

Short Paper

Load Balancing for the Parallel Map Overlay-Operation In the Geographic Information System*

YU-TAI CHING

*Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan 300, R.O.C.
E-mail: ytching@cis.nctu.edu.tw*

The map overlay-operation is one of the most important and most time consuming operations in the Geographic Information System. In general, a map consists of a huge number of line segments. The major cost for overlaying two maps is that of computing the intersection points between the line segments. Parallel processing is one of the ways to reduce the computing time. If one can partition the line segments into independent subsets, then these subsets can be processed in different processors simultaneously; thus, the computing time can be reduced. In this paper, we consider the problem of partitioning line segments into independent sets such that the load is balanced among the processors. An easy yet effective strategy is proposed to balance the load for a multi-processor computer which does not have many processors. The proposed algorithm can achieve good load balance when the average length of the line segments is short compared to the width of a map.

Keywords: geographic information system, parallel algorithm, load balancing, computational geometry, line segments intersection

1. INTRODUCTION

Overlaying two maps is one of the most important operations in the Geographic Information System (GIS). A map is generally a planar graph in which each face contains information of a region. If there are two maps for an area, overlaying these two maps produces a new map which contains information from both maps.

To overlay two maps, we first compute the intersection points between the line segments of the two maps and split the line segments if they have intersections. We then organize the fragment edges to reconstruct the overlaid map. The most time consuming step in this operation is that of computing the intersection points. The intersection points can be evaluated using a brute force approach which checks the intersections between all pairs of edges formed by the line segments from both maps. This approach, if both maps

Received November 7, 1997; revised May 15 & August 17, 1998; accepted September 1, 1998.
Communicated by Jhing-Fa Wang.

*This work was supported in part by the National Science Council, Taiwan, Republic of China under Grant No. NSC81-0408-E009-534 and NSC82-0408-E009-415.

have n line segments, requires examination of $O(n^2)$ pairs of line segments. Since the bottleneck of the overlay-operation is computing the intersection points, much work has been done to reduce this cost [1, 2].

Recently, one of the approaches proposed to reduce the computing time takes advantage of parallel computers [3, 4]. If the line segments are divided into subsets and distributed into different processors, then the intersection points can be evaluated in parallel. Wang and Waugh studied the parallel approach in the area of the Geographic Information System [5, 6]. In [6], a map is divided into chains. Each processor evaluates the intersections between two chains from different maps. Waugh [6] studied parallel algorithm for an MIMD architecture parallel machine. Neither author considered partitioning the line segments into independent subsets.

If the set of line segments can be divided into independent subsets, where a line segment in a subset cannot intersect line segments in the other subsets, then the computing time will then be dominated by the processor which takes the longest time. The longest time taken by one of the processors can be minimized by balancing the load among the processors.

In this paper, we discuss the load balancing problem for parallel computing the intersections of line segments. We assume that the cost is determined by the number of pairs of line segments been evaluated for intersection. Our goal is to balance the number of pairs of line segments being assigned to each processor. We also assume that the system is a share-memory multiprocessor machine which does not have many processors (not a massively parallel machine). This is a reasonable consideration since most of the GIS packages presently run on workstations which have two or four processors.

The strategy we propose is to partition the line segments using a vertical line. In Section 2, we shall define a cost function with respect to a vertical line dividing the line segments. By minimizing the cost function, we can find the optimal bipartition. In Section 3, efficient algorithms for finding the optimal bipartition are presented. We then show that the efficiency of the algorithm can be further improved. A simple simulation was done to evaluate the "speed up" of the proposed algorithm. Simulation results and discussion are given in Section 4.

2. THE COST FUNCTION

Let M_1 and M_2 be two sets of n line segments. There are $O(n^2)$ pairs of line segments in M_1 and M_2 that may have intersections. Let C be a vertical line that divides M_1 and M_2 into M_{1_1}, M_{1_2} and M_{2_1}, M_{2_2} respectively. The two subsets M_{1_1}, M_{2_1} and M_{1_2}, M_{2_2} are independent, so these two subsets can be processed in two processors simultaneously (see Fig. 1). Suppose that C divides M_1 and M_2 into subsets of size $n/2$. The number of pairs of line segments been processed by a processor is reduced to a quarter of the original number. It may not be possible to have a perfect balance such that the line segments are divided into two subsets of size $n/2$. In this case, the total time for computing the intersection points is dominated by the one which handles more pairs of line segments than the other. The goal is to balance the number of pairs of line segments by selecting a vertical cut. We now define a cost function with respect to a vertical cut C . The cost function represents the maximum number of pairs of line segments held in a processor for a given cut C . Let C be a vertical line dividing M_1 and M_2 . Let $k_1, k_2, p_1, p_2, q_1,$ and q_2 be as the follows:

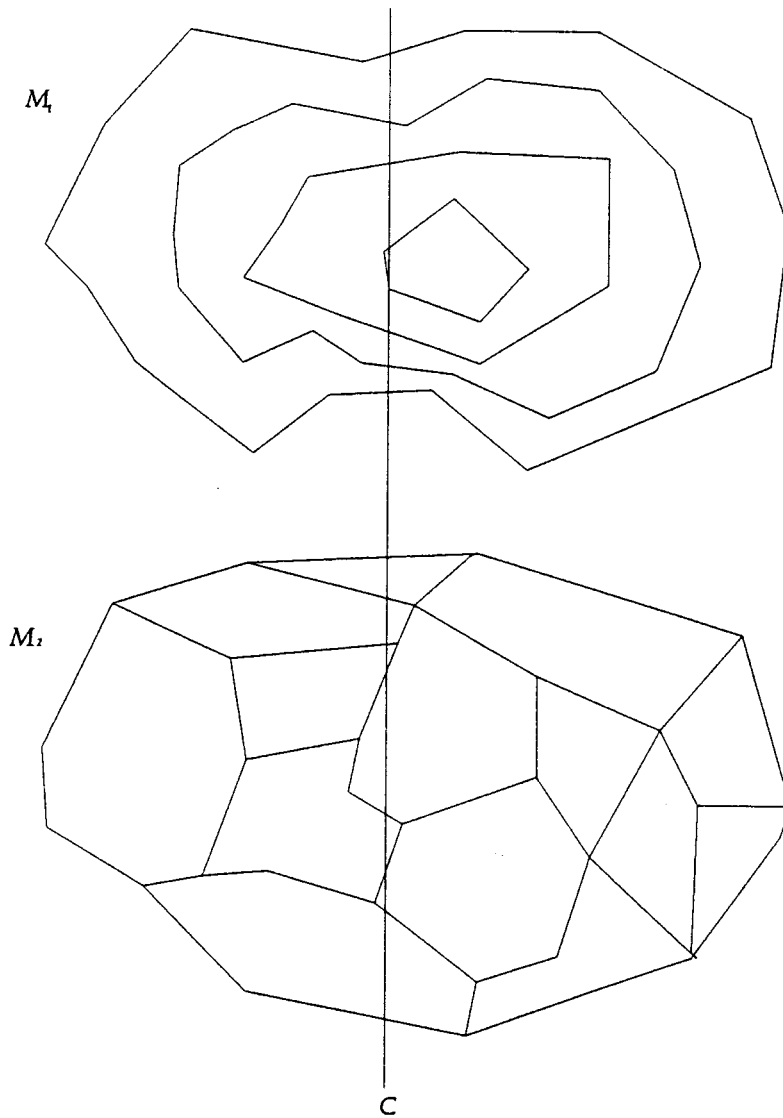


Fig. 1. Overlaying two maps M_1 and M_2 . If there is a vertical cut C that divides M_1 and M_2 into two, then there are line segments to the left, to the right of C and cut by C . The intersection between line segments to the left and to the right of C are evaluated in different processors simultaneously. Note that the line segments intersected by C are split in two and distributed into two processors. The number of pairs of line segments to the left of C in M_1 and M_2 is $(p_1 + k_1)$ and $(p_2 + k_2)$. p_1 and p_2 are the number of line segments in M_1 and M_2 totally to the left of C . k_1 and k_2 are the number of line segments in M_1 and M_2 cut by C . The number of pairs of line segments which need to be evaluated for intersection is $(p_1 + k_1)(p_2 + k_2)$.

- k_1 edges in M_1 are cut by C .
- k_2 edges in M_2 are cut by C .
- p_1 edges in M_1 are totally to the left of C .
- p_2 edges in M_2 are totally to the left of C .
- q_1 edges in M_1 are totally to the right of C .
- q_2 edges in M_2 are totally to the right of C .

The cost function with respect to the cut C is defined as

$$\max((p_1 + k_1)(p_2 + k_2), (q_1 + k_1)(q_2 + k_2)). \quad (1)$$

In Equation (1), $k_1 + k_2$ is the number of line segments which are intersected by C and, thus, are split into two subsets. After splitting the line segments, the number of line segments in M_{1_1} , M_{2_1} , M_{1_2} , and M_{2_2} is, respectively, $p_1 + k_1$, $p_2 + k_2$, $q_1 + k_1$, and $q_2 + k_2$. Since the intersection points to the left and to the right of C are calculated in different processors simultaneously, the total cost is dominated by the processor which handles more pairs of line segments, i.e., $\max((p_1 + k_1)(p_2 + k_2), (q_1 + k_1)(q_2 + k_2))$. The cut determining the minimum cost in Equation (1) is the **optimal cut**.

3. OPTIMAL BIPARTITION

We will now show that the optimal cut for a two-processor system can be obtained efficiently.

For ease of presentation, we assume that both M_1 and M_2 have n line segments. The j th edge in M_i is denoted as e_i^j , $i = 1, 2, j = 1, \dots, n$. Let H be a horizontal line, and let s_i^j be the interval which is the vertical projection of e_i^j on H . Note that the set of e_i^j , $i = 1, 2, j = 1, \dots, n$, corresponds to a set of intervals on H . A vertical cut C defines a point c on H . The number of intervals containing c is exactly $K = k_1 + k_2$. p_1, p_2 and q_1, q_2 are the number of intervals totally to the left and to the right of c , respectively.

Consider sweeping a vertical cut from left to right over the maps. The sweeping action is equivalent to sliding c along H from the first interval to the last interval. Observe that whenever c enters an interval, K is increased by one. On the other hand, if c leaves an interval, K is decreased by one. This fact states that when sweeping a cut from left to right, the number of edges in M_1 and M_2 intersected by the sequence of cuts is a series $A = \langle a_1, a_2, \dots, a_{4n} \rangle$. Furthermore $a_{i+1} = a_i + 1$ if the sweeping cut crosses the left end of an edge, and $a_{i+1} = a_i - 1$ if the sweeping cut crosses the right end of an edge. Note that each a_i is an *interval* on H ; i.e., no cut in the interval crosses any end point of any edge. In the following, we shall use $C(i)$ to refer to a cut in the interval a_i . The number of edges in M_j which are to the left of $C(i)$, to the right of $C(i)$, and cut by $C(i)$, is denoted by p_j^i, q_j^i , and $k_j^i, j = 1, 2, i = 1, \dots, 4n$.

Definition: a_i is a valley on A if and only if $a_{i-1} = a_i + 1$ and $a_{i+1} = a_i + 1$.

It is obvious that the number of valleys is no more than $2n$ since a valley occurs only when the sweeping line crosses a right end point of a line segment.

Theorem: If a_i is a valley, then the cost function, $\max((p_1^i + k_1^i)(p_2^i + k_2^i), (q_1^i + k_1^i)(q_2^i + k_2^i))$, with respect to $C(i)$ is a local minimal.

Proof: We assume that the Theorem is not true. Then there is a cut $C(j)$ such that both costs with respect to $C(j-1)$ and $C(j+1)$ are greater than the cost with respect to $C(j)$, and that a_j is not a valley. Since a_j is not a valley, we have two cases: either $a_{j-1} < a_j$ or $a_{j+1} < a_j$. We argue that the costs with respect to $C(j-1)$ or $C(j+1)$ are less than or equal to the cost with respect to $C(j)$ and, thus, prove the theorem.

We first consider the case in which $a_{j-1} < a_j$. This is a case in which a sweeping vertical line crosses a left end point of an edge $e \in M_1$. In this case, $p_1^{j-1} + p_2^{j-1}$ is the same as $p_1^j + p_2^j$, and $q_1^{j-1} + q_2^{j-1}$ is one more than $q_1^j + q_2^j$. Assume that $q_1^{j-1} = q_1^j + 1$ and $q_2^{j-1} = q_2^j$. Then, $k_1^{j-1} = k_1^j - 1$ and $k_2^{j-1} = k_2^j$. We have

$$\begin{aligned} (p_1^{j-1} + k_1^{j-1}) \cdot (p_2^{j-1} + k_2^{j-1}) &= (p_1^j + k_1^j - 1) \cdot (p_2^j + k_2^j) \\ &< (p_1^j + k_1^j) \cdot (p_2^j + k_2^j) \text{ and} \\ (q_1^{j-1} + k_1^{j-1}) \cdot (q_2^{j-1} + k_2^{j-1}) &= (q_1^j + 1 + k_1^j - 1) \cdot (q_2^j + k_2^j) \\ &= (q_1^j + k_1^j) \cdot (q_2^j + k_2^j). \end{aligned}$$

The cost with respect to $C(j-1)$ is less than or equal to the cost with respect to $C(j)$. On the other hand, $e \in M_2$, then $q_1^{j-1} = q_1^j$ and $q_2^{j-1} = q_2^j + 1$. In this case, $k_1^{j-1} = k_1^j$ and $k_2^{j-1} = k_2^j - 1$:

$$\begin{aligned} (p_1^{j-1} + k_1^{j-1}) \cdot (p_2^{j-1} + k_2^{j-1}) &= (p_1^j + k_1^j) \cdot (p_2^j + k_2^j - 1) \\ &< (p_1^j + k_1^j) \cdot (p_2^j + k_2^j) \text{ and} \\ (q_1^{j-1} + k_1^{j-1}) \cdot (q_2^{j-1} + k_2^{j-1}) &= (q_1^j + k_1^j) \cdot (q_2^j + 1 + k_2^j - 1) \\ &= (q_1^j + k_1^j) \cdot (q_2^j + k_2^j). \end{aligned}$$

The cost with respect to $C(j-1)$ is still less than or equal to the cost with respect to $C(j)$.

The second case is $a_j > a_{j+1}$. In this case, the sweeping cut crosses a right end point of an edge. We can use arguments similar to those above to prove that the cost with respect to $C(j+1)$ is less than or equal to the cost with respect to $C(j)$.

Both cases stated above show that the costs with respect to either $C(j-1)$ or $C(j+1)$ are no more than the cost with respect to $C(j)$. The theorem follows. \square

Based on the above Theorem, if the sorted list for the end points along the x -coordinate in M_1 and M_2 is available, since the sequence A can be obtained from the sorted list in linear time and there are at most $2n$ cuts reaching local minimal, then optimal cut for bipartition can be obtained in linear time by choosing the minimum cost among all these cuts.

The linear time algorithm stated above can be further improved. The main idea is to not check all the valleys since an algorithm similar to binary search is applicable. However, we need to preprocess the data set so that the sorted order of the end points along H and the sequence A are available.

Lemma: Let $F(i)$ and $G(i)$ be, respectively, the values $(p_1^i + k_1^i) \cdot (p_2^i + k_2^i)$ and $(q_1^i + k_1^i) \cdot (q_2^i + k_2^i)$ with respect to a cut $C(i)$. As i increases from 1 to $4n$, $F(i)$ increases monotonically, and $G(i)$ decreases monotonically.

Proof: We first show that $F(i)$ increases monotonically by induction on $i, i = 0, \dots, 4n$.

It is obvious that $F(1) \geq F(0)$. Assume that $F(\cdot)$ is increasing monotonically for all $k, 0 \leq k \leq i$. We will show that $F(i + 1) \geq F(i)$.

There are two cases: either $a_{i+1} = a_i + 1$ or $a_{i+1} = a_i - 1$.

Case 1: $a_{i+1} = a_i + 1$.

The cuts are $C(i+1)$ and $C(i)$. Then, we have $k_1^{i+1} + k_2^{i+1} = k_1^i + k_2^i + 1$ and $p_1^{i+1} = p_1^i, p_2^{i+1} = p_2^i$. There are two sub-cases as follows:

- $q_1^{i+1} = q_1^i, q_2^{i+1} = q_2^i - 1$ and
- $q_1^{i+1} = q_1^i - 1, q_2^{i+1} = q_2^i$.

If $q_1^{i+1} = q_1^i$ and $q_2^{i+1} = q_2^i - 1$, then $k_1^{i+1} = k_1^i$ and $k_2^{i+1} = k_2^i + 1$. We have

$$(p_1^{i+1} + k_1^{i+1})(p_2^{i+1} + k_2^{i+1}) = (p_1^i + k_1^i)(p_2^i + k_2^i + 1)$$

$$> (p_1^i + k_1^i)(p_2^i + k_2^i).$$

If $q_1^{i+1} = q_1^i - 1$ and $q_2^{i+1} = q_2^i$, then $k_1^{i+1} = k_1^i + 1$ and $k_2^{i+1} = k_2^i$. We have

$$(p_1^{i+1} + k_1^{i+1})(p_2^{i+1} + k_2^{i+1}) = (p_1^i + k_1^i + 1)(p_2^i + k_2^i)$$

$$> (p_1^i + k_1^i)(p_2^i + k_2^i).$$

Case 2: $a_{i+1} = a_i - 1$.

In this case, we have $k_1^{i+1} + k_2^{i+1} = k_1^i + k_2^i - 1$ and $q_1^{i+1} = q_1^i, q_2^{i+1} = q_2^i$.

- $p_1^{i+1} = p_1^i, p_2^{i+1} = p_2^i - 1$ and
- $p_1^{i+1} = p_1^i - 1, p_2^{i+1} = p_2^i$.

If $p_1^{i+1} = p_1^i$ and $p_2^{i+1} = p_2^i - 1$, then $k_1^{i+1} = k_1^i$ and $k_2^{i+1} = k_2^i + 1$. We have

$$(p_1^{i+1} + k_1^{i+1})(p_2^{i+1} + k_2^{i+1}) = (p_1^i + k_1^i)(p_2^i - 1 + k_2^i + 1)$$

$$= (p_1^i + k_1^i)(p_2^i + k_2^i).$$

If $p_1^{i+1} = p_1^i - 1$ and $p_2^{i+1} = p_2^i$, then $k_1^{i+1} = k_1^i + 1$ and $k_2^{i+1} = k_2^i$. We have

$$(p_1^{i+1} + k_1^{i+1})(p_2^{i+1} + k_2^{i+1}) = (p_1^i - 1 + k_1^i + 1)(p_2^i + k_2^i)$$

$$= (p_1^i + k_1^i)(p_2^i + k_2^i).$$

We have $F(i+1) \geq F(i)$ in both cases.

To show that $G(i)$ decreases monotonically as i increases is the same as showing $G(i)$ is increases monotonically as i decreases. We assume that the line sweeps from right to left. The arguments are similar to the above. □

Theorem: The optimal bipartition can be obtained in $O(\log n)$ if the sequence A is available.

Proof: First note that $F(\cdot)$ and $G(\cdot)$ are continuous, and that $F(0) = 0, F(4n) = n^2$ and $G(0) = n^2, G(4n) = 0$. There must exist $j, 0 \leq j \leq 4n$ such that $F(j) = G(j)$. Furthermore, for any i less than $j, G(i) > F(i)$. On the other hand, for any k greater than $j, G(k) < F(k)$. The algorithm then first checks a_j , the median on A , and determines whether $F(j) = G(j)$. If $F(i) < G(i)$, then the intersection of $F(\cdot)$ and $G(\cdot)$ must be to the right of j (or to the left of j).

otherwise). This operation drops half of the search space upon each iteration in constant time. Since there are at most $O(\log n)$ iterations, the optimal cut can be found in $O(\log n)$ time. \square

4. CONCLUSIONS

In this paper, we have presented a simple and efficient algorithm to balance the load in a parallel computer. Assume that the cost for a processor is the number of pairs of line segments being processed. We have suggested splitting a map using a vertical line. The optimal bipartition can be obtained efficiently. The proposed strategy requires that a map be stored in a Doubly Connected Edge List [1]. Based on the data structure, the proposed algorithm can be implemented in a multi-processor workstation.

A simple simulation was done to evaluate the “speed up” of the proposed algorithm. The assumption for this simulation was that the cost was determined by the number of intersection tests. Based on this assumption, the speed up was measured using the ratio of the numbers of pairs of line segments tested before and after partitioning. The simulation is briefly described in the following. We generated two sets of intervals representing the projection of the line segments in M_1 and M_2 on H . The left end point of each interval was uniformly distributed over the range 0 to 100. The right end point of an interval depended on the length of the interval, which was uniformly distributed over the range for a given length l . The average length of the intervals was thus, $l/2$.

According to the simulation results, the line segments having short average length could achieve better speedup. The reason is that the portion of the line segments intersected by a cut was relatively small compared to the total number of line segments. If the average length was less than 2.5% of the width of the map, the speedup was above 3.8, which is close to the ideal case (four times the original cost)(Table 1).

We also partitioned the intervals into 4 subsets by recursively applying the algorithm in section 3. That is, we first computed the optimal bipartition and then computed the optimal bipartition for the two subsets. This was not an optimal four-partition. But the simulation showed that it also achieved good speed up for a four-processor system. Since most of the maps in GIS do not have long average length except for street maps, we expect our approach will work well in real applications.

An interesting observation is that the speedup could be better than 4 or 16 in these two cases. In general, we expect to have a speedup of 4 for bipartitioning. But there are cases where the speed up is more than 4. Consider the following case. If both maps have 100 line segments and there is a cut which divides the two maps, respectively, into two subsets of size 1, 99 and 99, 1, then in this case, we have more than 100 times speed up. However, if the orientation of the cut is not fixed, then this will increase the cost of finding the optimal cut. Determining the optimal cut when that the orientation of the cut is not fixed is left for a future work.

Table 1. Simulation results of the relationship between the average length of intervals and the speedup in a two-processor system and a four-processor system.

Number of Line Segments	Average Length of Intervals	Speed Up		
		Two Processors	Four Processors	
$n_1 = 50$	50.00/100	1.83	2.61	
	25.00/100	2.55	4.79	
	10.00/100	3.32	9.40	
	5.00/100	3.56	12.76	
	$n_2 = 50$	2.50/100	3.85	13.74
		2.00/100	3.70	12.76
		1.00/100	3.75	13.89
$n_1 = 100$	50.50/100	1.9	2.82	
	25.00/100	2.53	5.19	
	10.00/100	3.26	9.78	
	5.00/100	3.63	10.99	
	$n_2 = 100$	2.50/100	3.70	13.23
		2.00/100	3.97	14.81
		1.00/100	3.92	14.88
$n_1 = 500$	50.00/100	1.87	2.75	
	25.00/100	2.59	5.41	
	10.00/100	3.25	9.30	
	5.00/100	3.64	11.66	
	$n_2 = 500$	2.50/100	3.79	13.46
		2.00/100	3.80	14.08
		1.00/100	3.86	15.02
$n_1 = 1000$	50.50/100	1.89	2.64	
	25.00/100	2.58	5.34	
	10.00/100	3.22	8.99	
	5.00/100	3.64	12.13	
	$n_2 = 1000$	2.50/100	3.81	13.63
		2.00/100	3.88	14.46
		1.00/100	3.89	15.03

REFERENCES

1. F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.
2. T. Pavlidis, *Algorithms for Graphics and Image Processing*, Computer Science Press, Rockville, Maryland, 1982.
3. C. Rüb "Line-segment intersection reporting in parallel," *Algorithmica*, Vol. 8, No. 2, 1992, pp. 119-144.

4. L. Boxer, R. Miller and A. Rau-Chaplin, "Some scalable parallel algorithms for geometric problems," Technical Report 96-12, Department of Computer Science, SUNY Buffalo, June 1996.
5. F. Wang, "A parallel intersection algorithm for vector polygon overlay," *IEEE Computer Graphics & Applications*, Vol. 13, No. 2, March 1993, pp. 74-81.
6. T. C. Waugh and S. Hopkins, "An algorithm for polygon overlay using cooperative parallel processing," *International Journal of Geographical Information Systems*, Vol. 6, No. 6, 1992, pp. 457-467.

Yu-Tai Ching (荆宇泰) was born on 20 February 1957 in Taipei, Taiwan, Republic of China. He received his BS degree in industrial engineering in 1980 from National Tsing Hua University, and his MS and Ph. D. degrees in 1983 and 1987, respectively, in computer science from Northwestern University. He is currently with the Department of Computer and Information science, National Chiao Tung University. His research interests are medical imaging, computer graphics, and design and analysis of computer algorithms.