BCG - Boletim de Ciências Geodésicas - On-Line version, ISSN 1982-2170
http://dx.doi.org/10.1590/S1982-21702014000200015

OPTIMIZATION APPROACHES TO MPI AND AREA MERGING-BASED PARALLEL BUFFER ALGORITHM

Estratégias de otimização de algoritmos de armazenamento paralelo baseados em fusão de área e MPI

JUNFU FAN[1, 2, 3]
MIN JI[4(1)]
GUOMIN GU[5]
YONG SUN[4]

[1]State Key Laboratory of Resources and Environmental Information System, Institute of Geographic and Nature Resources Research, Chinese Academy of Sciences, Beijing 100101, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China
[3]School of Architectural Engineering, Shandong University of Technology, Zibo 255049, China
[4]College of geomatics, Shandong University of Science and Technology, Qingdao 266510, China
[5]Computer Science and Technology College, Zhejiang University of Technology, Hangzhou 310023, China
fanjf@lreis.ac.cn ; jamesjimin@126.com

ABSTRACT

On buffer zone construction, the rasterization-based dilation method inevitably introduces errors, and the double-sided parallel line method involves a series of complex operations. In this paper, we proposed a parallel buffer algorithm based on area merging and MPI (Message Passing Interface) to improve the performances of buffer analyses on processing large datasets. Experimental results reveal that there are three major performance bottlenecks which significantly impact the serial and parallel buffer construction efficiencies, including the area merging strategy, the task load balance method and the MPI inter-process results merging strategy. Corresponding optimization approaches involving tree-like area merging strategy,

(1)Corresponding author

the vertex number oriented parallel task partition method and the inter-process results merging strategy were suggested to overcome these bottlenecks. Experiments were carried out to examine the performance efficiency of the optimized parallel algorithm. The estimation results suggested that the optimization approaches could provide high performance and processing ability for buffer construction in a cluster parallel environment. Our method could provide insights into the parallelization of spatial analysis algorithm.
Keywords: Area Merging; Parallel Buffer; Task Partition; Vertex Accumulation Effect; Tree-like Merging; MPI.

RESUMO

Na construção de uma área de influência, o método de dilatação baseada em rasterização inevitavelmente introduz erros, e o método de dupla linha paralela envolve uma série de operações complexas. Neste artigo, propõe-se um algoritmo de armazenamento paralelo baseado na fusão de área e MPI (Interface de transferência de Mensagem) para melhorar o desempenho de análise de armazenamento, no processamento de grandes conjuntos de dados. Os resultados experimentais revelam que há três grandes gargalos de desempenho que impactam significativamente a eficiência de construção de armazenamentos seriais e paralelos, incluindo a estratégia de fusão de área, a tarefa / método do balanceamento de carga e os MPI resultantes da estratégia de fusão. Para superar esses gargalos, são sugeridas abordagens de correspondência de otimização envolvendo a estratégia de fusão em árvore, um método orientado de partição do número de vértices em tarefas paralelas e uma estratégia de fusão dos inter-processos. Os experimentos foram realizados para examinar a eficiência do algoritmo paralelo de otimização. Os resultados estimados sugerem que as abordagens de otimização podem fornecer alto desempenho e capacidade de processamento para a construção de armazenamento em um ambiente paralelo agrupado. Esse método pode fornecer sugestões sobre a paralelização de algoritmos de análise espacial.
Palavras-chave: Fusão de Área; Armazenamento Paralelo; Tarefa de Partição; Efeito de acumulação de Vérices; Fusão em Árvore; MPI.

1. INTRODUCTION

Rapidly expanded spatial datasets has brought unprecedented pressure to existing computational resources and the traditional analysis algorithms in geoscience. With the advancement in computer hardware technology and the development of parallel programming models, high performance computation has become an important issue for analyzing, processing, and visualizing massive geo-spatial data (TURTON and OPENSHAW, 1998 and 2000; CLARKE, 2003; HAWICK et al., 2003). High-performance computation can be implemented via many ways, such as multi-core parallelization, cluster parallelization, graphic processing unit (GPU) acceleration, and hybrid parallelization (LIN and SNYDER,

2009; McKENNEY et al., 2011), which all depend on the paradigm of parallelism (BARNEY, 2012). In a cluster computing environment, most of proposed parallelized spatial analytical algorithms are based on the framework of the message passing interface (MPI). The two major modes for the realization of parallel computation are data decomposition and task partition (GRAMA et al., 2003), which correspond to spatial data division and pipeline parallel processing, respectively. Data division strategies for parallel spatial analysis algorithms with topological relations have been extensively discussed (SLOAN et al., 1999; MINETER and DOWERS, 1999; DARLING et al., 2000). For instance, a parallel task partition approach based on the partition of geometries has been designed and implemented (MINETER and DOWERS, 2000), and an idea of software stratifying at a low level has been proposed to encapsulate the complexity and reuse the codes of parallel algorithms (MINETER and DOWERS, 2000). Furthermore, Mineter (2003) presented a parallel vector spatial analysis platform called the TSO (Topology-Stitching-Output) software framework. This approach were based on the NTF data model, containing topological information allowing complex topology to be created and checked in parallel task partition and result sewing. However, the processing and maintenance of topological information are time consuming for large dataset.

In GIS (Geographical Information System), a buffer is defined as a zone around a map feature measured in units of distance or time (ESRI, 2013). As an important function in map information retrieval, comprehensive spatial analysis, and processing in GIS, buffer analysis solves the problem of proximity and represents an influence extent or service extent (WU, 1997). Buffer analysis algorithm is widely used in many geo-spatial fields, such as spatial data query, hybrid overlay analysis of vector and raster data, thematic mapping, and so on.

Most previous studies of buffer algorithms focused on the double-sided parallel line generation algorithm with a self-intersected polygon processing model and the dilation method by means of rasterization and vector boundary extraction. Zalik et al. (2003) elaborated an algorithm for asymmetric segment buffer generation based on the idea of sweep line through the following steps: creating basic geometric outlines, identifying intersection points between them, construct rings, and determining spatial relationships amongst the rings. Based on Zalik's algorithm, an algorithm for buffer creation and result area merging has been implemented using the sweep line approach and vector algebra (BHATIA et al., 2013). Despite their precision, the aforementioned algorithms require complex computation and spatial relation identification, and their realization is complicated. In response, Li and Du (2005) proposed a buffer creation algorithm based on a dilation algorithm. Essentially, the idea of rasterization is used to simplify vector buffer creation, and the target buffer is created by extracting the boundaries of rasterized geometries, which are expanded according to certain window sizes and rules. However, these algorithms based on the extraction of rasterized boundaries

result in many errors. In research and engineering applications, buffer analysis algorithms also face efficiency limitations caused by large dataset.

Area merging can be achieved through polygon clipping algorithms, which have been intensively studied, and many algorithms have been proposed (SUTHERLAND and HODGMAN, 1974; WEILER and ATHERTON, 1977; LIANG and BARSKY, 1983). The currently recognized efficient algorithms that can process arbitrary polygon clipping within a limited amount of time include Vatti's algorithm (VATTI, 1992) and Greiner-Hormann's algorithm (GREINER and HORMANN, 1998), which with similar performances. The Vatti's algorithm supports clipping between polygons with any number of edges and in any shape (e.g., self-intersection with islands and/or holes). Murta (1998) then modified Vatti's algorithm to overcome the problem that horizontal edges could not be processed properly. Based on Vatti's algorithm, we implemented area merging and avoided its performance bottleneck by using a divide-and-conquer method. Moreover, area merging was introduced into the buffer creation to replace the complex ring construction and spatial relationship processing. Thus, the buffer creation algorithm was simplified, and a parallel buffer analytical algorithm was implemented.

At present, there is little research on optimization approaches to parallel buffer algorithms under high-performance computation, and the buffer analytical tools provided by GIS software do not have satisfactory efficiency. Therefore, it is valuable to further explore and discuss parallel buffer algorithms and their optimization approaches under the background of big data. Firstly, a serial buffer construction algorithm based on area merging was proposed, and an optimization approach based on area merging and the divide-and-conquer method was proposed. The efficiencies of the optimized algorithm and ArcGIS™ Buffer tool were compared with or without dissolving of the buffer result polygons. Secondly, a parallel buffer analysis algorithm was developed on the basis of data parallelism, and its accelerating abilities under the above two conditions were analyzed. Thirdly, the operation of the parallel buffer analysis algorithm was analyzed to identify the possible performance bottlenecks, and corresponding optimization solutions were then proposed.

In this paper, the experiments were performed under the same hardware conditions. The results showed that the buffer creation algorithm based on area merging and optimized using the divide-and-conquer method was feasible and had some advantages over the general buffer algorithm. The optimized algorithm effectively improved efficiencies in buffer creation and result dissolving, and an ideal speedup ratio was obtained. Therefore, the optimizing approaches are feasible pathway to improve area merging-based serial and parallel buffer algorithms.

2. AREA-MERGING BASED BUFFER ALGORITHM

The area-merging buffer construction algorithm will be introduced in this section, and then a divide-and-conquer method-based buffer zone merge strategy will be described to improve the buffer algorithm. Based on the above work,

comparison experiments between our serial buffer algorithm and the ArcGIS™ buffer tool are conducted. The vertex accumulation effect and the optimization solution to it in the process of polygon merging will be discussed in this section too.

2.1 Construction of Buffering Zone

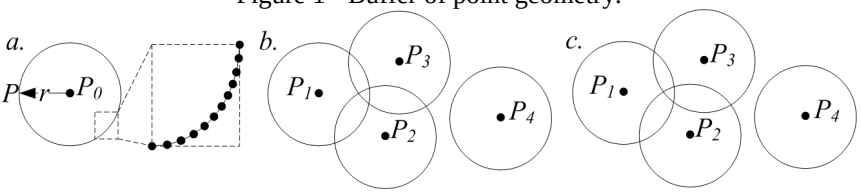
Buffer creation algorithms based on vector geometry are generally implemented through the following steps: creation of parallel lines, construction of rings, and processing of spatial relationships between intersected rings. The latter two steps involve numerous complex numerical computations (e.g., intersection calculation, vector computation, identification of included angles, and processing of self-intersection). These two steps may hardly be realized owing to many types of special cases should be handled. Therefore, we proposed a new method to replace these two steps for simplification of buffer creation by introducing the mature polygon clipping algorithm into vector buffer construction. Vatti's algorithm supports the polygon clipping operation and boolean operations (e.g., union and difference) in polygon overlay, so it is recognized as being able to process arbitrary polygon overlay within a limited time. In this study, area merging was realized on the basis of Vatti's algorithm. With the area merging approach, an unilateral buffer can be easily realized, and an asymmetric buffer can be realized using bilateral distinction and endpoint arc center translation. In this paper, only the most typical bilateral and symmetric buffer was discussed. The buffer creation algorithm based on area merging for the three basic types of geometries (point, polyline and polygon) is described as below:

(1) - For point object, when the radius (r) of a buffer is known, the buffer creation and construction methods for a point are the simplest because users can only draw an end-to-end ring with P_0 as the center and r as the radius. A point ($P(x,y)$) on the ring and the center ($P_0(x_P,y_P)$) satisfy the following equation:

$$\left(x - x_P\right)^2 + \left(y - y_P\right)^2 = r^2 \quad (1)$$

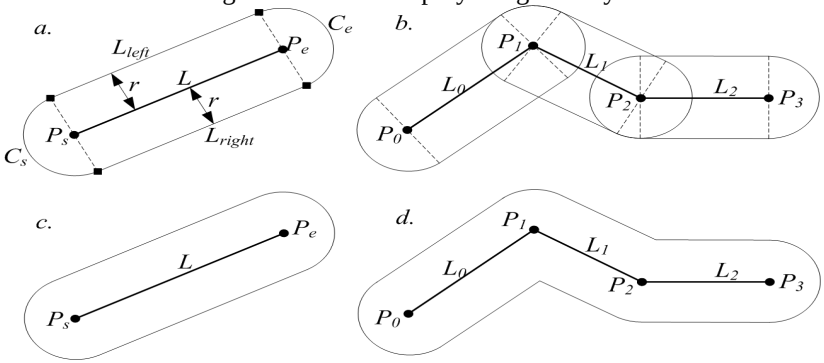
A point on the ring can be computed from Eq. (1), and the points are connected one-by-one to form a closed ring, namely a buffer zone with P_0 as the center and r as the radius (Figure 1-(a)). For a multi-point geometry, the results may be overlapped if the buffer of each point is created as per the rule for point geometry (Figure 1-(b)). Area merging can then be called to dissolve the overlapped areas, and the final result is shown in Figure 1-(c).

Figure 1 - Buffer of point geometry.



(2) - For polyline object, polyline geometry can be regarded as a group of end-to-end segments, and each segment is composed of a starting point and an ending point. Figure 2-(a) shows that the buffer for segment L is created in 3 steps as follows: 1) find L's two parallel lines (L_{left} and L_{right}) at two sides and with a distance of r; 2) draw two semi-arcs (C_s and C_e) with the starting point (P_s) and the end point (P_e) as the center, respectively; and 3) connect L_{left}, C_s, L_{right}, and C_e successively to construct a polygon (Figure 2-(c)), namely a buffer zone of segment L with r as the radius. If a polyline consisted of several segments (Figure 2-(b)), area merging will be used to dissolve all segment buffer zones to form a final buffer polygon result for it (Figure 2-(d)). A linear geometry composed of several independent polylines is called a multi-polyline, and its buffer zone can be created by dissolving the buffer zones of all its polylines.

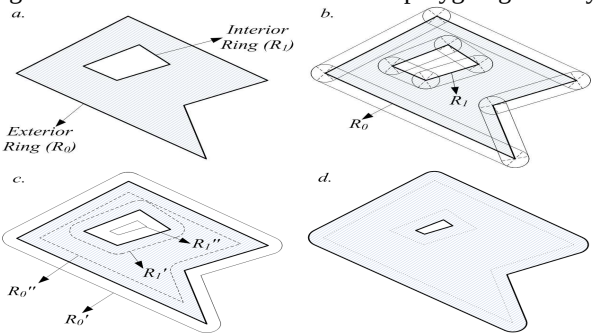
Figure 2 – Buffer of polyline geometry.



(3) - For polygon object, a polygon denotes a plane-shaped area enclosed by a group of closed polylines. As shown in Figure 3-(a), an enclosed polyline is also called a ring, which can be divided into an interior ring and an exterior ring according to the strike of the points constituting the ring. A simple polygon only contains one exterior ring and several interior rings, and a polygon that contains several exterior rings is called a multi-polygon. Buffer creation based on area merging for a simple polygon includes the following steps: decomposition, ring construction, dissolving, and deletion. First, the rings of a polygon are decomposed into a group of polylines, and independent buffer polygons are then constructed for each polyline as per the method in Figure 2. The polyline buffers are then dissolved, and the rings are selected and deleted. For instance, in creation of a bilateral buffer, the rules are as follows: the exterior ring created from the input polygon's exterior ring is reserved; the interior ring created from the polygon's interior ring is reserved; and other rings are deleted. Figure 3-(b) shows that the input polygon was composed of an exterior ring (R_0) and an interior ring (R_1), and all of the rings were split up at the starting point/end point. A buffer was created for each ring by using the buffer creation

algorithm for polyline resulting in 4 rings (R_0', R_0'', R_1' and R_1'') (Figure 3-(c)). Based on the conservation rule for result buffer polygon rings, the R_0' exterior ring created from the R_0 exterior ring as well as the R_1'' interior ring created from the R_1 interior ring were conserved, while R_0'' and R_1' were deleted. Finally, a result polygon was created as indicted by the shadow-filled region enclosed by the real line (Figure 3-(d)). The buffer of a multi-polygon can be created by dissolving the buffers of simple polygons. In creation of the interior ring's inside buffer, the buffer's radius exceeded the buffer range that the interior ring could hold if the buffer polygon's interior ring disappeared after dissolving, so all rings created from this interior ring should be discarded. The buffer created from an interior ring will never surpass the buffer created from the exterior ring that contains it.

Figure 3 – Buffer zone construction of polygon geometry.



2.2 Divide-and-Conquer Method for Area Merging

In this study, the widely validated Vatti's algorithm was used for area merging, and the time costs relation in area merging were statistically analyzed with different data volumes. Table 1 shows that the time cost of the UNION operation of Vatti's algorithm increased with the increasing number of polygons, but the performance was unsatisfactory. Thus, we statistically analyzed how the time cost in a single UNION operation changed with the increasing number of vertices, and regression analysis was also used.

Table 1 – Time costs of polygon merging implemented by Vatti's algorithm.

polygon number	merging time/s
400	3.283
2 000	135.198
4 000	1 650.110
6 000	4 727.272
8 000	7 034.313
10 000	13 595.301

With the increasing number of vertices, Figure 4 shows that the Vatti's algorithm showed a rapid growth similar to the power function. In dissolving a polygon set, we used a one-by-one 'snowball' strategy. With the progression of dissolving, the number of vertices contained in the polygons in each operation would inevitably be increased in most cases. In Figure 5, U_{AB} is the dissolved result from polygons *A* and *B*, and the number of vertices in U_{AB} was obviously larger than that of *B* or *A*. When Vatti's algorithm is used, U_{AB} will consume more time than *A* and *B*, which is called the vertex accumulation effect in area merging and is the major cause of the low efficiency shown in Table 1.

Figure 4 – Fitting curve of time costs of UNION operator of Vatti's algorithm.

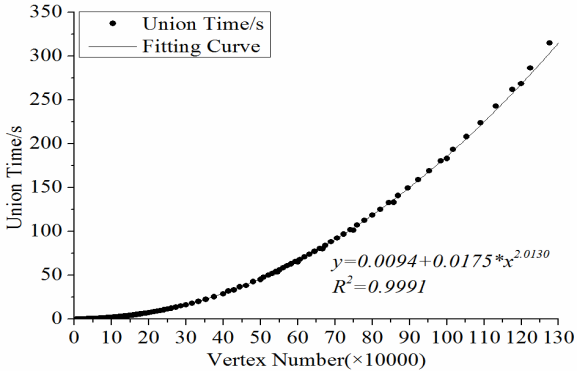
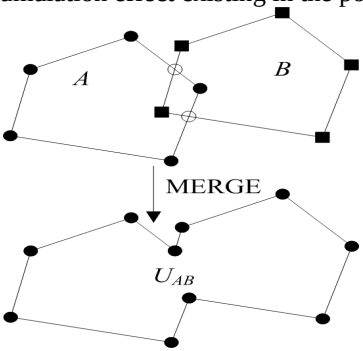


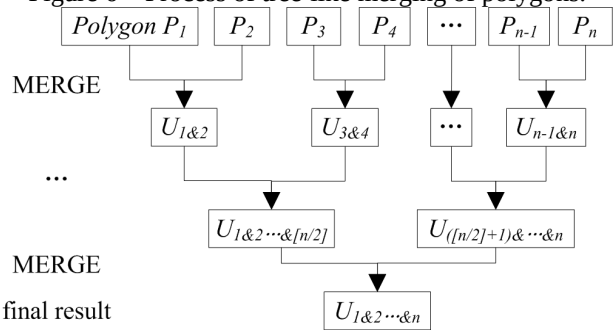
Figure 5 – Vertex accumulation effect existing in the polygon merge process.



A tree-like merging strategy for polygon sets was designed on the basis of the divide-and-conquer method (Figure 6), which well avoided the vertex accumulation effect and effectively shortened time costs in area merging. In the divide-and-conquer method, the original problem was divided into smaller scale sub-problems (n) in a similar structure as the original problem. The sub-problems were then

recursively solved, and their results were conquered to obtain the solution to the original problem (Thomas et al., 2011). This method is the foundation for many efficient algorithms, such as sorting algorithms (fast sorting and merge sorting), and the Fast Fourier Transform (FFT). The divide-and-conquer method has long been applied to solving geo-spatial problems, such as the divide-and-conquer algorithm for finding the closest point pairs proposed by Bentley and Shamos (1976) and the improved algorithm in calculation of the Delaunay triangular network based on the plane point set in the divide-and-conquer method (Dwyer, 1987). Under recursive mode, each recurring operation contains three steps as follows: divide, solve, and dissolve. The final result of area merging is not related to the internal dissolving order of the polygon set. Though the divide-and-conquer method is used mainly in recursive problems, it can also be used to dissolve a polygon set with a fixed number of polygons because the process and target of dissolving are explicit.

Figure 6 – Process of tree-like merging of polygons.



In tree-like merging, the polygons are first paired and dissolved, and the results of two adjacent pairs are then dissolved until only one polygon remains. Compared to 'snowball' merging, tree-like merging does not increase (or reduce) the number of calls of the UNION operator, but it accelerates computation by successfully avoiding the vertex cumulative effect. The data in Table 1 were also used for tree-like merging, and the statistics of time costs are listed in Table 2.

Table 2 – Time costs of tree-like merging of polygons.

polygon number	**dissolving time/s**
400	0.140
2 000	0.796
4 000	2.824
6 000	6.053
8 000	10.234
10 000	17.222

Tree-like merging showed obvious efficiency improvement because it effectively reduced the average number of vertices contained in the polygons in each function call of the UNION operator and, thus, successfully avoided the vertex accumulation effect hidden in the process of polygon set dissolving. The tree-like merging method was not only used in the creation of the single feature buffer polygon but also in the dissolving of intersected multi-feature buffer polygons.

2.3 Performance Analysis of Serial Algorithm

For the three major types of geometries of point, polyline, and polygon, the proposed algorithm was most representative in processing polyline geometries. To analyze the performances of the serial buffer algorithm with different data volumes, relevant experiments were conducted using real road network polyline datasets, and comparisons to the serial Buffer tool of ArcGIS™ 9.3/10.1 SP1 software on the same hardware platform were made.

Table 3 shows that when the intersected multi-feature buffer result polygons were not dissolved, the proposed algorithm showed a lower efficiency (0.7-1 lower) than the ArcGIS™ buffer tool. However, if the buffer result polygons were dissolved, ArcGIS™ 10.1 SP1 failed even after more than 10 h. Although some results were achieved by ArcGIS™ 9.3, the time cost was considerably greater than the proposed algorithm, and the proposed algorithm was more efficient with the increase of data volume. Therefore, the algorithm proposed in this study was feasible and could effectively overcome the severe performance bottleneck faced by GIS software during buffer analysis and result dissolving.

Table 3 – Time costs of serial polygon merging based buffer algorithm. [a]

polyline number	vertex number	time/s (merge none)		time/s (merge intersected)	
		our alg.	ArcGIS™ 9.3/10.1[b]	our alg.	ArcGIS™ 9.3/10.1
1 950	24 012	1.030	<1/<1	1.677	10.50/42.00
13 324	147 824	6.911	4.00/3.67	13.291	189.00/2 053.00
33 205	318 590	14.547	9.33/7.60	32.098	787.00/—[c]
45 850	470 825	21.544	13.67/10.50	57.885	1 243.50/—
108 414	1 067 682	52.097	31.33/25.00	128.545	3 676.50/—

[a] Experiments were carried out on Windows 7 Ultimate (x64).
[b] ArcGIS™ 10.1 with SP1 and background geo-processing switched off.
[c] Task did not get results within 10 hours and was canceled.

Some abnormalities were observed during the experiments. In ArcGIS™ 10.1, when the Parallel Processing Factors under ArcToolBox were set as 0, 1 or below 10% and when the created buffer result polygons were not dissolved, the CPU utilization rate of the Buffer tool was still maintained at 25-27%, but this rate was only 12% in the proposed algorithm. The computer's CPU was an Intel i7-2600, which is a quad-core CPU with hyper-threading function. In general, multiple

physical or virtual cores were engaged in computation when the CPU utilization rate was greater than 12%. Thus, ArcGIS^{TM} might conduct hyper-threading optimization or multi-core parallelization and optimization for its Buffer tool codes, which also explained why the proposed algorithm showed approximately doubled time costs compared to the ArcGIS^{TM} buffer tool.

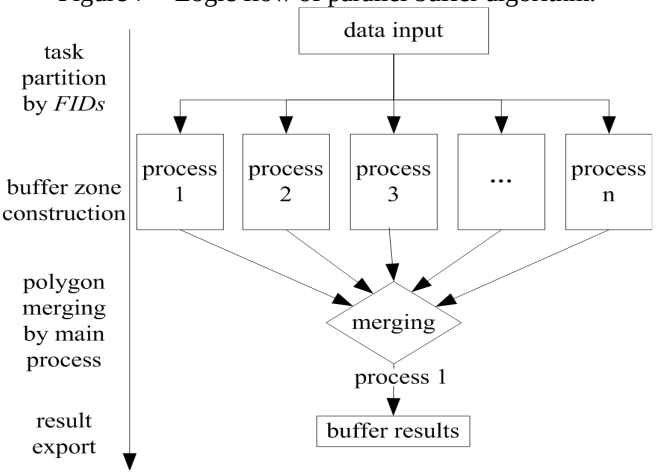
3. PARALLEL BUFFER ALGORITHM BASED ON MPI

The logical flow of the parallel buffer algorithm based on MPI will be described and the performance of the parallel algorithm will be studied by conducting of some parallel experiments in detail in this section. Experiments results reveal that the task load balancing and MPI inter-process results merging methods are two main bottlenecks of the parallel buffer algorithm.

3.1 Logical Flow of Parallel Buffer Algorithm

In practices, buffer analysis tools also face the challenge of large data volumes. Therefore, it is necessary to use high performance computation technology to design a parallel buffer algorithm and to study its optimization algorithms to overcome the problem of massive data in buffer analysis. The logic flow of parallel buffer analysis algorithms includes the following 4 stages: task division, parallelization to create buffer polygons, buffer area merging, and output of result data. These tasks correspond to decompose, compute, dissolve, and output in Figure 7.

Figure 7 – Logic flow of parallel buffer algorithm.



Data decomposition for parallelization based on a simple feature model uses feature identifier (FID) as the foundation, and it allocates the vector features into each computation node and concurrently creates a series of buffer polygon sets. The outputs from all processes are then delivered to the main process for dissolving. The

first three steps are the core of parallel buffer analysis algorithms, and the optimization of the proposed algorithm was discussed considering these three aspects. Data output may involve several detail conditions of application environments (e.g., vector data model, parallel file system, and parallel database). These detail conditions are largely different from each other and are not the core procedures in the parallel buffer analysis algorithm. These conditions consume little time compared with other steps; therefore, they were not discussed in this paper.

3.2 Performance Analysis of Parallel Buffer Algorithm

Based on the above logic flow, a parallel buffer analysis algorithm was implemented on the basis of the MPI program model and data parallelism, and it was tested using data of several groups of real linear road networks. Table 4 shows that the proposed algorithm could improve efficiency to a certain level. When buffer result polygons were not dissolved, the 4-process parallel computation could achieve efficiency as high as that of ArcGIS™. However, MPI and data parallelism did not bring buffer analysis algorithms with an ideal speedup ratio. With increased processes, parallel computation efficiency was reduced, indicating that parallel algorithms based on plain parallelism could be optimized further, which suggested that its bottleneck should be analyzed carefully and eliminated

Table 4 – Time costs of area merging-based parallel buffer algorithm[a].

polyline number	vertex number	time with different processes/s (dissolve none)				time with different processes/s (dissolve intersected)			
		1	2	3	4	1	2	3	4
1.950	24.012	1.342	1.209	1.161	1.288	1.426	1.225	1.091	1.255
13.324	147.824	8.959	6.070	5.059	4.667	11.191	8.046	8.280	7.217
33.205	318.590	19.723	12.964	11.211	10.092	26.802	21.248	18.881	17.756
45.850	470.825	26.797	17.671	16.338	14.976	56.901	52.758	55.062	52.350
108.414	1.067.682	68.113	39.048	34.747	33.324	126.907	105.749	109.755	109.927
134.145	1.482.071	86.318	49.955	44.208	43.016	243.110	214.180	211.997	224.911
269.450	3.231.870	192.465	106.241	93.741	84.229	494.808	368.687	418.435	442.850

[a] Experiments were carried out on Fedora 15 (Linux 2.6.38.6-26.rc1.fc15.x86_x64).

3.3 Bottlenecks of Parallel Buffer Algorithm

In achieving high performance using parallel computation, one inevitable problem is how to balance the loads among parallel tasks because all computation tasks can be completed within a similar time only under load balance, which is extremely important for MPI-based parallel buffer algorithms under cluster parallel environments as the cluster system's overall utilization rate can be improved only when the waiting time before the dissolve is reduced for MPI processes that finished early. We performed two parallel buffer analysis experiments for parallel task distribution using the FID-based data decomposition strategy, and we statistically

analyzed the time costs for the two procedures of buffer zone generation and dissolving with the largest velocity difference.

Table 5 shows that the numbers of features were evenly distributed among MPI processes and that certain parallel acceleration was achieved, but the numbers of vertices contained in the vector features were different among processes.

Table 5 – Differences of time costs between MPI processes (data partition by *FIDs*).

process /feature number	speed	time costs of single process/s		data volume of single process	
		buffering	tree-like merging	feature number	vertex number
4/134 145	fastest	9.284	9.073	33 536	256 908
	slowest	20.150	57.449	33 537	492 465
8/269 450	fastest	21.491	7.833	33 681	313 627
	slowest	37.772	91.226	33 683	547 461

The area merging based on Vatti's algorithm was sensitive to the number of vertices, and the buffer algorithm based on this operation was inevitably affected, which would cause large computation time differences among processes. The slowest process had a time cost that was 2.2 times that of the fastest process and a dissolving time that was 11.6 times that of the fastest one. Unreasonable data decomposition would result in a potential performance bottleneck for MPI algorithms; therefore, the premise for MPI inter-process load balance was to homogeneously decompose the parallel tasks under data parallelism mode, which is also an important direction for the optimization of parallel algorithms.

Based on the principle to reduce the mutual waiting time among MPI processes, there is also space for optimization and acceleration in result set merging after all processes are completed, which usually requires the redesign of a strategy to merge the MPI inter-process result sets. Table 5 shows the difference of computation time costs among different processes, especially when load balance cannot be achieved. As a result, the first finished process had to wait for the other unfinished processes. If the task of inter-process result merging is assigned to a single process (e.g., the main process in Figure 7), the single process can continue the task only after all processes are finished, which obviously reduces the parallel computation efficiency and thus becomes a performance bottleneck. In response, considering that the principle of MPI inter-process result merging is similar to that of tree-like area merging, the final target result is not associated with the order of merging between processes, and its result and process are all explicit. Thus, the final target result can also be optimized using the divide-and-conquer method. Therefore, at the process level, a tree-like merger strategy can be designed for MPI inter-process result sets to reduce the merging waiting time for inter-process result sets and to optimize and accelerate the parallel buffer algorithm.

4. APPROACHES TO OPTIMIZING THE PARALLEL BUFFER ALGORITHM

To overcome the bottlenecks introduced in section 3, a vertex amount-based parallel task partition strategy and a tree-like inter-process results merging method are proposed and described in this section.

4.1 Parallel Task Partition

The most straightforward method to process vector spatial data based on a simple feature model is to realize a parallel task partition through dataset division by the number of features. The principle of this method is easy. Suppose that the input data have *F* features and that a parallel environment contains *n* MPI processes, the number (*m*) of features that are distributed to each process are as follows when based on data decomposition:

$$m = \lceil F / n \rceil \text{ Or } m = \lfloor F / n \rfloor \tag{2}$$

This method can obtain uniform results when the dataset has uniform features, but this situation rarely occurs. Furthermore, the low level algorithm is sensitive to the volume of vertices holding the features, not to the number of features. In most cases, this method cannot obtain load balance; therefore, new data decomposition methods should be developed.

In response to this defect, we proposed a parallel task data decomposition method based on vertex number statistics because the UNION operator for parallel vector buffer results is sensitive to the number of vertices in polygons. For data decomposition, this method depends on the number of vertices contained in geometries. Suppose that a group of input data contains *N* vertices and a parallel environment contains *n* MPI processes, then each process is expected to be assigned with a group of vector features with *P* vertices as follows:

$$P = \lceil N / n \rceil \tag{3}$$

The number of features distributed into a process is no longer constant. However, the geometries cannot be split, and the total number of vertices P_i (i=1,2,3,…,*n*) should be values close to *P*. The data decomposition can be finished by reviewing the numbers of vertices for all vector feature geometries. This method is more time-consuming than the task partition method based on the FID of features, but the experiments revealed that the higher time cost for counting the amount of vertices is acceptable considering the performance improvement. The number of MPI processes was consistently 4. When the other experimental characteristics were held constant, each of the 7 groups of road network data with different data volumes

was divided based on the number of features and on the number of vertices. The contradistinction experimental results are listed in Table 6.

In Table 6, T_{FIDs} is the total time costs of parallel computation based on the number of features, and T_{points} is the total time costs of parallel computation based on the number of vertices. Moreover, T_{DP} is the time cost in data division based on the number of vertices, which is already contained in T_{points}.

Table 6 – Improvements by the method of point number-based data partition.

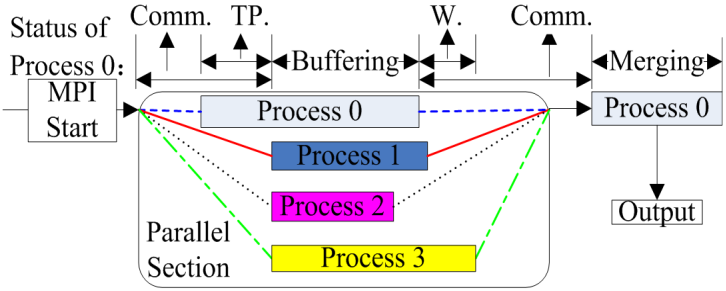
feature/point number	time costs (4 MPI processes)*T/s*		
	T_{FIDs}	T_{points}	T_{DP}
13 324/147 824	7.517	6.558	0.042
18 154/215 048	14.043	11.618	0.101
33 205/318 590	18.267	15.594	0.103
45 850/470 825	52.023	46.478	0.142
108 414/1 067 682	110.365	101.155	0.396
134 145/1 482 071	224.633	204.446	0.429
269 450/3 231 870	430.176	419.370	1.379

The results indicate that the partition method based on the number of vertices achieved a 10% higher performance at the expense of a 0.43% time consumption increase. Therefore, this method can improve computation efficiency for the parallel vector buffer algorithm.

4.2 Tree-Like Merging Between MPI Processes

When several parallel MPI processes are finished, the polygon result sets derived from all processes should also be determined for intersection and be dissolved. A simple method is to distribute all results to a single process (e.g., the main process shown in Figure 7) for area merging and output. The operation flow of this method is shown in Figure 8.

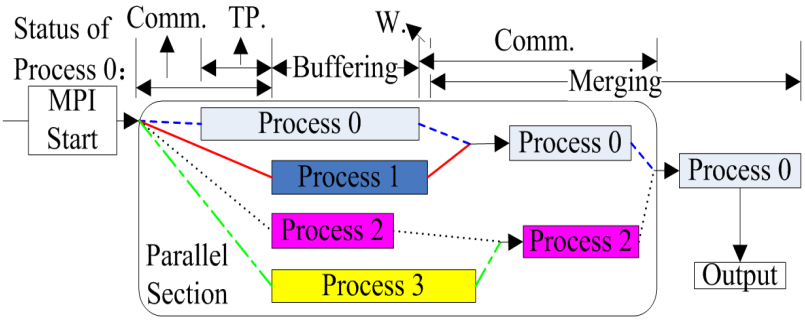
Figure 8 – Single-process merging flow of buffer results of 4 MPI processes.



Comm.: Communication; TP.: Task Partition; W.: Waitting

One evident defect of this method is that the single process responsible for results merging has to wait until all processes are finished to continue and finish the final merge process. Regarding the significant effect of tree-like merging, we proposed to design a new strategy for merging inter-process result sets, which accelerates computation by decreasing the inter-process waiting time. This process was called the MPI inter-process tree-like merging optimization strategy, and its work flow is shown in Figure 9.

Figure 9 - Tree-like merging flow of buffer results of 4 MPI processes.



Comm.: Communication; TP.: Task Partition; W.: Waitting

With the 4 MPI processes in Figure 9 as an example, the result will be preserved and processed by the 1st process when the 1st and 2nd processes are merged. When the 3rd and 4th processes are merged, the result will be preserved and processed by the 3rd process, followed by the results of the 1st and 3rd processes being merged again. In this way, the difficulty of developing the MPI program can be reduced by providing a tree-like merging pathway for predesigned MPI parallel processes. The parallel buffer algorithm with the above merging flow was implemented to compare the parallel buffer algorithm with a single process merging strategy. Seven groups of road network data with different data volumes were used, and the other characteristics were kept constant.

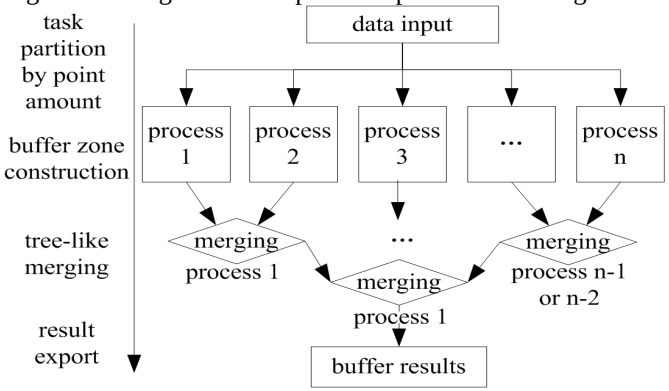
Table 7 shows that the optimization of tree-like merging in MPI processes can improve efficiency by 46.6% for parallel buffer analysis algorithms on average. With regard to the 4 MPI processes, the parallel speedup ratio was increased from 1.411 to 2.708, which indicated a significant effect. Therefore, this result suggested that the tree-like merging approach in the MPI inter-process polygon set shows a significant optimizing effect for parallel buffer analysis algorithms and shows certain practical values. The logic flow of the parallel buffer analysis algorithm based on this optimizing strategy is presented in Figure 10.

Table 7 – Time costs of parallel buffer algorithm optimized by tree-like merge strategy between MPI processes.

feature/point number	parallel time costs (4 MPI processes)/s		serial time costs	final speedup	efficiency improvements
	single process merge	tree-like merge			
13 324/147 824	6.432	5.503	11.288	2.051	16.9%
33 205/318 590	15.691	10.499	27.067	2.578	33.1%
45 850/470 825	46.747	21.909	57.271	2.614	53.1%
108 414/1 067 682	100.939	44.612	127.707	2.863	55.8%
134 145/1 482 071	198.670	78.781	243.979	3.097	60.3%
269 450/3 231 870	419.306	172.288	499.723	2.901	58.9%
1 329 758/12 471 234	1 685.036	878.841	2 509.740	2.856	47.8%
Average	—	—	—	2.708	46.6%

The optimization of the MPI inter-process tree-like merging can still be improved. For instance, the merging order is not preset, but a 'first finish first merge' mode is used. An evolution coefficient can be defined for each process, and the two earliest finishing processes are merged first. After each merging process, the evolution coefficient of one process is added by 1, and the other process is ended. In each merging step, only the processes with the same evolution coefficient are merged, unless the number of processes marked by a certain evolution coefficient is only 1. After all processes are merged, the results are finally merged and output by the process with the highest evolution coefficient. However, this method would greatly increase the complexity of inter-process communication and programming. Thus, this method would significantly increase the difficulty for developing MPI parallel programs; therefore, appropriate selection and rejection are necessary in practical applications, which should be further studied.

Figure 10 - Logic flow of optimized parallel buffer algorithm.



5. CONCLUSIONS AND FUTURE ISSUES

In this paper, a vector buffer generation algorithm based on the traditional segment buffer zone construction algorithm and the area merging approach was proposed. The algorithm simplified the process of buffer zone construction by introducing a mature polygon clipping algorithm to dissolve the buffer results of a single feature or several features, and the processing of complex spatial relationships during feature buffer creation was avoided. Moreover, the code complexity and coupling degree were reduced. For optimization of the buffer result dissolving, a divide-and-conquer method was used to overcome the bottleneck of the vertex accumulation effect in serial buffer algorithms. The efficiency of this method was lower than that of mature commercial GIS software when the buffer results of different features were not dissolved, but numerous experiments revealed that this method could finish buffer construction for a massive dataset with arbitrary geometries in a reasonable amount of time. In creating intersected buffer zones that should be dissolved, the proposed algorithm was far more efficient in serial computation than the ArcGIS^{TM} Buffer tool. Therefore, this buffer creation algorithm based on area merging has certain practical values.

Parallel computation is a feasible way to overcome the problem of increasingly larger spatial data volumes. Though the development of parallel algorithms is important, their optimization is also important for accelerating computation and scaling up the problems to be solved. In this paper, parallel buffer construction and a dissolving algorithm were implemented on the basis of a serial buffer algorithm and the MPI parallel programming model. We elaborated the two possible performance bottlenecks in the parallel buffer algorithm that caused low efficiency, and we proposed specific solutions, including the parallel task partition approach based on the number of vertices for parallel task load equilibrium and the tree-like merging approach to MPI inter-process result polygon sets. In the case of 4 MPI processes, the results showed that the new parallel task partition strategy improved performance by 10% at a 0.43% time cost increase. Moreover, the inter-process tree-like merging method improved efficiency by 46.6%, and the parallel speedup ratio was increased from 1.4 to 2.7, which indicated a significant effect. Therefore, we suggest that the two optimization approaches mentioned above could effectively improve performance for buffer construction and are feasible for the parallel optimization of buffer analytical algorithms. The two approaches provide certain reference values for the parallelization and optimization of other vector analysis algorithms in GIS.

In addition, the more reasonable ‘first finish first merge’ mode can be used in merging MPI inter-process result sets. Considering the hypothesis that buffer result polygons of adjacent vector features are more likely intersected, the relationships of adjacent vector features should be considered in parallel task division. Other rules (e.g., Hilbert spatial division curves coordinated with the number of vertices of features) can be used to obtain a better optimization approach. The above problems were not discussed in this paper and will be studied further.

ACKNOWLEDGEMENTS

This study was supported by National Key Technology R&D Program (No. 2011BAH06B03 and No. 2011BAH24B10, and No. 2012BAH27B04) and Research Fund for the Doctoral Program of Higher Education of China (No. 20113718110001). Additional supports were provided by Chinese Academy of Sciences (No. KZZD-EW-07). The authors thank Dr. Jorge Pimentel Cintra for his help in the Brazilian Portuguese translation.

BIBLIOGRAPHICAL REFERENCES

BARNEY B. *Introduction to Parallel Computing*. 2012. Accessed 10/01/2013. https://computing.llnl.gov/tutorials/parallel_comp/.

BENTLEY J. L., SHAMOS M. I. Divide-and-conquer in multidimensional space. In: *Proceedings of the eighth annual ACM symposium on Theory of computing (Proceeding STOC '76)*, Proceedings, New York: ACM Press, p. 220-230, 1976.

BHATIA S., VIRA V., CHOKSI D. An algorithm for generating geometric buffers for vector feature layers, *Geo-spatial Information Science*, 16(2), p. 130-138, 2013.

CLARKE K. C. Geocomputation's Future at the Extremes: High Performance Computing and Nanoclients. *Parallel Computing*, 29(10), p. 1281-1295, 2003.

CORMEN T., LEISERSON C., RIVEST R. Section 2: Sorting and Order Statistics, *Introduction to Algorithms (Second ed.)*, MIT Press, Cambridge, MA USA, p. 123-196, 2001.

DARLING G.J., SLOAN T.M., MULHOLLAND C. The input, preparation and distribution of data for parallel GIS operations. In: *Proceedings of Euro-Par 2000, Lecture Notes in Computer Science*, 1900, p. 500–505, 2000.

DWYER R. A. A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations. *Algorithmica*, 2(2), p. 137–151, 1987.

ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE, INC. *Buffer - GIS Dictionary*. 2012. Accessed 01/03/2013. http://support.esri.com/en/knowledgebase/GISDictionary/term/buffer.

GRAMA A, GUPTA A, KARYPIS G,. Chapter 3: Principles of Parallel Algorithm Design, *Introduction to Parallel Computing (Second Edition)*. Pearson Education Limited, p. 86-143, 2003.

GREINER G., HORMANN K. Efficient clipping of arbitrary polygons. *ACM Transactions on Graphics*, 17(2), p. 71-83, 1998.

HAWICK K.A., CODDINGTON P.D., JAMES H.A. Distributed frameworks and parallel algorithms for processing large-scale geographic data. *Parallel Computing*, 29(10), p. 1297-1333, 2003.

LI K., DU L. An Algorithm of Buffer Zones Based on Algorithm of Dilation. *Journal of Institute of Surveying and Mapping*, 22(3), p. 229-231, 2005. (*in Chinese*)

LIANG Y. BARSKY B. A. An analysis and algorithm for polygon clipping. *Communications of the ACM*, 26(11), p. 868-877, 1983.

LIN C., SNYDER L. *Principles of Parallel Programming*. Addison-Wesley Publishing Company, Reading, MA, USA, 352pp, 2009.

MCKENNEY M., LUNA G D., HILL S. Geospatial overlay computation on the GPU. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Proceedings, ACM. NY, USA, p. 473-476, 2011.

MINETER M. J., DOWERS S. Parallel processing for geographical applications: A layered approach. *Journal of Geographical Systems*, 1(1), p. 61-74, 1999.

MINETER M. J., DOWERS S. Towards a HPC Framework for Integrated Processing of Geographical Data: Encapsulating the Complexity of Parallel Algorithms. *Transactions in GIS*, 4(3), p. 245-262, 2000.

MINETER M. J. A software framework to create vector-topology in parallel GIS operations. *International Journal of Geographical Information Science*, 17(3), p. 203-222, 2003.

MURTA A. *A Generic Polygon Clipping Library*, 1998. Accessed 01/11/2012. http://www.cs.man.ac.uk/~toby/alan/software/gpc.html.

OPENSHAW S., ABRAHART R J. *GeoComputation*. Tylor & Francis Ltd., New York, USA, 432pp, 2000.

SLOAN T. M., MINETER M.J., DOWERS S., Partitioning of Vector-Topological Data for Parallel GIS Operations: Assessment and Performance Analysis. In: *Proceedings of Euro-Par'99 Parallel Processing, Lecture Notes in Computer Science*, Proceedings, 1685, p. 691-694, 1999.

SUTHERLAND I. E., HODGMAN G. W. Reentrant Polygon Clipping. *Communications of the ACM*, 17(1), p. 32-42, 1974.

TURTON I., OPENSHAW S. High-performance computing and geography: developments, issues, and case studies. *Environment and Planning: A*. 30, p. 1839-1856, 1998.

VATTI B. R. A Generic Solution to Polygon Clipping. *Communications of the ACM*, 35(7), p. 56-63, 1992.

WEILER K., ATHERTON P. Hidden surface removal using polygon area sorting. In: *Proceedings of the SIGGRAPH'77*, Proceedings, New York: ACM Press, p. 214~222, 1977.

WU H. H. Problem of Buffer Zone Construction in GIS. *Journal of Wuhan Technical University of Surveying and Mapping (WTUSM)*, 22(4), p. 358-366, 1997. (*in Chinese*)

ZALIK, B., ZADRAVEC, M., CLAPWORTHY, G. Construction of a Non-Symmetric Geometric Buffer From a Set of Line Segments. *Computers & Geosciences*, 29(1), p. 53–63, 2003.

(Recebido em setembro de 2013. Aceito em janeiro de 2014).