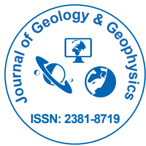
Huang et al., J Geol Geosci 2013, 2:2
DOI: 10.4172/2329-6755.1000115



Journal of Geology & Geosciences

Research Article Open Access

Parallel Buffer Generation Algorithm for GIS

Xiaomeng Huang[1]*, Tian Pan[2], Huabin Ruan[1], Haohuan Fu[1] and Guangwen Yang[1]

[1]*Ministry of Education Key Laboratory for Earth System Modeling, and Center for Earth System Science, Tsinghua University, Beijing, 100084, China*
[2]*Department of Computer Science, Yale University, 51 Prospect Street, New Haven, CT 06511, USA*

Abstract

A buffer generation algorithm that identifies areas of a given distance surrounding geographic features is one of the most frequently used functions in GIS. With the increase of scale and precision in geographic data, the efficiency of the buffer generation algorithm has been of great concern. This study presents a novel integrated solution consisting of a points-based, load-balanced method and a binary union tree method to accelerate the buffer generation. By comparing several parallel candidates, the experimental results show that our new parallel algorithm achieves greater performance and scalability, and its speed increases by 21 times with 32 processes.

Keywords: GIS; Buffer generation; Parallel computing

Introduction

Over the past two decades, GIS has become an important technology, deployed across a range of application areas to investigate and understand our world. With the growing amount of data, the processing time required to perform GIS analysis has increased. Although computational performance has been unhindered thus far, the large data volumes and the growing sophistication of analysis procedures suggest that performance will increasingly become a serious concern in GIS [1,2]. The traditional methods of performing GIS processing on desktop computers are quite insufficient in terms of performance. Parallel computing offers a potential solution to solve the problem. Massive numbers of parallel computers provide a huge amount of memory and computation power to accelerate GIS simulation and analysis. However, traditional GIS algorithms may not run effectively in a parallel environment, so the utilization of parallel technology is not entirely straightforward. Processing algorithms must be reworked to allow for this collaboration between computers. Buffer analysis is the core function of spatial analysis in GIS; thus, the method for generating buffer zones effectively plays a pivotal role in the software development of GIS. Buffer analysis is used to identify areas with a given distance surrounding geographic features. In other words, polygons are created at a certain distance around the input features and are stored in the database as geographic layers, e.g., rivers, roads, Contour lines, etc. Irrespective of the type of input geographic features, buffer polygons can delineate geographic spatial proximity. As one of the fundamental functions of GIS, buffer polygons have significant applications for environmental and ecological protection, decision support, automatic procession of structural geo-data, etc [3]. Buffer generation itself is computing-intensive and time-consuming. Large-scale geographic data have further curbed the speed of buffer generation. For these reasons, real-time reactions of the trends in modern GIS are not able to be met. This paper focuses on the current strategies used by scientists and engineers for the development of this crucial algorithm for parallel computing and GIS. An implementation case study involving a parallel buffer generation algorithm is included and is based around a parallelization GIS problem, which illustrates some of the principles involved in [4]. This paper proposed a number of methods, including a points-based load-balance and a binary union tree, to implement parallel optimization. The experimental results show that our parallel algorithm has good performance and scalability. The algorithms speed increases by 21-fold with 32 processes, and its efficiency is well-maintained with the increase in process number. The rest of the paper is organized as follows. The next para describes the work related to buffer generation for GIS, followed by introduction of the detailed algorithm of our parallel buffer generation and the experimental results and analysis, and conclusions are provided at the end.

Related Work

In general, the sequential buffer generation algorithm can fall into two categories: raster-based algorithms and vector-based algorithms [5]. The main difference between the two categories is that the former thickens entities in the raster and records a new border, while the latter sets up double parallel lines and circular arcs around the entities and then smoothes their borders. Wang et al. [6] analyzed the existing buffer generation algorithms systematically and re-viewed their characteristics, performance, and applicability, among other aspects.

Traditional raster-based algorithms are relatively easier to understand and to implement than vector-based algorithms. The earliest foundation of raster-based algorithms is laid by primitives of dilation and erosion in mathematical morphology, which can explain how pixels expand or contract the raster shape. Raster based algorithms dilate raster points, lines, areas, etc., to a target cell in the distance and then identify their borders to compute the buffer zones. The precision of buffer rendering relies on the resolution of the raster, which is measured by the size of the raster. However, high resolution raster exhaustively consumes memory resources. A trade-off between speed and resolution must be made. On the contrary, vector-based algorithms can handle the precision problem rather well. Nevertheless, they are more complex for dealing with curve intersection, arc-segment cutting and recombination, inclusion relation judgment, etc. A vector-based algorithm consists of three main steps: simple parallel lines are drawn for each line-segment, cusp-smoothing correction is performed, and finally, self-intersection is tackled. A large number of strategies have been proposed to optimize the second and the third procedures. For example, Elber et al. [7] compared offset curve approximation methods. Er et al. [8] defined and implemented a corridor rendering algorithm with variable leg buffer distances. A corridor is defined by a path and two distances for each leg to yield a buffered zone around the path.

***Corresponding author:** Xiaomeng Huang, Ministry of Education Key Laboratory for Earth System Modeling, and Center for Earth System Science, Tsinghua University, Beijing, 100084, China; Tel: +86-10-62798365 E-mail: hxm@tsinghua.edu.cn

Received February 21, 2013; **Accepted** April 27, 2013; **Published** May 23, 2013

Citation: Huang X, Pan T, Ruan H, Fu H, Yang G (2013) Parallel Buffer Generation Algorithm for GIS. J Geol Geosci 2: 115. doi: 10.4172/2329-6755.1000115

Copyright: © 2013 Huang X, et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Citation: Huang X, Pan T, Ruan H, Fu H, Yang G (2013) Parallel Buffer Generation Algorithm for GIS. J Geol Geosci 2: 115. doi: 10.4172/2329-6755.1000115

Wolff et al. [9] created an algorithm to calculate curved labels and to run in O(n2) time, where n is the number of points of a polyline. Ren et al. [10] took advantage of the Douglas-Peucker algorithm to remove non-characteristic points on curves and to extract characteristic points. Jiechen et al. [11] designed a novel method of buffer generation based on vector boundary tracing. This method could avoid complex vector calculations while retaining the high precision of existing vector-based algorithms. The emergence of parallel computing further improved the speed of buffer computation. Sorokine [12] introduced a parallel high-performance visualization technique in GRASS GIS. Darling et al. [13] reported the performance of a prototype parallel data partitioning algorithm for the input of vector-topological data into parallel processes. Huang et al. [14] proposed and designed a specific parallel IDW interpolation algorithm, incorporating single processes, multiple data and master/slave frameworks. Yao et al. [15] presented a parallel algorithm for buffer analysis, featuring a task-assignment based on the map layer and geographic spatial area. Their algorithm consists of four parts: task division, communication analysis, task union, and task assignment. They adopted a client/server architecture in which the servers generate buffer zones and the client collects and combines the results from these servers. Their experimental results showed that this parallel algorithm achieved speed increases of 1.8x, 2.7x, and 3x with 2, 3, and 5 nodes, respectively. Pang et al. [16] analyzed the sequential buffer generation algorithm and advanced a parallel mechanism by establishing a master/slave cluster. They distributed N entities to M computing nodes, each of which could process N/M entities. Finally, the master simultaneously combined the results. A case study showed that the authors achieved a greater than 1.5x performance increase with 10 nodes. Obviously, there is still great potential for improving efficiency in existing solutions. In fact, sequential buffer generation has evolved to a certain degree of maturity. Tools such as GRASS GIS, Quantun GIS, etc., have matured technologies in sequential buffer generation. Additionally, for processing large-scale geographic data, GIS is usually flooded by the number of features in databases, instead of the number of entities in one feature. Considering the granularity of the input, parallel solutions without the need for re-implementing the existing sequential algorithms have been considered. However, there is no need to reinvent the wheel. In the following sections, we will mainly discuss the optimization of vector based buffer analysis algorithms because gDOS-GIS is based on vector-based geographic data structures. gDOS-GIS is a cluster-based GIS service developed by the Institute of Geographic Sciences and Natural Resources Research (IGSNRR) of the Chinese Academy of Sciences (CAS). We believe some parallel principles in our vector-specific algorithm can also be applied to raster-based algorithms.

Design of the Parallel Buffer Generation Algorithm

Buffer generation and union operations

To generate the sequential buffer algorithm in gDOS-GIS, the pseudo code is shown in table 1. Essentially, there are two operations required for parallel buffer analysis. The first operation is buffer generation stage, where the raw geographic data are computed to generate separate buffer zones. The second operation is buffer union

1. fetch data from the database
2. generate buffer zones as features // buffer generation
3. for each feature // buffer union
4. merge current feature with previous features
5. end for each
6. store the result feature into the database

Table 1: The pseudo code of the buffer generation algorithm.

ID	Number of Features	Number of Points	T_{buffer} (Seconds)	T_{union} (Seconds)
E1	11,840	196,659	7.96	11.79
E2	30,128	393,351	15.25	32.82
E3	52,611	590,004	21.94	58.12
E4	72,078	786,656	29.16	85.94
E5	88,985	983,305	36.40	109.04
E6	103,797	1,181,160	43.98	124.64
E7	112,963	1,377,917	55.64	156.16
E8	126,439	1,574,599	63.72	179.42
E9	144,375	1,771,258	72.27	207.35
E10	169,131	1,968,152	77.92	240.16
E11	187,788	2,164,802	84.25	273.93

Table 2: The execution time for buffer generation and union in eleven experiments.

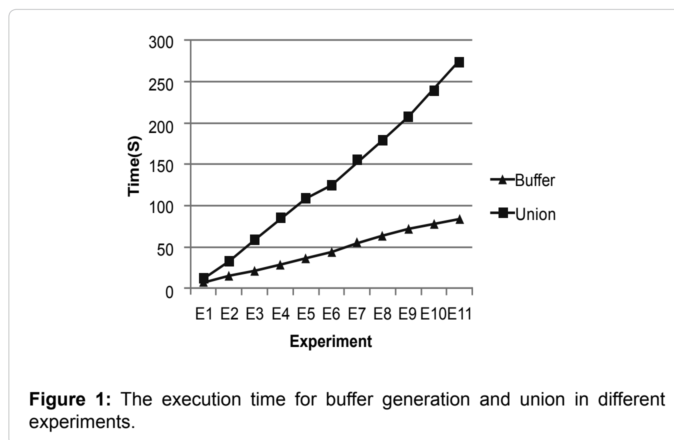


Figure 1: The execution time for buffer generation and union in different experiments.

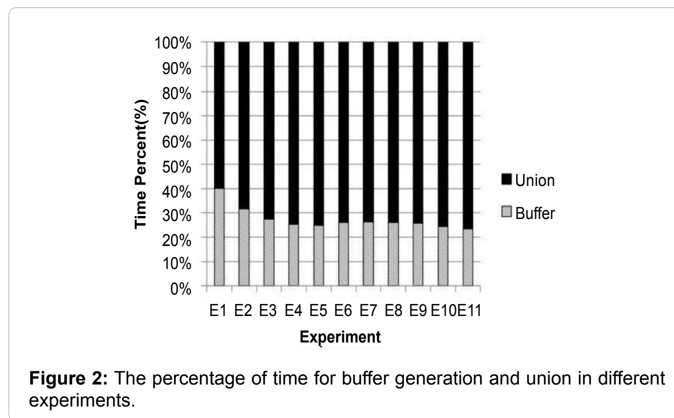


Figure 2: The percentage of time for buffer generation and union in different experiments.

stage, where the separate buffer zones are combined, with refinements, to produce a final result. To accelerate the buffer generation algorithm, we mainly aimed to optimize these two operations. We assume the total consumption time for the serial program to be $T_s(n)$, where n is the number of features. The individual parts of buffer generation and union are $T_{buffer}(n)$ and $T_{union}(n)$, respectively, yielding the following equation:

$$T_s(n) = T_{buffer}(n) + T_{union}(n) \quad (1)$$

To evaluate the relations between each operation in buffer analysis, we conducted eleven experiments with different numbers of rivers (i.e., features or layers) and points from a random set of geographic data consisting of 208,637 features provided by IGSNRR. The rivers are stored in the database as features, and a river feature may contain many points. Table 2 shows how execution time is related to the number of features and points. Direct manipulation of the points is not permitted in the database. To make the results more comprehensible, figure 1 and 2 were created. Figure 1 shows the relationship of the execution time

to the number of points in the different experiments. Figure 2 displays the distribution of time, as a percentage, for the union and buffer generation parts in the different experiments. From figure 1 and 2, we can see that in most of the experiments, the union operation takes up 75% of the total execution time, while buffer generation only takes up 25% of the total execution time. A linear relationship exists between the execution time and the number of points. In the following sections, we will describe the optimization methods for buffer operation and union operation.

Data decomposition for individual buffer generation

In a serial program, if we divide a dataset into small data chunks, in which the number of chunks is p, perform a buffer operation on each chunk, and finally perform a union operation on their results. The equation 1 becomes:

$$T_s(n,p) = p.\ T_{buffer}(n/p, p) + T_{union}(n, p) \quad (2)$$

These data subsets do not need to communicate with one another. Therefore, the buffer operation could be ideally computed using a parallel data method. Assuming that there are p processes and the execution time for the parallel program is T_p (n), it is easy to achieve the following:

$$T_p(n, p) = T_{buffer}(n/p, p) + T_{union}(n, p) \quad (3)$$

Next, we must define how to partition the data and assign tasks in parallel. Because the GIS data are always stored in databases as features and are processed by features, it is natural to allocate different layers to different computing units. The computing units do not need to communicate when processing different features. However, as the entities included in each feature are uncertain, we cannot confirm that every task allocated to every computing unit can finish synchronously. Thus, it is hard to keep a good load balance using this feature-based parallel method; it is too coarse-grained. The area-based parallel method [17] is another choice. This method divides the area covered by all of the features in the database among the computing units. Each computing unit is responsible for all of the records that exist in that geographical region. However, for most existing GIS systems, proper map size evaluation and map division are too complicated. Thus, this area-based method is too difficult to implement, as the basic data structure usually does not support the method. Moreover, a feature may appear in several sub-maps repeatedly, which will lead to redundant computing. Based on the above analysis, we designed a points-based parallel method to achieve fine-grained parallelization. As we have discussed previously, the execution time of individual buffer operations is defined by the number of points. To achieve the maximum performance for buffer operation, the best way is to allocate all computing tasks evenly. As there were no direct data that showed the number of points included in all of the features, we had to iterate all of the data in the database to map the number of points in each feature to compute the total number of points. Furthermore, we needed to divide all of the features into smaller groups to align the number of points in each group to be closer to the average span. This process adds extra overhead to the points-based parallel method, whose execution time is acceptable as discussed in the following section 4. Its pseudo codes based on MPI (Message Passing Interface) are shown in table 3. The most important feature of this algorithm is the ability to compute a specific span to ensure that each participating computing unit is responsible for approximately the same number of points.

Optimization of union operation

In this section, we will discuss the optimization method for the union operation. The classical parallel method utilizes a split-map-reduce scheme as shown in figure 3. The master process splits and assigns tasks to slave processes. Each slave process executes buffer operations and union operations individually. Finally, the master process gathers results from slaves and combines them into one final feature. This classical method has been deployed in current gDOS-GIS systems. An obvious bottleneck of this method is the reducing phase. In the reducing

phase, the master process has to collect each union result from the slave process and execute the final union operation. Because the data waiting for combination are large and are not parallel, the final union step is very slow. To address this problem, we designed a binary union tree for parallel buffer generation and the union algorithm. This method is shown in figure 4. There are no differences among the master processes and the slave processes. Every process follows the same steps: retrieval part of a portion of the geographic data and execution of the

```
1. initialize MPI
2. if (master process){
3. map the number of points in each feature to an array
4. get the number of total points
5. calculate the average span of the number of points
6. iterate the array and compute the specific span for each node
7. broadcast the specific span information
8. } else {
10. receive the specific span information
11. generate the buffer zone for specific span
12. }
13. finalize MPI
```

Table 3: The pseudo code of parallel buffer operation based on MPI.

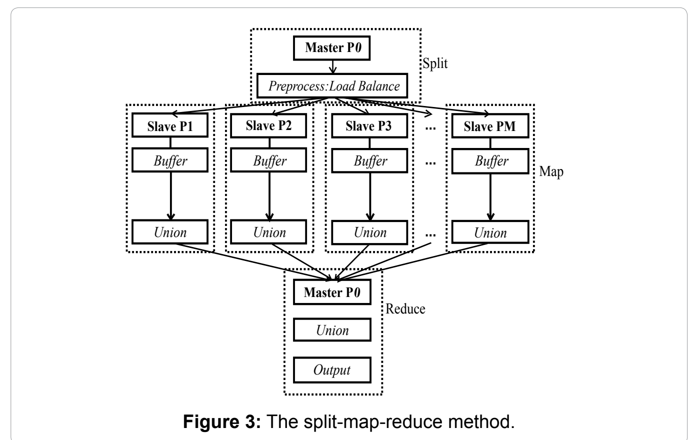


Figure 3: The split-map-reduce method.

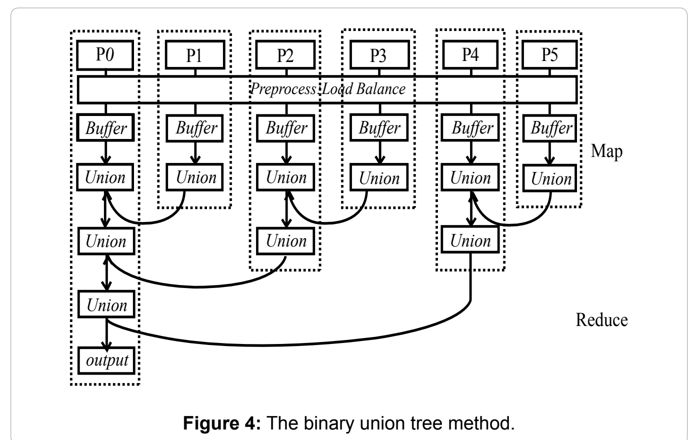


Figure 4: The binary union tree method.

Citation: Huang X, Pan T, Ruan H, Fu H, Yang G (2013) Parallel Buffer Generation Algorithm for GIS. J Geol Geosci 2: 115. doi: 10.4172/2329-6755.1000115

buffer operation and the union operation. Subsequently, one combined buffer feature is pulled by another process and then merged with the counterpart buffer feature in that process. Every pair of buffer zones is combined until only one bu_er zone exists. The union process is entirely parallel. This binary union tree method can decrease the time complexity of union operations from O(N) to O(logN).

Experiments and Analysis

Based on the analyses presented above, our points-based load balance and binary union tree strategies are predicted to be much better than the existing work. The following experiments and results are presented to illustrate the advantage. The results display the uniqueness of our algorithm compared to the different candidates. All of the programs run on a two-node cluster. It is noteworthy that the nodes do not directly send data with MPI; instead, the nodes save the data in the local database, send IP addresses, and fetch data from remote databases. In fact, data Structures (or memory models) in geodata do not fully support parallel computing, and MPI is unable to serialize and send data represented as points in a geographic object in C++. For comparison, we implemented three algorithms in C++ to evaluate the binary union tree and points-based load balance. The first algorithm, named Algorithm1, highlights a features-based load balance and a one-by-one union operation conducted by the master. Every slave inputs the same number of features, conducts buffer generation (including a buffer function and a union function), and saves its result into the local database. Finally, the master process retrieves data from the databases in all slaves and combines them one at a time. The second algorithm, named Algorithm2, differs from the previous algorithm by replacing the one-by-one union with the binary union tree. Every slave inputs the same number of features, conducts buffer generation (including a buffer function and a union function), and saves its result into the local database. The features in every two processes are combined until there is only one output, as shown in figure 4. The third algorithm, named Algorithm3, is our integrated design. It replaces the features-based load balance mentioned above with the points-based load balance. The remaining part is the same as Algorithm2. To prove that parallel buffer generation, highlighting a points-based load balance and binary union tree, has a better performance and a higher efficiency, we conducted experiments in which three algorithms were run on 1, 2, 4, 8, 16, or 32 processes. The data set contains over 200,000 features. The experimental results are shown in figure 5, 6 and 7. These figures show the execution time, speedup and efficiency of the three algorithms with the various numbers of processes. The execution time is the actual time for executing the GIS program. Speedup is defined as the ratio of Ts, the execution time for the single-process algorithm, to Tp, the execution time for the parallel algorithm: $S=T_s / T_p$

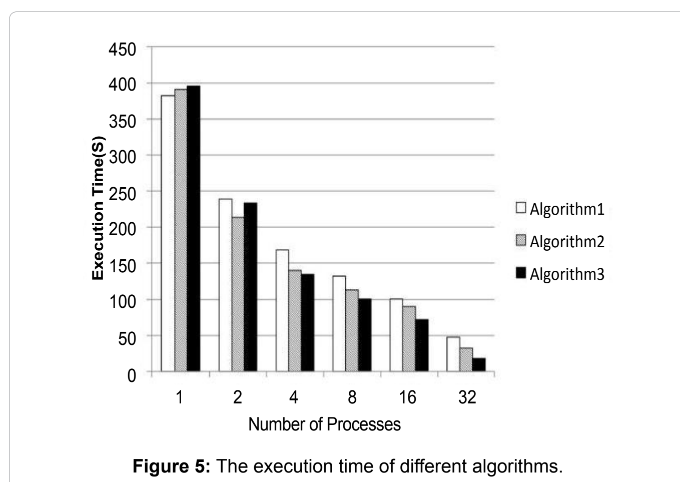


Figure 5: The execution time of different algorithms.

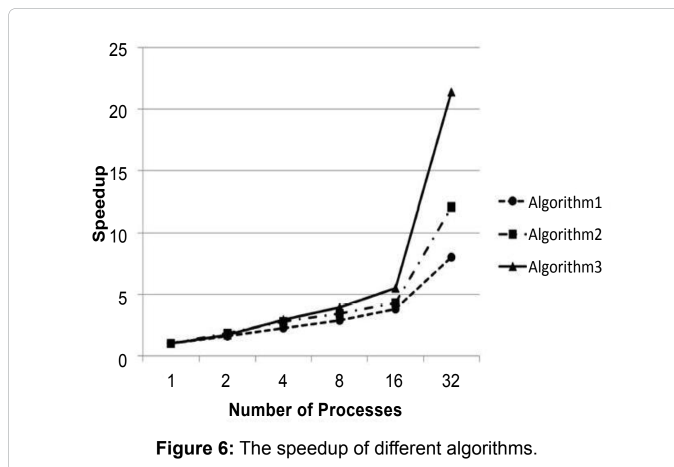


Figure 6: The speedup of different algorithms.

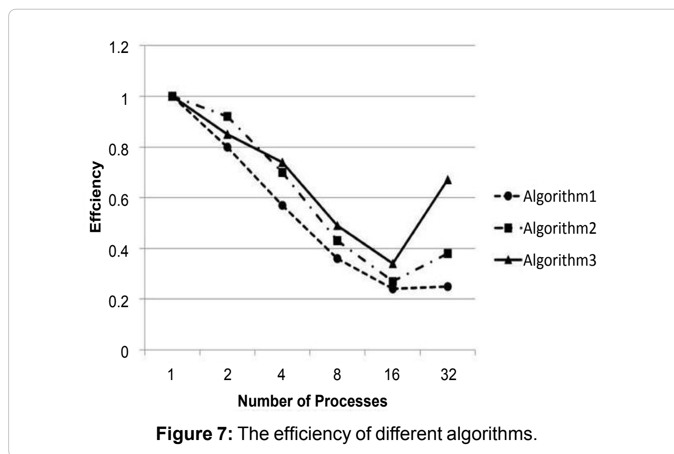


Figure 7: The efficiency of different algorithms.

Efficiency is defined as the ratio of speedup to P, the number of processes executing the algorithm: $E = S/P$ The closer the efficiency is to 1, the greater the efficiency of the parallel algorithm. For Algorithm1, the single-process algorithm spends 381.9 s in buffer generation. While utilizing 32 processes, the parallel algorithm reduces the execution time to 47.52 s. The maximum speedup is 8.04. However, as the number of processes increases, the efficiency decreases to 0.25 with 32 processes. The low efficiency is largely caused by union operation in the master process. As we have mentioned before, when the master process combines the generated buffer zones, other processes are idle. For Algorithm2, the speedup reaches 12.07 with 32 processes. Algorithm3 surpasses the other two algorithms in both speedup and efficiency. Algorithm3 decreases the execution time from 395.51 s with the original single process to 18.47 s with 32 processes, arriving at a speedup of 21.41. Similar to Algorithm1 and Algorithm2, the overall execution time of Algorithm3 also decreases substantially. This result confirms that the overall computation time of Algorithm3 is at a minimum when the buffers are combined by every two processes. Figure 5 also reveals that when the number of processes is low, Algorithm 3 is not the optimal method, as its load balance goes through an additional iteration of each source feature. In general, Algorithm 3 was the most outstanding choice when the number of processes is greater than 4. Figure 7 reveals the trends of efficiency among the three algorithms. Efficiency is an important criterion for developing a parallel

algorithm. As speedup is proportional to efficiency, for a given number of processes, a higher efficiency correlates with greater speedup. When developing a parallel algorithm, the algorithm is made to be as efficient as possible when executed by multiple processes. Unexpectedly, when the number of processes increases from 16 to 32, the efficiency increases from 0.24 to 0.25, 0.27 to 0.38, and, amazingly, 0.34 to 0.67. This result is mainly due to our special experimental environment. The cluster has two nodes, each consisting of 16 cores. Within each node, 16 processes share the same database, as well as other resources. The competition for computing and storage resources results in additional overhead. When the number of nodes (and databases) grows, the situation is well relieved. In practice, this buffer algorithm and the gDOS-GIS aim for a large cluster of commodity machines. The algorithm would most likely fit into the commodity cluster very well, with each node having its own database-serving processes, because the competition for the database would be largely eased. It is reasonable to predict that the efficiency would be better if Algorithm3 runs on the commodity cluster. In summary, Algorithm3 had the most satisfying performance and could successfully meet our needs for speedup and efficiency.

Conclusion

In this study, we designed a novel parallel solution consisting of a points-based load-balanced method and a binary union tree method to accelerate buffer generation. By comparing several parallel candidates, the experimental results show that the new parallel algorithm achieves much better performance and scalability. The speedup of the algorithm reached 21-fold with 32 processes.

References

1. Healey RG, Minetar MJ, Dowers S (1997) Parallel Processing Algorithms for GIS. Taylor & Francis Inc, Bristol, PA, USA.
2. Kerr N (2009) Alternative approaches to parallel GIS processing. Master's thesis, Arizona State University.
3. Wu H (1997) Problem of buffer zone construction in GIS. Journal of Wuhan Technical University of Surveying and Mapping 22: 358-365.
4. Dowers S, Sloan T, Gittings B, Healey R, Waugh T (1992) Exploring GIS performance issues. In: 5th International Symposium on Spatial Data Handling, Charleston.
5. Longley PA, Goodchild MF, Maguire DJ, Rhind DW (2005) Geographic information systems and science. Wiley.
6. Wang J, Chen Y, Cui C (2008) Review of buffer generation algorithm studies. In: Proceedings of the 2008 Second International Symposium on Intelligent Information Technology Application IITA, IEEE Computer Society, Washington, DC, USA.
7. Elber G, Lee IK, Kim MS (1997) Comparing offset curve approximation methods. IEEE Comput Graph Appl 17: 62-71.
8. Er E, Kilinc I, Gezici G, Baykal B (2009) A buffer zone computation algorithm for corridor rendering in GIS. 24th International Symposium on Computer and Information Sciences, Guzelyurt.
9. Wolff A, Knipping L, Kreveld MV, Strijk T, Agarwal P (2000) A Simple and Efficient Algorithm for High-Quality Line Labeling. GeoComputation 11: 147-159.
10. Ren Y, Yang C, Yu Z, Wang P (2004) A way to speed up buffer generalization by douglas-peucker algorithm. IEEE International of Geoscience and Remote Sensing Symposium 5: 2916-2919.
11. Jiechen W, Qing Y, Yanming C (2009) A novel method of buffer generation based on vector boundary tracing. Proceedings of the 2009 International 14 Manuscript Forum on Information Technology and Applications, Washington, DC, USA.
12. Sorokine A (2007) Implementation of a parallel high-performance visualization technique in GRASS GIS. Comput Geosci 33: 685-695.
13. Darling G, Sloan TM, Mulholland C (2000) The Input, Preparation, and Distribution of Data for Parallel GIS Operations. Proceedings from the 6th International Euro-Par Conference on Parallel Processing, London, UK.
14. Huang F, Liu D, Tan X, Wang J, Chen Y, He B (2011) Explorations of the implementation of a parallel IDW interpolation algorithm in a Linux cluster-based parallel GIS. Comput Geosci 37: 426-434.
15. Yao Y, Gao J, Meng L, Deng S (2007) Parallel computing of buffer analysis based on grid computing. Geospatial Information 5: 1672-4623.
16. Pang L, Li G, Yan Y, Ma Y (2009) Research on parallel buffer analysis with grided based HPC technology. IGARSS 4: 200-203.
17. Patel J, Yu J, Kabra N, Tufte K, Nag B, et al. (1997) Building a scaleable geo-spatial dbms: technology, implementation, and evaluation. SIGMOD 26: 336-347.