Computers & Geosciences 59 (2013) 31–40

Contents lists available at SciVerse ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Parallel scanline algorithm for rapid rasterization of vector geographic data

Yafei Wang, Zhenjie Chen, Liang Cheng *, Manchun Li **, Jiechen Wang

Jiangsu Provincial Key Laboratory of Geographic Information Science and Technology, Nanjing University, 22 Hankou Road, Nanjing 210097, China

ARTICLE INFO

Article history:
Received 7 September 2012
Received in revised form
15 May 2013
Accepted 17 May 2013
Available online 27 May 2013

Keywords:
Vector geographic data
Rasterization
Parallel processing
Scanline
Speedup

ABSTRACT

With the expansion of complex geographic calculations and the increase of spatial data types involved in the spatial analysis of large areas, the need becomes urgent for fast rasterization of massive multi-source geographic vector data. A parallel scanline algorithm is proposed for rapid rasterization. It provides a systematic solution to solve the complicated situation in parallel processing (cross-processor boundaries, common boundaries, and tiny polygons), thus ensuring the accuracy of the parallel scanline algorithm. The relationship of parallel speedup with the number of processors, the data partition pattern, and the raster grid size is discussed. Massive vector geographic data (approximately 0.7 million polygons) used in the experiment were effectively processed, thereby dramatically reducing the processing time and getting good speedup.

1. Introduction

Vector data and raster data are the two basic geographic data types used in the Geographic Information System (GIS) (Maguire et al., 1991), and the latter is more suitable for spatial analysis and spatial simulation (Goodchild, 2011). As satellite technologies advance forward, raster data have become increasingly more popular than vector data. At present, raster data is already the most dominant format used by data sources. With the expansion of complex geographic calculations and the increase of spatial data types involved in the spatial analyses on large areas (Lee et al., 2011), the need for rasterization of massive multi-sourced vector geographic data becomes more and more urgent. However, due to the sequential architecture of existing rasterization algorithms and the traditional desktop computer platform, current solutions cannot meet the strong demand of fast rasterization on massive vector geographic data. In recent years, the gradual popularity of the new parallel hardware architecture, such as computer clusters and multi-core processors, offers a new opportunity to improve the conversion speed of massive geographic data, which had been restricted by the limited computing performance of older technologies (Kenneth et al., 2003; Gong and Xie, 2009). To achieve the requirements of rapid rasterization on massive vector data, it is necessary to combine geographic data conversion technology with parallel hardware architecture by developing new parallel algorithms, thereby reducing processing time through parallel computing (Mariethoz, 2010; Schiele et al., 2012).

Parallel rasterization of vector geographic data includes parallel rasterization of point data, of line data, and of polygon data. Among these three, parallel rasterization of polygon data is relatively complex. Therefore, this paper focuses on the study of parallel rasterization algorithms on polygon data. Based on the analysis of the limitations of the existing researches, a new parallel scanline algorithm for rapid rasterization of vector polygon data is proposed. Its main procedures include: (1) Executing particle-sized partitions on vector polygon data, according to the number of processors and the spatial location of the vector polygon data, so as to adapt to different-sized grid blocks, and (2) Fusing partition parts, in which a parallel strategy using pixel-center scanlines is proposed to solve the complicated situation, including cross-processor boundaries, common boundaries, and tiny polygons.

The algorithm was implemented for multi-core processors. The land use data of Changsha City, China was used to check the performance of this algorithm, and the results were compared with those derived from the commercial software ArcGIS. Meanwhile, this research measures the operating time and speedup of this parallel algorithm, and this paper discusses the relationship of the parallel speedup with the number of processors, the data partition pattern, and the grid cell size.

2. Related work

2.1. Parallelization potential of existing serial rasterization algorithms

Vector data rasterization is the basic issue in the Geographic Information System (GIS) (Congalton, 1997). Rasterization is a lossy

* Corresponding author. Tel.: +86 18652071855; fax: +86 25 83597359.
** Corresponding author. Tel.: +86 13705167312; fax: +86 25 83597359.
E-mail addresses: lcheng@nju.edu.cn (L. Cheng), limanchun_nju@126.com (M. Li).

0098-3004/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cageo.2013.05.005

conversion process; therefore, errors are inevitable, regardless of improvements in the conversion accuracy (Liao and Bai, 2010). Different GIS software applications have different raster algorithms, which could produce different results, even with the same data (van der Knaap, 1992). Traditional polygon rasterization algorithms include the point-based rendering method, the X-ray method, the scanline method, and the boundary algebraic method. Later rasterization algorithms were mostly derived from or improvements of these methods. Some researchers focused on the fast implementation of rasterization, such as the winding number algorithm (Hormann and Agathos, 2001), the hierarchical triangle-based method (Jiménez et al., 2009), and the point-in-polygon method based on the quasi-closest point (Yang et al., 2010). Other researchers considered the lossy conversion process and focused on improving the accuracy of the rasterization, such as the optimization algorithm based on errors in minimized areas (Wang et al., 2006) and the conversion of equal areas (Zhou et al., 2007). Among all the available methods, the scanline algorithm is the simplest and the most efficient. Moreover, average partitioning of rows and columns is easy to do, because different rows and columns are not related, making the algorithm ideal for parallel processing. This algorithm is also highly accurate and can solve difficult problems, such edge pixels and tiny polygons, by defining a threshold.

2.2. Analysis of existing parallel rasterization algorithms

Two kinds of parallel rasterization algorithms exist: one is geared toward the computer graphics field, while the other is geared toward the GIS field. Parallel rasterization algorithms have long been used in computer graphics research, resulting in relatively significant advancements. Pineda (1988) examined the position between the point and the polygon and drew the 3D graph in parallel by interpolating pixels through linear marginal functions. McManus and Beckmann (1997) proposed a mathematical method to decide the best screen-blocking process and to render the sub-partitions using a multi-core processor. Correa et al. (2003) presented a sort-first parallel system for out-of-core rendering of large models on cluster-based tiled displays, which rendered high-resolution images of large models at interactive frame rates using off-the-shelf PCs. Popescu and Rosen (2006) presented two forward rasterization algorithms designed for small polygonal primitives, which were efficiently rasterized by interpolation among a polygon's vertices. Roca et al. (2010) presented a rasterization approach that could be fully integrated in the current GPU rendering pipeline and that was optionally selected by API users to process streams of microtriangles. Hollander et al. (2011) proposed a new GPU algorithm to perform an efficient parallel computation of multiple-view levels of detail from a Bounding Volume Hierarchy, which could be used on low-detail representations such as polygon soups. Aimed toward the GIS geographic data processes, Healey et al. (1998) designed a parallel algorithm that converts vector data into raster data, which provided the selection criteria for and used the parallel vector data algorithm. However, none of these algorithms has considered the storage characteristics of the vector polygon in GIS, such as the class area, the shape and structure of the polygon, its geometric position, and its attributes, thus making the algorithms difficult to apply to vector polygon rasterization.

3. Design of parallel algorithm

3.1. Partition of polygon data

Common architectures of parallel algorithms include the task parallel architecture, the pipeline parallel architecture, and the data parallel architecture (Subhlok et al., 1993). The scanline algorithm on vector polygons is simple and has strong data dependencies; therefore, the task parallel architecture would waste significant communication time. Different processes have very different computing times, with the majority of the time spent on calculating intersections for the scanline algorithm; therefore, the pipeline parallel architecture is very difficult to use. The scanline algorithm uses almost the same operation on each pixel of every line/column, making it easy to slice the data and to achieve an optimal loading balance (Guan and Clarke, 2010).

For this study, we chose the data parallel architecture, which focuses on dividing the data. The main process is presented in Fig. 1 using pseudocode.

The affine transform describes the relationship between raster positions (in pixel/line coordinates) and georeferenced coordinates. It consists of six coefficients that map pixel/line coordinates (*xPixel*, *yLine*) into georeferenced space (*xGeo*, *yGeo*) using the following equations:

$$xGeo = gT[0] + xPixel \times gT[1] + yLine \times gT[2]$$

$$yGeo = gT[3] + xPixel \times gT[4] + yLine \times gT[5]$$

For north-up images, the $gT[2]$ and $gT[4]$ coefficients are zero, $gT[1]$ is the pixel width, and $gT[5]$ is the pixel height. The ($gT[0]$, $gT[3]$) position is the upper left corner of the upper left pixel of the raster.

All raster blocks derived from the raster partition are processed using the equations above. If one polygon crosses several raster blocks, it will be extracted by the corresponding raster blocks.

3.2. Parallel partition fusion

This section focuses on the processes used in complicated situations, including cross-processor boundaries, common boundaries, and tiny polygons.

3.2.1. Cross-processor boundary

As shown in Fig. 2(a), *Polygon A* covers two adjacent raster blocks (*Block* 0 and 1). The boundaries (blue and red) cross these two blocks (corresponding to two processors) and are named

Begin do
Create a null raster dataset according to spatial location of the polygon datasets.
Get the row size of raster dataset *rasterYsize* and the number of the processors *np*. Calculate the size of raster partition *nYchunksize* by the by the following formula
nYchunksize←*ceil [rasterYsize / np]*
where *ceil[n]* function is to return the smallest value that is not smaller than *n*.
Calculate the first row coordinate of each raster partition *iY* is calculated by the following formula
iY← *cp*× *nYchunksize*
where *cp* is the process *ID*.
//Calculate the first row coordinate of the last processor
If (*iY* + *nYchunksize*)< *RasterYsize*
iY←*rasterYsize-iY*
/*Divide the polygons through a spatial query operation*/
// Four corner points' coordinates of each raster block
dulx←0, *duly*←*iY*, *dlrx* ←*rasterXsize-1*, *dlry* ←*iY+nYChunkSize-1*
Transform *dulx*, *duly*, *dlrx*, *dlry* to the new coordinates belonging to spatial references in the vector data through affine transform
Create a window by the four corner points and use the window to intersect the polygons through a spatial query operation
Get the intersected polygons for each raster partition
End do

Fig. 1. Pseudo-code for the procedure of data partition.

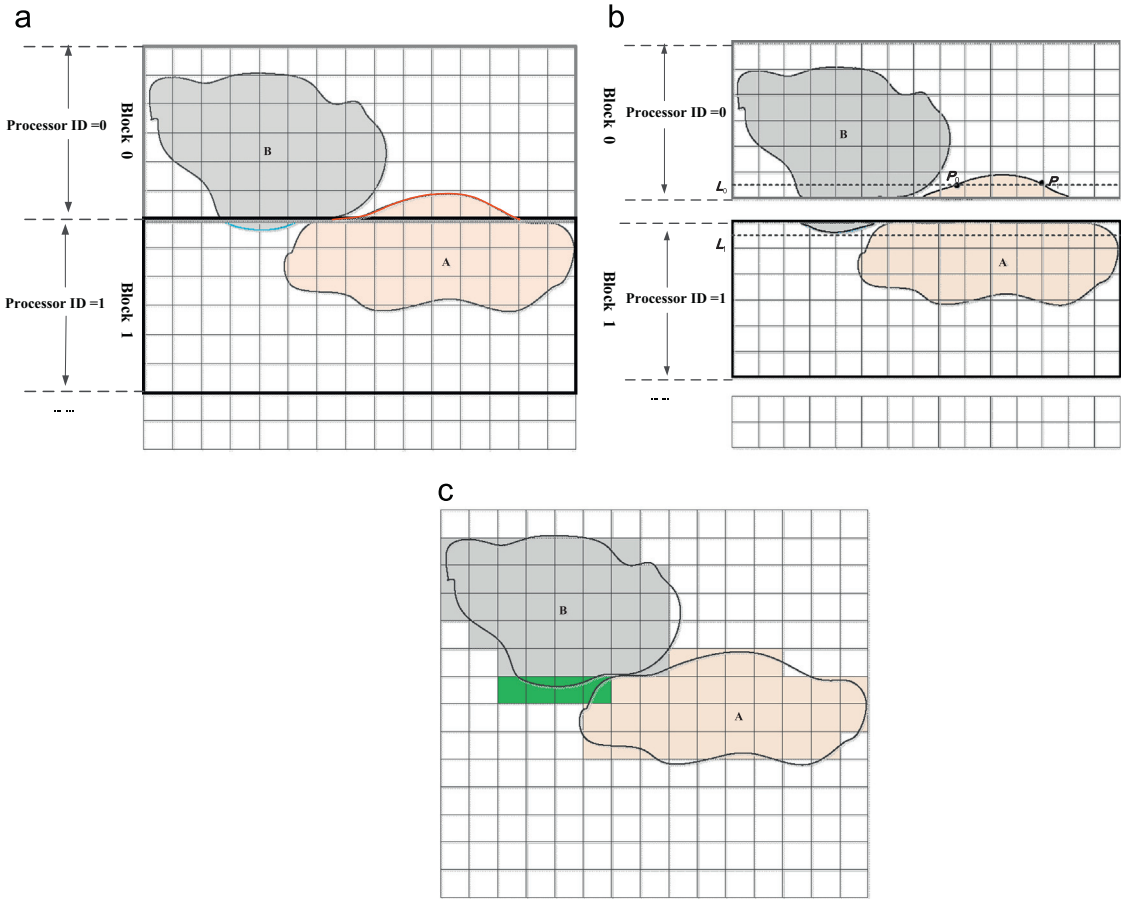

Fig. 2. Parallel scanline processing of the cross-processor boundaries. (a) Vector Polygon A and B operated in parallel by Processors 0 and 1 (the blue and red boundaries named as cross-processor boundaries), (b) Scanline operation performed to two polygons in Processor 0 and 1, simultaneously, (c) The corresponding rasterization results (gray and pink area) to Polygon A and B (green area not involved in rasterization based on pixel center-based scanline algorithm). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

cross-processor boundaries. The process used with these cross-processor boundaries significantly influences the accuracy of the rasterization. As shown in Fig. 2(b), *Polygon A* is divided into two parts by *Processor* 0 and *Processor* 1. In *Processor* 0, a pixel-center-based scanline strategy is presented to examine the position between *Polygon A* and the raster grids in *Block* 0. *Line* L_0 scans from left to right, and two points P_0 and P_1 are then obtained by intersecting L_0 with *Polygon A*. The region between P_0 and P_1 in *Block* 0 is filled by using the attribute of *Polygon A*. In parallel, the same process is implemented in *Processor* 1 to fill the corresponding raster region of *Block* 1 by using *Polygon A*. Similar operations are applied to *Polygon B* and other polygons.

The main advantage of this pixel-center-based strategy is that the slight cross-processor boundaries (the blue boundary of *Polygon B* in Fig. 2) can be processed accurately. Although *Polygon B* also covers two adjacent raster blocks, the green area in *Block* 1 is not filled with *Polygon B*, because no intersecting points exist between *line L*1 and *Polygon B*. The use of this strategy improves the accuracy of the rasterization. The corresponding rasterization results (gray and pink area) to Polygon A and B are shown in Fig. 2(c).

3.2.2. Common boundary

In Fig. 3(a), a common edge exists between Polygon A and Polygon B and covers the black area. The task is to determine the polygon that each grid in the black area belongs to. If a grid's pixel center point is inside Polygon A, the grid is assumed to belong to Polygon A. Before conducting the scan, all vertices of the polygon are pushed into the stack according to ID number, and the vertices coordinates are transformed from the geospatial coordinate system into the pixel/line coordinate system through the affine transform, as shown in Fig. 3(b). In order to ensure accuracy, the coordinates of the polygon vertices are recorded as double-precision data types after the affine transform. In Fig. 3(b), Point $P_i(x_i, y_i)$ and $P_{i+1}(x_{i+1}, y_i)$ (red point) are obtained by intersecting the scanline with Polygon A. The x coordinate of the raster grid M (green) belonging to the common boundaries is set to d_x. The attribute of M is examined by Eq. (1). If d_x satisfies Eq. (1), then grid M belongs to *Polygon A*. If not, the next common edge pixel is selected. If two polygons equally share the area of a pixel, the pixel is considered to belong to the polygon with the smaller ID number.

$$\text{floor}(x_i + 0.5) \le d_x \le \text{floor}(x_{i+1} + 0.5) \quad (1)$$

where floor(n) returns the largest integer less than or equal to the specified double-precision floating-point number.

3.2.3. Tiny polygon

Tiny polygons are an exception to the intersection case, because the size of the polygon is smaller than the cell size, as shown in Fig. 4(a). The generation of tiny polygons is related to its data source and to the cell size when creating the raster datasets. In Fig. 4(b), Point P_1 and P_2 were created by intersecting a scanline with *Polygon A*. Point P_3 and P_4 were created by intersecting a scanline with *Polygon B*. If the coordinate difference between two intersection points of a polygon is greater than 0.5 pixel, the grid is assigned as belonging to this polygon; otherwise, the tiny polygon is discarded.

In Fig. 4(b), the coordinate difference between P_1 and P_2 is greater than 0.5 pixel; therefore, the corresponding grid is assigned as belonging to *Polygon A*. However, the coordinate difference between P_3 and P_4 is less than 0.5 pixel; therefore, *Polygon B* is discarded.

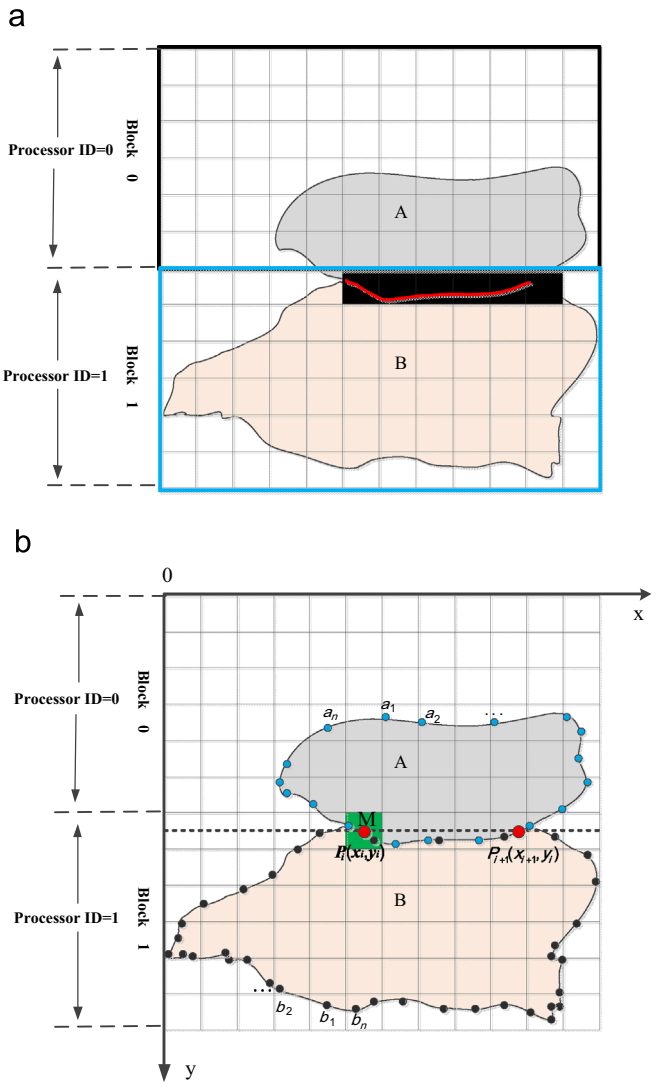

Fig. 3. Parallel scanline processing of the common boundaries. (a) Common boundaries (black), (b) The pixel value of the green grid estimated by analyzing the relationship between it and two red points.

3.3. Design of parallel efficiency analysis

Speedup is the most common standard for judging the efficiency of parallel algorithms (Xie, 2012). Speedup can be defined as the ratio of execution time of an optimal serial algorithm to that of the corresponding parallel algorithm for the same task. The speedup of this parallel scanning algorithm can be evaluated by considering three factors: number of processors, data partition pattern, and raster grid size. This study discusses the relationship of parallel speedup with these three factors.

Multi-core processors were used for the experiment, and the change of speedup with different numbers of processors was measured. The influence of the data partition on the speedup can be analyzed by two partition patterns: row-partition and column-partition. The row-partition pattern divides the input data into several partitions by row, as illustrated in Fig. 2. The column-partition pattern divides the input data into several partitions by column. We compared the difference in parallel speedup between the two partition patterns. Finally, the relationship between the raster grid size and the speedup is considered. The following raster grid sizes were selected for this experiment: 5 m, 10 m, 20 m, 30 m, 40 m, 50 m, 60 m, 70 m, 80 m, 90 m, 100 m, 200 m, 300 m, 400 m, and 500 m. For each of these grid sizes, the corresponding maximum speedup value was obtained and compared with the others.

In order to compare with the parallel scanline algorithm in this paper, we designed and implemented another parallel rasterization algorithm, based on cell rasterization algorithm, which is one of the most accurate rasterization algorithms. It pinpoints the inclusion relationships between a specific raster grid cell and the polygons, one by one. We adopted the same parallel architecture in our parallel cell rasterization algorithm.

4. Implementation of parallel algorithm

We used the C++ programming language to implement the parallel algorithm with Message Passing Interface (MPI), using the development platform Microsoft Visual Studio 2008. The rasterization of vector data mainly consists of three parts: reading the vector data, calculating, and writing the raster data. All parts can be parallelized by MPI. The read and write operations were implemented using the open-source Geospatial Data Abstraction Library (GDAL). The data formats of the available polygon vector data included ESRI Shapefile, MapInfo MIF (*.mif), MapInfoTab (*.tab), AutoCAD (*.dxf), and MicroStation Design (*.dgn). The output data formats were raster data formats like the ENVI.hdr

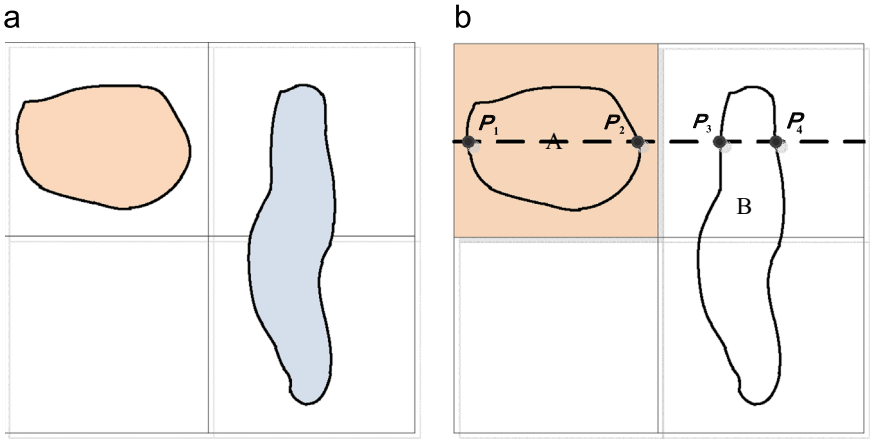

Fig. 4. Processing of tiny polygons. (a) vector tiny polygon, (b) Result of rasterization.

Labelled Raster, ERDAS Imagine(*.img), GeoTIFF(*.tif), and ESRI.hdr Labelled. The pixel values of the output raster were assigned as the attribute fields of a polygon, and the input field types determined the types of the output raster. The specific procedures of the raster parallel algorithm on vector polygons are as follows:

Step 1: The master processor opens the vector file and determines the extent of the vector data.
Step 2: The master processor creates a null raster dataset, according to the minimum bounding window of all vector polygons, and assigns its parameters, including the grid size and data format.
Step 3: The master processor perform data partition and creates *Block* 0, 1, …, $n-1$ by dividing the raster dataset.
Step 4: The master processor sends a message to slave processors by point-to-point communication. The message includes the start row, the rows of raster partition, and the end row. All slave processors ($n-1$) are called to read the $n-1$ raster blocks in parallel synchronously.
Each slave processor is used for the subsequent processing from Step 5 to Step 8.
Step 5: In one slave processor (*Processor* 1), the coordinates of four corner points belonging to *Block* 1 are transformed according to the spatial reference information of the vector data through affine transform. A window created by the transformed coordinates extracts the polygons through a spatial query operation.
Step 6: Each slave processor reads the extracted vector polygons, and the vertex coordinates and attribute values of the polygons are stored in a vector, according to the polygon's ID number.
Step 7: *Processor* 1 performs the parallel scanline operation. The detailed process is presented in Fig. 5 using pseudocode.
Step 8: The slave processors write the $n-1$ raster blocks($n-1$).
Step 9: A new raster image is created by combining the rasterization results from all slave processors.

Complicated situations, especially cross-processor boundaries, common boundaries, and tiny polygons, are handled based on the proposed systematic solution to ensure the accuracy of the parallel scanline algorithm.

```
Calculate the total number of polygons pcount and the vertex number of each polygon vcount
      of within Block 1
Estimate the start-row and end-row of scanning line according to the minimum bounding
      window of the polygons in this block.
For Linex = start-row to end-row
   dy= Linex +0.5
   For Polygony=0 to pcount-1
      For Vertexz = 0 to vcount-1
      The row coordinates of two adjacent vertex are set to (dx1,dy1), ( dx2,dy2) ( dx1
      <dx2,dy1 <dy2)
      Calculate the intersection coordinates of polygons with dy
      If((dy < dy2 ) && (dy >= dy1))
            Intersect ionx= (dy-dy1) * (dx2-dx1) / (dy2-dy1) + dx1
      Sort by the row and record the coordinates of the intersections
      Set the pixel value of each raster grid between two intersections in this line by the
      corresponding polygon ID
      End for Vertexz loop
   End for Polygony loop
End for Linex loop
```

Fig. 5. Pseudo-code for specific procedures of parallel scanline operation.

5. Experiment

5.1. Experimental data and parallel environment

The experiment data, which comes from the land use data of Changsha City in China (Fig. 6), has a specific area range of 28.666 degrees north latitude, 27.836 degrees south latitude, 111.877 degrees west longitude, and 114.256 degrees east longitude, covering an area of 11,819.46 square kilometers. The data format is the ESRI shapefile. The total number of the polygons is 692,177, and the data volume is 938 MB. Its spatial reference system is the 1980 Xi'an coordinate system of China. The land use types have 24 categories (Table 1), including paddy fields, irrigated land, dry land, garden plots, forest land, pasture land, agricultural land for facilities, rural roads, etc.

The computer used for this program is an IBM System x3500-M3X, which is equipped with eight CPUs (Intel Xeon Quad Core E5620 with a 2.4 GHz speed, 12 MB cache, quad-core processors), two 4 GB memory chips (DDR3 1333 MHz LP RDIMM); four hard drives of 500 GB each (7.2 K 6 Gbps NL SAS 2.5-in. SFF Slim-HS HD). The network is the Integrated Dual Gigabit Ethernet, and the operating system is the CentOS Linux 6.0. OpenMPI 1.4.1 was selected among the MPI products.

5.2. Accuracy analysis

The indexes used to judge the rasterization accuracy usually include the area accuracy, perimeter accuracy, and classification accuracy. For this study, we chose the area accuracy and classification accuracy indexes. The field values of the land use type in the polygon data are chosen as the pixel values in the process of rasterization.

This experiment compares the result of this parallel algorithm rasterization with the result of rasterization using the ArcGIS software. ArcGIS9.3 provides 3 kinds of rasterization methods including the CELL_CENTER method, the MAXIMUM_AREA method, and the MAXIMUM_COMBINED_AREA method. In the CELL_CENTER method, the polygon that overlaps the center of the cell provides the attribute to assign to the cell. In the MAXIMUM_AREA method, the single feature with the largest area within the cell provides the attribute to assign to the cell. In the MAXIMUM_COMBINED_AREA method, the areas of features with the same value are combined and the combined feature with the largest area within the cell determines the value to assign to the cell. This experiment compares the result of this algorithm with the result of the CELL_CENTER method, which is very similar to this algorithm.

The vector polygon data used in this experiment is shown in Fig. 7(a). The size of the output raster data is 11630×4510, and the grid size is 20 m*, as shown in Fig. 7(b). Using a spatial scale of 20 m and comparing the rasterization results against the results from the ArcGIS software, we estimated the rasterization accuracy (classification accuracy) of various land use types by processing the data using the proposed parallel algorithm, and the results are listed in Table 1. From this table, the accuracy of the rural roads type is 99.45%, whereas the accuracy of other types is larger than 99.70%.

We also calculated the classification accuracy of the proposed parallel algorithm under different spatial scales. Table 2 lists the classification accuracy for the parallel scanline algorithm and for the parallel cell algorithm under various spatial scales (5 m, 20 m, 50 m, 100 m, and 500 m). The classification accuracy of the parallel cell algorithm is greater than that of the parallel scanline algorithm, and it very slightly declines with the increase of raster spatial scale.

The area accuracy of the proposed parallel algorithm was also computed by taking the rasterization results of the ArcGIS software as reference. Table 3 lists the area accuracy results for the

Fig. 6. Study area.

Table 1
The accuracy of rasterization in various land use types by the proposed parallel algorithm under 20 m spatial scale (classification accuracy).

Land use type	ArcGIS rasterization (reference)	Parallel rasterization	Area difference	Classification accuracy (%)
Paddy fields	7,080,997	7,081,321	324	99.99
Irrigated land	22,118	22,093	25	99.88
Dry land	873,431	873,440	9	99.99
Garden plots	301,605	301,404	201	99.93
Forest land	15,418,648	15,418,607	41	100.00
Pasture land	634	633	1	99.84
Agricultural land for facilities	12,506	12,535	29	99.76
Rural roads	1,474	1,482	8	99.45
Pond surface	900,171	900,254	83	99.99
Land for farmland water conservancy	27,427	27,454	27	99.90
Areas of cities and town	1,006,040	1,006,248	208	99.98
Residential quarters in rural areas	2,458,507	2,458,407	100	99.99
Mining land	94,769	94,666	103	99.89
Railways	31,520	31,520	0	100.00
Highways	1,87,802	1,87,592	210	99.88
Civil airports	9,639	9,646	7	99.92
Harbors and wharfs	842	840	2	99.76
Reservoir surface	135,031	135,189	158	99.88
land for water-control structures	50,447	50,333	114	99.77
Scenic spots	61,435	61,355	80	99.87
River surface	480,713	480,683	30	99.99
Lake surface	25,127	25,113	14	99.94
Beaches and flats	100,059	100,078	19	99.98
Natural reserved area	266440	266407	33	99.98
Total	29,547,382	29,547,300	1,826	99.99

two parallel algorithms. Only the area accuracy at 500 m resolution is 99.97%, whereas other area accuracies reached 100.00% for the parallel scanline algorithm.

The detailed comparisons of rasterization results from the parallel scanline algorithm and from ArcGIS in various spatial scales are shown in Fig. 8. Based on these experimental results, we can see that the parallel cell rasterization algorithm gets higher accuracy than the parallel scanline rasterization algorithm. Both of them are almost identical to the results of the ArcGIS rasterization, thus leading to a very high accuracy.

5.3. *Efficiency analysis*

5.3.1. *Relationship of the number of processors and speedup*

The computing time of the parallel algorithm is the basic performance measurement. If the number of processors is 1, the serial processing is performed; otherwise, the parallel algorithm is used. This experiment compares the execution times of serial processing and the parallel algorithm, as shown in Fig. 9. The execution time of the serial processing was 42.775 s. When using less than 12 processors, the execution time of the parallel

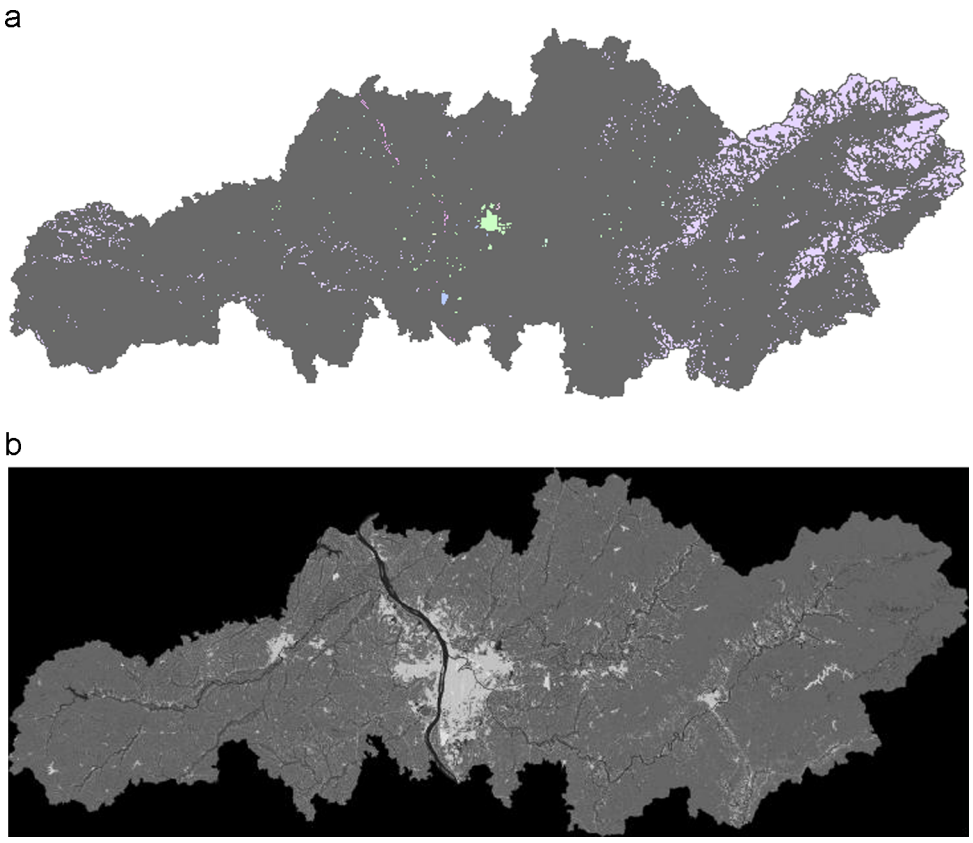

Fig. 7. (a) Vector polygons, (b) rasterization results of (a).

Table 2
The classification accuracy of the proposed parallel algorithm in different raster gird scales.

Raster grid size (m)	ArcGIS rasterization (reference)	Parallel cell rasterization			Parallel scanline rasterization		
		Total of Raster grids	Grid difference	Classification accuracy (%)	Total of Raster grids	Grid difference	Classification accuracy (%)
5	472,758,808	472,758,806	4,769	100.00	472,758,809	10,769	100.00
20	29,547,382	29,547,395	568	100.00	29,547,300	1,826	99.99
50	4,727,561	4,727,568	378	99.99	4,727,550	1,053	99.98
100	1,181,866	1,181,869	176	99.99	1,181,872	500	99.96
500	47,268	47,256	98	99.98	47,283	625	98.68

Table 3
The area accuracy of rasterization in different raster gird scales by the proposed parallel algorithm.

Raster grid size (m)	ArcGIS rasterization (reference)	Parallel cell rasterization			Parallel scanline rasterization		
		Total of Raster grids	Area difference	Area accuracy (%)	Total of Raster grids	Area difference	Area accuracy (%)
5	472,758,808	472,758,806	2	100.00	472,758,809	1	100.00
20	29,547,382	29,547,395	13	100.00	29,547,300	82	100.00
50	4,727,561	4,727,564	3	100.00	4,727,550	11	100.00
100	1,181,866	1,181,872	6	100.00	1,181,872	6	100.00
500	47,268	47,261	7	99.99	47,283	15	99.97

algorithm improves as the number of processors increase. The execution time reaches its best value (9.508 s) when 12 processors are used. When the number of processors is greater than 12, the operating time gradually increases and eventually reaches a plateau. The I/O time and the computation time are shown in Figs. 10 and 11, respectively. The I/O operation time improves when the number of processors is less than 12 and increases as more processors are used. Computation time improves consistently as the number of processors increase. However, the minimum time achieved by the parallel cell algorithm is 69.462 s when the processor number is 12, while the time cost of the serial algorithm is 454.384 s. This huge difference of time cost is the

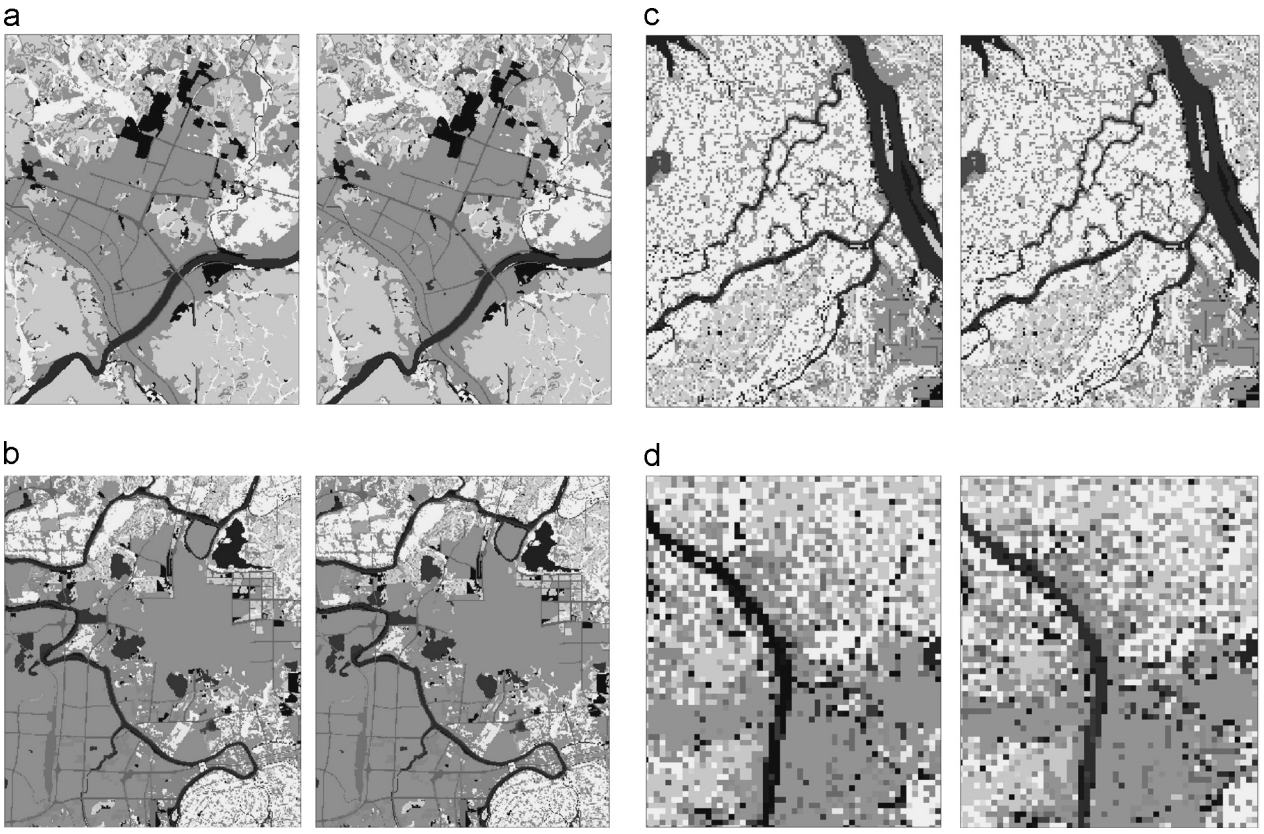

Fig. 8. Comparison between results of the proposed parallel algorithm (left) and ArcGIS software (right) in different spatial scales. (a) 20m*20m, (b) 50m*50m, (c) 100m*100m, (d) 500m*500m.

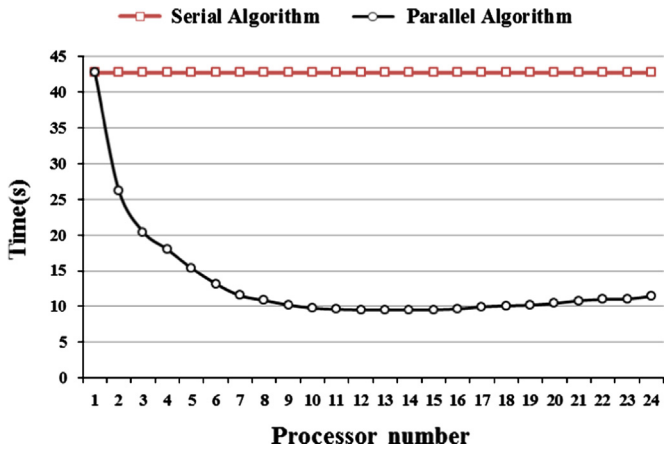

Fig. 9. Comparison of the execute time between the parallel algorithm and serial algorithm.

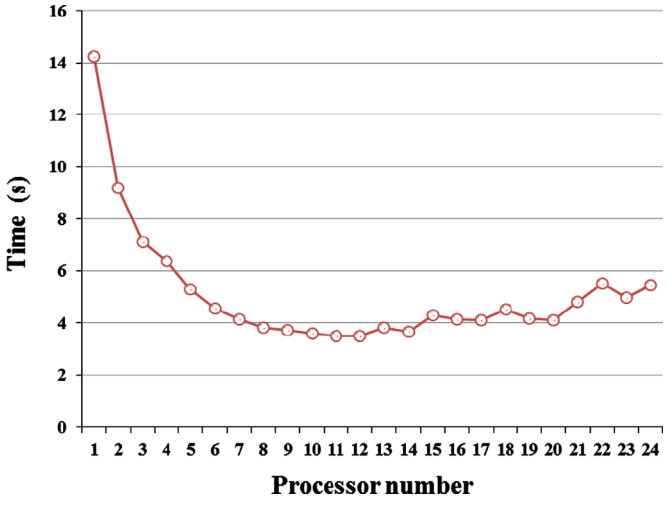

Fig. 10. The cost time of I/O for parallel scanline algorithm.

reason that we chose scanline algorithm as the parallel architecture, even though the cell algorithm achieves higher rasterization accuracy.

Fig. 12 shows the change of speedup with the parallel scanline rasterization algorithm, when the number of processors increase. The speedups show that good parallel efficiency is maintained. The speedup reaches its maximum value (4.496) when the number of processors is 12. As the number of processors increases, the communication cost also increases, and the load on different processors become imbalanced; therefore, the speedup tends to decline.

In theory, the best speedup should be directly related to the number of processors in a parallel environment. Our parallel environment had 16 processors, and we should have achieved the best speed-up at 16 processors. However, the algorithm achieved the best speedup between 11 and 15 processors, and the results could be attributed to system overhead and instability.

5.3.2. *Relationship of data partition pattern and speedup*

The experiment compares the effect of two partition patterns (row-partition and column-partition) used with this parallel algorithm. Fig. 13 shows the comparison diagram of the speedup, which shows that the maximum speedup of the row-partition pattern is significantly bigger than that of the column–partition

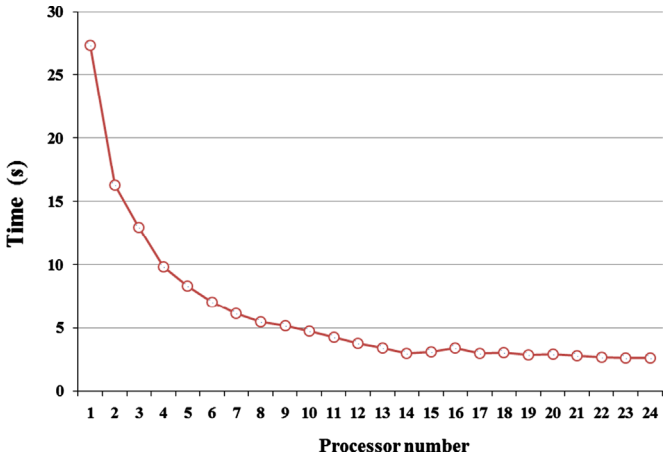

Fig. 11. The computation time of parallel scanline algorithm.

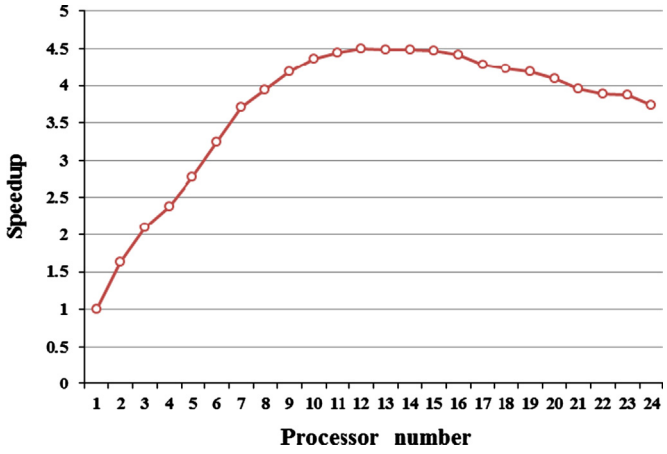

Fig. 12. The change of speedup with the increase of processor numbers.

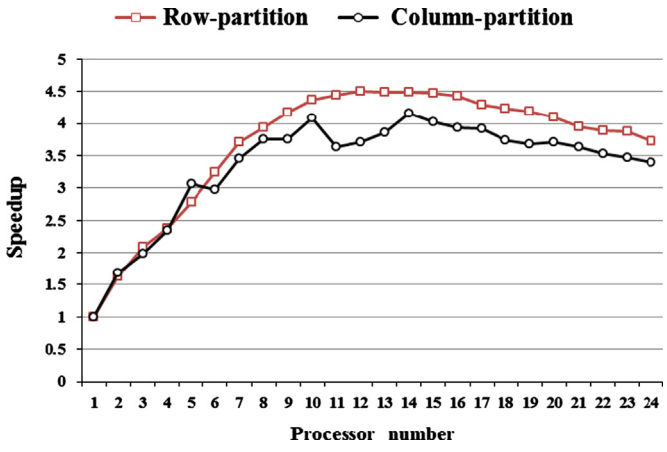

Fig. 13. Comparison of speedup change between row-partition and column-partition in this parallel algorithm.

pattern. The speedup change of the row-partition pattern is generally smoother, as the number of processors changes. The reason is that the raster is recorded by line or row, which means that, with the row-partition pattern, the reading and writing are performed directly on the original data, whereas, with the column-partition, the reading and writing are performed in a segmented way. As a result, speedup is strongly related to the pattern of data partition and the correctness of the partition pattern depends on how the data is stored.

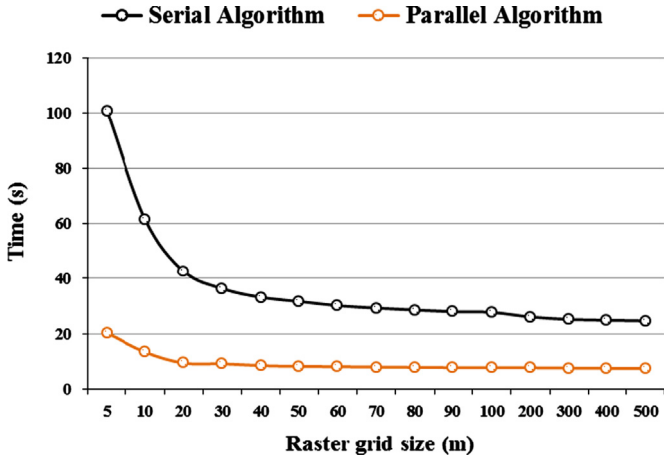

Fig. 14. Comparison between operation time of the serial algorithm and minimum operation time of the parallel algorithm in different raster grid sizes.

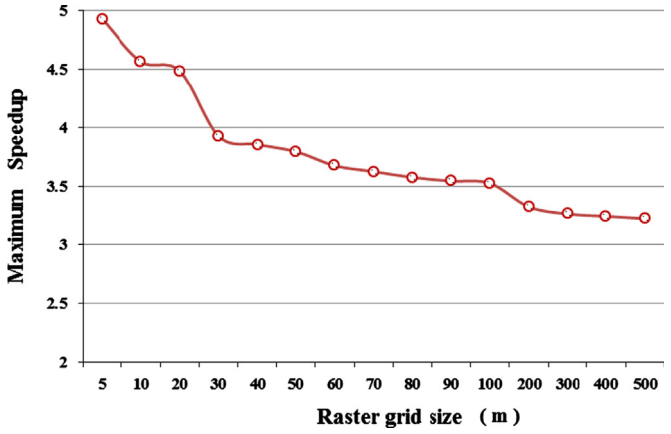

Fig. 15. The relationship between maximum speedup and raster grid size.

5.3.3. Relationship of raster grid size and speedup

Fig. 14 shows the operating time of the serial algorithm and the minimum execution time of the proposed parallel algorithm. Under serial algorithm, the operating time dramatically decreases with the increase of the raster size, especially with sizes from 5 m to 100 m. This proves that rasterization time using the proposed algorithm is closely related to the raster cell size. Under a parallel environment, the minimum execution time with different raster grid sizes decreases with the increase of the grid cell size and the change is generally slow. Fig. 15 shows the relationship between the maximum speedup and the raster grid sizes. The speedup decreases with the increase of the raster grid size, especially with sizes from 5 m to 100 m. Above 100 m, the change becomes slow and tends to be balanced. We attempted to select a grid cell size that was smaller than 4 m; however, the processing load exceeded the capacity of the processors. The reason for the memory overflow is that the system cannot satisfy the memory allocation for the raster blocks. As the grid size decreases, the memory allocation increases, and a grid size of 4 m is the threshold for our parallel environment. Based on this experiment, we can predict that the smaller the grid cell size is, the greater the speedup is.

6. Conclusion

This paper proposes a parallel scanline algorithm for rapid rasterization of massive geographic vector data. The following conclusions were reached based on our experimental analysis.

(a) Massive geographic vector data (about 0.7 million polygons) used in the experiment were effectively processed, dramatically reducing the data transformation time and achieving good speedup.
(b) In comparison to the rasterization results from the ArcGIS software, the parallel algorithm leads to almost identical results, thus ensuring a high accuracy.
(c) The speedup of this algorithm can be optimized by selecting the number of processors, the data partition pattern, and the raster grid size.

However, the parallel efficiency of this algorithm is related to the spatial distribution of the vector polygons. A better scheduling strategy might achieve greater parallel efficiency; therefore, this requires further study and experiment. Further research is needed on finding the right balance among raster grid size, number of processors, and speedup, in order to make full use of the existing hardware architecture to meet the specific application requirements.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (GrantNo.41001238), the National 863 Project of China (GrantNo.2011AA120301), the gons, are handled based on the proposed systematic solution theNational Key Technology R&D Program of China (GrantNo.2012BAH28B02). Sincere thanks are given to Mr. ZHOU Chen and Mr. YANG Kang for technical assistances. Many thanks are given for the comments and contributions of anonymous reviewers and members of the Editorial team.

References

Congalton, R.G., 1997. Exploring and evaluating the consequences of vector-to-raster and raster-to-vector conversion. Photogrammetric Engineering & Remote Sensing 63 (4), 425–434.
Correa, W.T., Klosowski, J.T., Silva, C.T., 2003. Out-of-core sort-first parallel rendering for cluster-based tiled displays. Parallel Computing 29 (3), 325–338.
Goodchild, M.F., 2011. Scale in GIS: an overview. Geomorphology 130 (2), 5–9.
Guan, Q.F., Clarke, K.C., 2010. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. International Journal of Geographical Information Science 24 (5), 695–722.
Gong, J.Y., Xie, J.B., 2009. Extraction of drainage networks from large terrain datasets using high throughput computing. Computers & Geosciences 35 (2), 337–346.
Healey, R., Dowers, S., Gittings, B., Mineter, M., 1998. Parallel Processing Algorithms for GIS [M]. Taylor and Francis, London, UK, pp. 253–263.
Hollander, M., Ritschel, T., Eisemann, E., Boubekeur, T., 2011. ManyLoDs: parallel many-view level-of-detail selection for real-time global illumination. Computer Graphics Forum 30 (4), 1233–1240.
Hormann, K., Agathos, A., 2001. The point in polygon problem for arbitrary polygons. Computational Geometry 20 (3), 131–144.
Jiménez, J.J., Feito, F.R., Segura, R.J., 2009. A new hierarchical triangle-based point-in-polygon data structure. Computers & Geosciences 35 (9), 1843–1853.
Kenneth, A.H., Coddington, P.D., James, H.A., 2003. Distributed frameworks and parallel algorithms for processing large-scale geographic data. Parallel Computing 29 (10), 1297–1333.
Lee, C.A., Gasster, S.D., Plaza, A., Chein-I, C., Bormin, H., 2011. Recent developments in high performance computing for remote sensing: a review. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 4 (3), 508–527.
Liao, S.L., Bai, Y., 2010. A new grid-cell-based method for error evaluation of vector-to-raster conversion. Computational Geoscience 14 (4), 539–549.
Maguire, D.A., Goodchild, M.F., Rhind, D.W., 1991. Geographical Information Systems: Principle. 1. Longman Group UK Limited, Essex, U.K, pp. 45–54.
Mariethoz, G., 2010. A general parallelization strategy for random path based geostatistical simulation methods. Computers & Geosciences 36 (7), 953–958.
McManus, D., Beckmann, C., 1997. Optimal static 2-dimensional screen subdivision for parallel rasterization architectures. Computers & Graphics 21 (2), 159–169.
Pineda, J., 1988. A parallel algorithm for polygon rasterization. Computer Graphics 22 (4), 17–20.
Popescu, V., Rosen, P., 2006. Forward rasterization. ACM Transactions on Graphics 25 (2), 375–411.
Roca, J., Moya, V., Gonzalez, C., Escandell, V., Murciego, A., Fernandez, A., Espasa, R., 2010. A SIMD-efficient 14 instruction shader program for high-throughput microtriangle rasterization. The Visual Computer 26, 707–719.
Schiele, S., Möller, M., Blaar, H., Thürkow, D., Müller-Hannemann, M., 2012. Parallelization strategies to deal with non-localities in the calculation of regional land-surface parameters. Computers & Geosciences 44, 1–9.
Subhlok, J.M., Stichnoth, J.M., O'Hallaron, D.R., Gross, T., 1993. Exploiting task and data parallelism on a multicomputer. ACM SIGPLAN Notices 28 (7), 13–22.
van der Knaap, W.G.M., 1992. The vector to raster conversion: (Mis)use in geographical information systems. International Journal of Geographical Information Systems 6 (2), 159–170.
Wang, X.L., Sun, Q.H., Jiang, C.S., 2006. An optimization algorithm for transferring vector to raster data based on minimized area error. Acta Geodaeticaet Cartographica Sinica 35 (3), 273–277. [in Chinese].
Xie, J.B., 2012. Implementation and performance optimization of a parallel contour line generation algorithm. Computers & Geosciences 49, 21–28.
Yang, S., Yong, J.H., Sun, J.G., Gu, H.J., Paul, J.C., 2010. A point-in-polygon method based on a quasi-closest point. Computers & Geosciences 36 (2), 205–213.
Zhou, C.H., Ou, Y., Yang, L., Qin, B., 2007. An equal area conversion model for rasterization of vector polygons. Science in China Series D: Earth Science 50 (S1), 169–175.