*)，计算方法如式 8 所示：

$$
\begin{aligned}
wx &= \begin{cases} \dfrac{(gistminX-gminX)\times tilesize}{gmaxX-gminX}, gistminX=gimgminX \\ 0, gistminX \neq gimgminX \end{cases} \\
wy &= \begin{cases} \dfrac{(gmaxY-gistmaxY)\times tilesize}{gmaxY-gminY}, gistmaxY=gimgmaxY \\ 0, gistmaxY \neq gimgmaxY \end{cases}
\end{aligned} \tag{8}
$$

然后各进程利用 GDAL 的 RasterIO 函数将 *wx*，*wy*，*wxsize*，*wysize* 填入相应参数将重采样数据并行写入瓦片文件。详细流程请参看图 8。

6 算法执行步骤

图 5 为 ParaTile 的算法实现步骤图，首先设置主进程、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栅格影像投影坐标、长宽等元数据信息，主进程利用 GDAL 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，后续的所有切片操作都是基于投影变换结果影像进行，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。

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*。

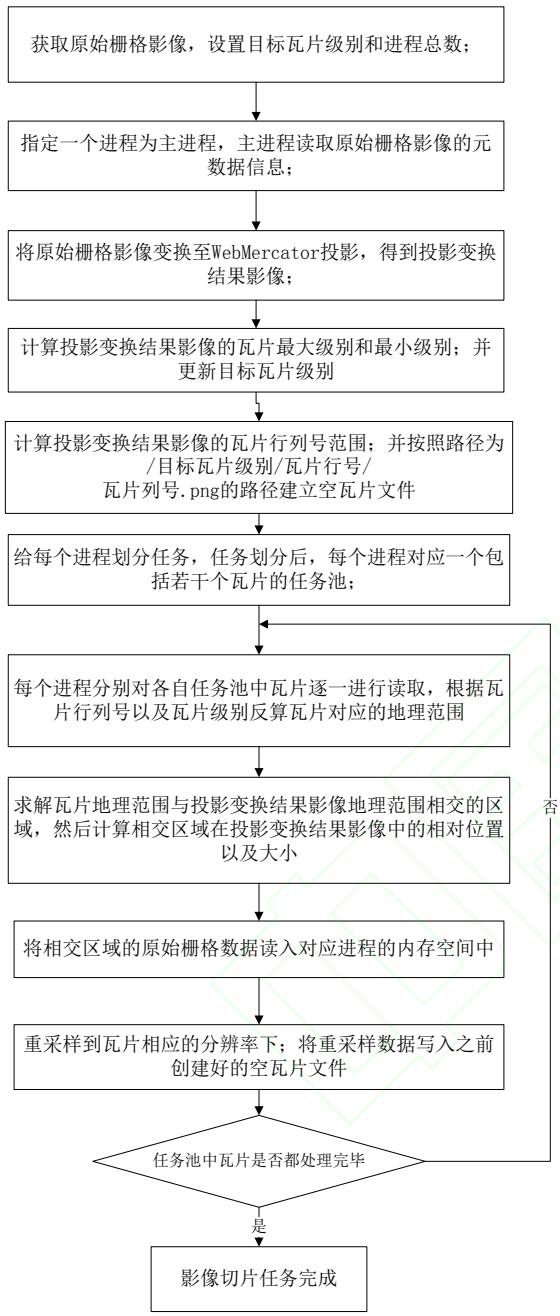

图 5 算法执行步骤

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。

7 实验结果分析

为了保证实验的准确性，实验平台采用两台硬件环境一模一样的超微高性能计算机，其中一台搭载 Linux CentOS6.3.X86_64 操作系统，另外一台搭载 Windows 7 操作系统，具体的硬件环境见表 1。

表1 实验硬件环境

类型	描述
CPU	Four Inter(R) Xeon CPU E5-4620, 2.60GHz, 8 cores per CPU, 64 virtual CPU cores using hyper-threading technology
RAM	Totally 768GB, Samsung 32GB per each
存储系统	48TB磁盘阵列
操作系统	Windows 7 Professional, 64-bit/ Linux CentOS6.3.X86_64

实验数据采用不同大小、分辨率以及不同投影坐标系下的的多波段遥感影像进行测试，实验数据详细信息见表 2。

表2 实验数据

影像名称	影像长	影像宽	数据类型	波段数	大小	分辨率	最大切片级别	投影信息
1.tif	17152	11520	8 位字节	3	605.04MB	2.389 米	15	WebMercator
2.tif	38265	38376	8 位字节	3	4.62GB	0.00014 度	13	无（大地坐标系）
3.tif	87040	58368	8 位字节	3	15.2G	0.299 米	19	WebMercator

7.1 评价 ParaTile 并行化程度

实验通过采用表 2 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对算法进行测试，记录算法在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。实验结果为 10 次实验的平均值，图 6 为实验结果，横坐标是进程数，纵坐标是算法执行时间（单位为秒）。

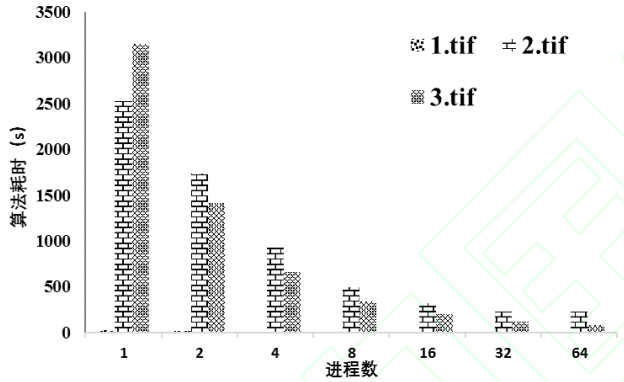

图 6 ParaTile 耗时随进程数的变化情况

从图 6 中可以看到：（1）一定范围内随着进程数的增加，ParaTile 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。

由于测试数据 1.tif 数据量比较小，执行速度比较快，所以在图中的柱形中不易观察到。2.tif 和 3.tif 虽然数据量上差了三倍多，但是从实验结果来看，处理 2.tif 并没有比 3.tif 快很多，这主要是因为对 2.tif 进行切片时必须先将它投影变换到 WebMercator 投影坐标系下，这部分是比较耗时的，所以造成了这种现象。

7.2 ParaTile 与 ArcGIS 性能对比

实验采用 ArcGIS10.2，测试数据采用上述 3 幅影像，切片级别都采用影像的最大切片级别，ParaTile 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，实验结果为 10 次测试结果的平均值，图 7 为实验结果，图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。

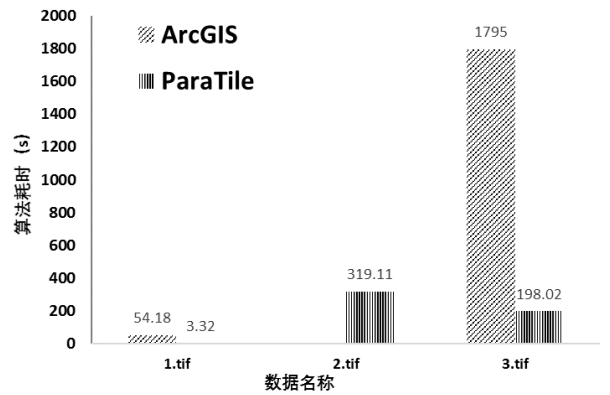

图 7 ParaTile 与 ArcGIS 性能对比

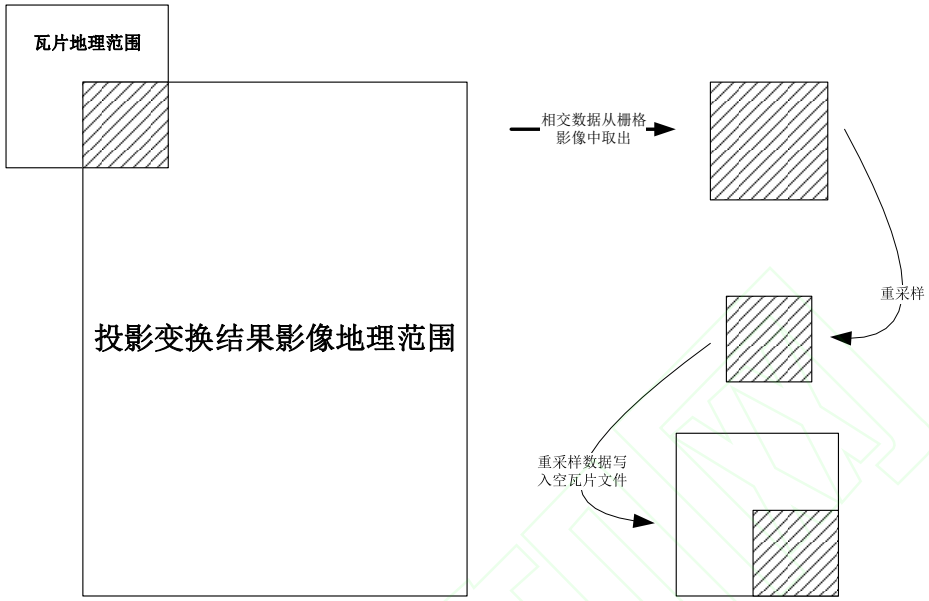

图 8 算法执行流程

8 结束语

本文通过深入研究瓦片模型的组织方式以及瓦片化的关键流程，针对当前一些栅格数据瓦片化方法存在的问题和不足，提出一种新的基于 MPI 的并行栅格影像瓦片化方法，算法通过重采样以及 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。算法未来进一步优化的方向可以从集中式共享存储架构转向分布式存储架构，这样可以解决目前的 IO 瓶颈问题。实验表明本文算法相比现有的瓦片化方法，性能得到了极大的提升，特别是随着数据规模以及波段数的增大，优势越明显，该算法已实际部署并服务于湖南省国土测绘部门，简化了传统基于 Arc GIS 的复杂切片操作流程，在相同数据量下至少可提高传统工作效率的 8 倍以上。

参考文献

[1].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.

[2].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.

[3].赵大龙, 孙恒宇. 地图切片技术分析与简单实现[J]. 测绘与空间地理信息, 2010, 33(1):116-118.

[4].Turner N, Fernandez C, Lessin M. Image tile server: US, US8244770[P]. 2012.

[5].刘帅. 一种基于 CPU+GPU 的地图切片的快速生成方法[P]. 中国：CN104267940A,2015-01-07.

[6].Yi L. Parallel Batch-Building Remote Sensing Images Tile Pyramid with MapReduce[J]. Geomatics & Information Science of Wuhan University, 2013, 38(3):278-282.

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

[8].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.

[9].Battersby S E, Finn M P, Usery E L, et al. Implications of Web Mercator and Its Use in Online Mapping[J]. Cartographica the International Journal for Geographic Information & Geovisualization, 2014, 49(2):85-101.

[10]. Frančula, Nedjeljko. Web Mercator projection[J]. Geodetski List Glasilo Hrvatskoga Geodetskog Društva, 2014.

[11]. 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.

[12]. 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.

[13]. 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.

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

[15]. 葛亮, 何涛, 王均辉,等. 基于 GDAL 的瓦片切割技术研究[J]. 测绘与空间地理信息, 2014(7):130-132.