

Parallel algorithms for least median of squares regression

Chong-wei Xu

Department of Computer Science, Southern Illinois University at Edwardsville, Edwardsville, IL, USA

Wei-Kei Shiue

Department of Mathematics and Statistics, Southern Illinois University at Edwardsville, Edwardsville, IL, USA

Received October 1991

Revised June 1992

Abstract: Recently much effort has been devoted to devising robust regression procedures to which the least median of squares (LMS) regression proposed by Rousseeuw (*Journal of the American Statistical Association* **79**, 1984) belongs. LMS's breakdown point achieves the highest possible value of 50%. It may not be possible to write down closed form expressions for the estimates of LMS and the computations involved are not trivial. Efforts have been made to look for better performance algorithms. Based on the algorithm of Rousseeuw, Steele and Steiger (*Discrete Applied Mathematics* **14**, 1986) provided a lemma that characterized the computation as a discrete optimization problem with worst time complexity of $O(n^3)$. Souvaine and Steele (*Journal of the American Statistical Association* **82**, 1987) developed a sweep-line algorithm with time complexity of $O(n^2 \log n)$. This paper reports the use of parallel computing techniques to speedup the computation of LMS. The Rousseeuw's algorithm and the T-sweep algorithm, an algorithm based on Tukey's idea, have been successfully parallelized, but parallel sweep-line algorithm and its refinement may not work at all in the sense that its performance is intolerantly low due to severe load imbalancing.

Keywords: Sweep-line algorithm; Parallel algorithm; Least median of squares regression; Load balancing.

1. Introduction

Regression analysis using least squares criterion has been very popular due to the computational simplicity and elegant statistical theory associated with it. However, it is not robust in the sense that a single outlier will produce a large

Correspondence to: Chong-wei Xu, Dept. of Computer Science, Southern Illinois University at Edwardsville, Edwardsville, IL 62026-1656, USA.

0167-9473/93/\$06.00 © 1993 – Elsevier Science Publishers B.V. All rights reserved

effect on estimates of parameters. Recently much effort has been devoted to devising robust regression procedures to which the least median of squares (LMS) regression proposed by Rousseeuw (1984) belongs. The least square procedure minimizes the sum of squared residuals to obtain estimates, while in the least median of squares regression, one finds estimates by minimizing the median of squared residuals. A measure of robustness is called the *breakdown point*, which is defined to be the smallest percentage of contaminated data (outliers) that can cause the estimate to take an arbitrarily large value. Least median of squares regression achieves the highest possible value of 50%. For least square regression, the breakdown point is only 0%.

For the case of simple linear regression, let (x_i, y_i) , $1 \leq i \leq n$, be the data points. The LMS estimates of intercept and slope, β_0^* and β_1^* , are the values of β_0 and β_1 that minimizes

$$f(\beta_0, \beta_1) = \text{median}(|y_i - \beta_0 - \beta_1 x_i|^2).$$

It may not be possible to write down closed form expressions for the estimates of LMS and the computations involved are not trivial. Consequently, efforts continue on looking for better performance algorithms. The first mention of an algorithm is in Rousseeuw (1984) and Leroy and Rousseeuw (1984). Since then, several algorithms have been developed to improve the time complexity of the LMS computation. Steele and Steiger (1986) provided a lemma that characterizes the computation as a discrete optimization problem with worst time complexity of $O(n^3)$, and verified that Rousseeuw's algorithm yields exact LMS fit for the simple linear regression case. In Souvaine and Steele (1987), sweep-line technique from computational geometry and two data structures form an algorithm that runs in time $O(n^2 \log n)$.

We attempt to use parallel computing techniques to speedup the LMS computation. Bailey (1989) pointed out that one way to overcome the principal obstacles in bringing advanced parallel computing power to real-world scientists and engineers is providing portable implementations of the latest high-performance parallel algorithms in a usable form. In this article, Rousseeuw's algorithm, sweep-line algorithm and a T-sweep algorithm based on Tukey's idea are parallelized and implemented on an Intel's iPSC/2 machine.

2. Rousseeuw's algorithm

The Rousseeuw's algorithm that has been utilized in the program PROGRESS (Leroy and Rousseeuw, 1984) for the simple linear regressions $y_i = \beta_0 + \beta_1 x_i$, can be described as follows.

(1) Choose a pair of data points from a set of size n to form a line, with equation

$$\hat{y}_i = b_0 + b_1 x_i.$$

```

for (i = 1; i <= n-1; i++) {
    for (j = i+1; j < n; j++) {
        make_a_line(data_ary[i], data_ary[j], &alpha, &beta);
        for (k = 1; k <= n; k++)
            med_ary[k] = calc_residual(data_ary[k], alpha, beta);
        sort(med_ary);
        median_val = med_ary[mid];
        optimal_med = find_opt(median_val);
    }
}

```

Fig. 1. Rousseeuw's algorithm (dimension $p = 2$) in the language C, where, variable `data_ary[]` is used for storing the data set; variable `alpha` corresponds to b_0 while `beta` corresponds to b_1 in the equation $\hat{y}_i = b_0 + b_1 x_i$.

(2) Calculate the median squared residual

$$\text{med}(y_i - b_0 - b_1 x_i)^2, \quad i = 1, 2, \dots, n.$$

(3) Repeat step 1 and 2 for all $\binom{n}{2}$ lines and the line corresponding to the minimal median squared residual is the LMS line.

There exists an important refinement which consists of adjusting the intercept at each replication (Rousseeuw and Leroy, 1987, p. 201). For each slope, the objective function is minimized by applying the exact one-dimensional LMS algorithm to the set of residuals. This intercept adjustment increases the computation time somewhat, because computing the objective for a given set of residuals takes $O(n \log n)$ time, whereas the plain median takes only $O(n)$ time. However, this increase is not significant (especially in multiple regression) because most of the computation time is spent in computing the trial estimates and the corresponding residuals.

The algorithm described above can be implemented by three nested for () loops illustrated by the C code in Figure 1. It is mentioned in Rousseeuw and Leroy (1987, p. 201) that the algorithm can benefit from parallel computing and the implementation was carried out on a 1CAP 1 system with significant reduction of computation time. It is therefore instructive to demonstrate in details the parallel architecture involved and provide speedup results. The parallel computations in this paper were done on an iPSC/2 hypercube computer with eight nodes.

The time required for the sequential computation can be expressed as

$$t_s = \binom{n}{2} (t_{\text{al}} + (n-2)t_{\text{cr}} + t_{\text{so}(n-2)}), \quad (1)$$

where: t_{al} – time spent for making one line; t_{cr} – time spent for one residual computation; and $t_{\text{so}(n-2)}$ – time spent for sorting a set of size $(n-2)$.

The parallel computing technique supports four different styles of parallel computation (King, Chou and Ni, 1990). The LMS computation is suitable for *concurrent data-parallel* style. Here multiple processors perform the same opera-

tion at the same time on different data sets. Referring to the C code shown in Figure 1, two ways may be used to parallelize Rousseeuw's LMS algorithm in a hypercube architecture which has one host processor and p node processors.

(1) Partition the set of n data onto the p node processors. Each node holds n/p points. The host forms $\binom{n}{2}$ lines one by one and broadcasts one line to all of nodes at a time. All the nodes calculate in parallel n/p residuals with respect to the current line. Sort the n/p residuals locally and, then, sort $(n-2)$ residuals globally cross all the nodes so that the median can be found. That is, the host performs the two outer for () loops of the algorithm, while every node performs $(1/p)$ th of the inner for () loop. The analytical expression of the time spent for this parallel computation is

$$t_{p1} = \Phi(n/p, p) + \binom{n}{2} \left(\Phi(1, p) + \frac{n}{p} t_{cr} + t_{ls(n/p)} \right) + t_{gs(n-2)} + \Phi(1, 1). \quad (2)$$

Where $\Phi(n/p, p)$, $\Phi(1, p)$ and $\Phi(1, 1)$ can be generalized as $\Phi(m_s, p)$ – time spent for transferring the data from host to nodes or from nodes to host, which is a function of both message size m_s and number of node processors p . The time spent for message transmission from host to one node or from one node to host is equal to message startup delay plus message transmission time: $t_{st} + m_s \times t_{ls}$.

$\Phi(n/p, p)$ – time spent for transferring n/p data to p nodes;

$\Phi(1, p)$ – time spent for transferring one line data to p nodes;

$\Phi(1, 1)$ – time spent for transferring the median from one node to the host;

$t_{ls(n/p)}$ – time spent for local sorting on the set of n/p residuals;

$t_{gs(n-2)}$ – time spent for a global sorting $(n-2)$ residuals cross p nodes.

According to the measured parameter (King, Chou and Ni, 1990), the host message startup delay t_{st} could be as large as $55 t_{cr}$. That is that the $\Phi(1, p)$ and $\Phi(1, 1)$ in the equation above will create a huge overhead: $\binom{n}{2} (\Phi(1, p) + \Phi(1, 1))$. Even though each node only needs n/p computations instead of $(n-2)$ that may save some time, but the global parallel sorting, say bitonic sorting, will add an additional time $t_{gs(n-2)}$ and make the implementation more complex.

(2) Partition the set of n data onto the p node processors. In parallel, each node forms lines from n/p data and calculates the $(n-2)$ residuals with respect to each of the lines. In addition to these 'local' lines, there are 'cross' lines between nodes, which are formed by the points stored in different nodes. In this case each node performs $(1/p)$ th of the two outer for () loops and the entire inner for () loop. Based on the data partition shown later, the time required is

$$\begin{aligned} t_{p2} &= \Phi(n, p) + \left(\binom{n/p}{2} + \frac{1}{2}(p-1) \left(\frac{n}{p} \right)^2 \right) (t_{al} + (n-2)t_{cr} + t_{so(n-2)}) \\ &\quad + \Delta(1, p) \\ &= \frac{\binom{n}{2}}{p} (t_{al} + (n-2)t_{cr} + t_{so(n-2)}) + \Phi(n, p) + \Delta(1, p), \end{aligned} \quad (3)$$

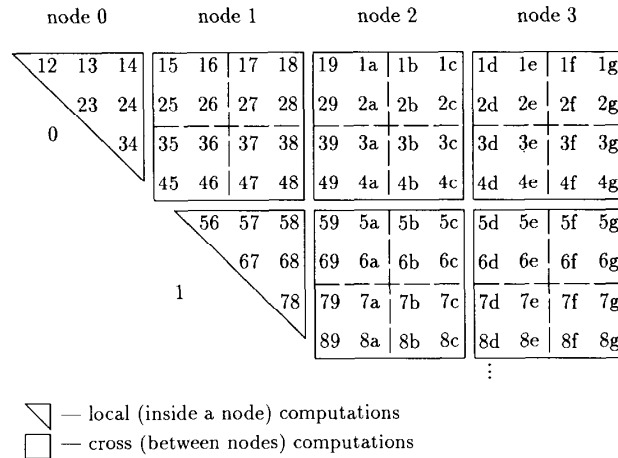


Fig. 2. Data partition for Rousseeuw's LMS algorithm.

where item (n_2^p) is the number of 'local' lines, item $(p-1)\frac{1}{2}(n/p)^2$ is the number of 'cross' lines each node processor involved; $\Delta(1, p)$ is the time that each node processor requires to transfer the optimal local median of size 1 to host processor and, then, the host sort p local optimal values to find the global optimal solution. The time length depends on the message transfer function $\Phi(1, p)$, relative computation speed between the p node processors and sorting time $t_{so(p)}$. We cannot have a deterministic value at the first place.

The first parallel algorithm requires that the host processor synchronizes the computations of all nodes on the basis of each line. Equation (2) indicates that the small granularity being accompanied with the huge overhead will dramatically degrade the parallel performance. The second parallel algorithm is selected because Eq. (3) indicates that it will reduce the number of synchronization points, i.e., increase the granularity of the parallelism. If we assign balanced loads to each node and each node performs the same number of computations asynchronously, then the speedup will be directly proportional to the value of p .

Suppose that a set of 16 input points are labelled as 1, 2, ..., 9, $a, b, \dots, f, g$ and the number of nodes used is 4, labelled as 0, 1, 2, 3. The data interval is $16/4$. The left data boundary of a node can be calculated as $(1 + \text{node_label} \times \text{data_interval})$. Figure 2 depicted the data partitions and their assignments for an LMS computation. The lines inside a triangle are assigned to each individual node. The computations inside each node can be called 'local computations', while the computations dealing with the lines in a square are the 'cross computations' between nodes. These 'cross' lines in one square are assigned to two nodes. For instance, the lines in the top first square of Figure 2 are evenly divided into four parts and assigned to node 0 and node 1 respectively.

Consequently, the algorithm needs to assign a partner for each node to share the computation load automatically. Taking eight nodes as an example. The

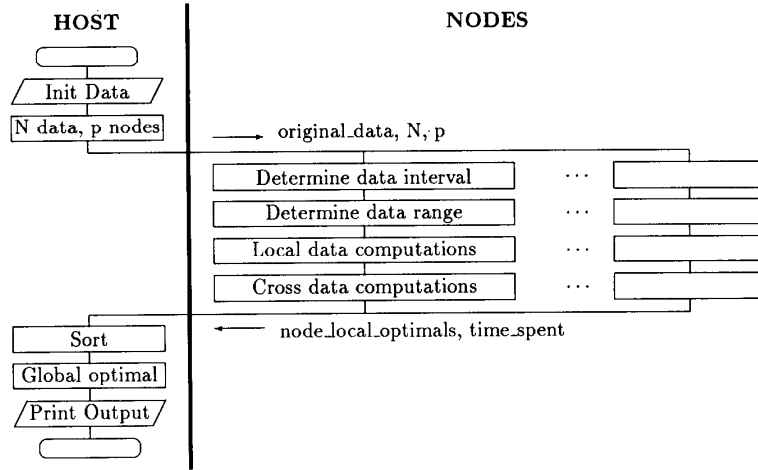


Fig. 3. Parallel version of Rousseeuw's LMS algorithm.

eight nodes correspond to dimension 3, i.e., each node can be labelled as three binary digits 000, 001, ..., 111. Each node is one, two, or three hops away from the other nodes. For instance, node 0 is labelled as 000. The node 0 has its one hop neighboring nodes 001, 010, and 100; two hops neighboring nodes 011, 101, 110; and three hops neighboring nodes 111.

A high-level description of the parallelized Rousseeuw's LMS algorithm is shown in Figure 3. The speedup of the parallel algorithm can be computed from the following equation.

$$s = \frac{t_s}{t_{p2}} = \frac{\frac{1}{2}n(n-1)t_{\text{one_line}} + \Phi(n, p) + \Phi(1, 1)}{(1/2p)n(n-1)t_{\text{one_line}} + \Phi(n, p) + \Delta(1, p)} \propto p, \quad (4)$$

where $t_{\text{one_line}} = t_{\text{al}} + (n-2)t_{\text{cr}} + t_{\text{so}(n-2)}$, the time spent for computing $(n-2)$ residuals with respect to one line. This equation predicts that the speedup of the parallel computation is directly proportional to the number of p nodes used if we omit the term $\Phi(m_s, p) + \Delta(1, p)$. The results of an experimental computation shown in Section 5 support this analysis.

3. Sweep-line algorithm

Let l_{β_0, β_1} be the line with equation $y = \beta_0 - \beta_1 x$ and define the residual $r_i = y_i - (\beta_0 + \beta_1 x_i)$. l_{β_0, β_1} is said to bisect the three distinct points (x_{ij}, y_{ij}) , $j = 1, 2, 3$, if the residuals r_{ij} are all the same size and do not have the same sign. Furthermore, if $x_{i1} < x_{i2} < x_{i3}$ and $r_{i1} = -r_{i2} = r_{i3}$, l_{β_0, β_1} is said to equioscillate the triple of points. Steele and Steiger (1986) have shown that an LMS line must be an equioscillating line with respect to some triple of points. This is

equivalent to finding a strip in the plane bounded by two parallel lines L_1 and L_2 that contains $K(\frac{1}{2}(n-4))$, for even n ; $\frac{1}{2}(n-5)$, for odd n points, and that the vertical distance between L_1 and L_2 is minimized. L_1 is formed by two of the triple points and L_2 passes through the third point. The LMS line is the line parallel to L_1 and L_2 and bisects the strip.

The estimate computation of LMS, therefore, becomes the minimal strip finding. Souvaine and Steel (1987) applied the point-line duality concept from computational geometry to transform a point to a line and a line to a point from the original plane to a mapped plane and thus obtained the dual problem of this procedure which can be described as follows: Given n lines L_i ($1 \leq i \leq n$) on the mapped plane, which correspond to the n data points on the original plane, in general positions with intersection points P_{ij} ($1 \leq i < j \leq n$), find the line L^* and intersection point P^* such that, among all line-point pairs (L, P) that have exactly K of L_i cutting the vertical segment S joining L and P , the pair (L^*, P^*) has the smallest vertical distance. They developed a sweep-line algorithm for solving the dual problem.

The sweep-line algorithm is to sweep a vertical line from the leftmost intersection point to the rightmost intersection point, stopping at each P_{ij} to perform some computations. To facilitate the technique two data structures are required. The first is called LIST which stores one line's slope and intercept in one record sorted in increasing order of slopes. To each line L_i in the structure, a set of four pointers are assigned to indicate the lines that are positioned 1 and K lines above or below L_i . The pointers are set to zero if the top or bottom condition is met. The second structure, referred to as HEAP, is used to store the intersection points of adjacent lines in LIST. Each intersection point stored in HEAP has two pointers pointing to the two adjacent lines stored in LIST. The intersection points are ordered according to x -coordinates.

The algorithm consists of the following steps:

- (1) Take the point stored in the root of HEAP as the leftmost intersection point (it has the smallest x -coordinate value among all of intersection points).
- (2) Initialize the sweep-line at the x -coordinate of the leftmost point.
- (3) Follow the pointers stored in the root of the HEAP to find the adjacent lines L_i and L_j which intersect and produce the leftmost point.
- (4) Compute the vertical distance from the leftmost point to the lines up_ K and down_ K , where $K = \frac{1}{2}(n-4)$, for even n ; $\frac{1}{2}(n-5)$, for odd n .
- (5) Compare the result with the previous optimum – the minimal vertical distance found in the previous step, and update the optimum.
- (6) Remove three points from HEAP, which were intersection points of L_i with up_1 line, L_i with L_j , and L_j with down_1 line.
- (7) Swap the positions of L_i and L_j in LIST.
- (8) Compute two new intersection pointers formed by the newly positioned L_i with its new adjacent up_1 line and the newly positioned L_j with its new adjacent down_1 line.
- (9) Insert these two new intersection points into HEAP.

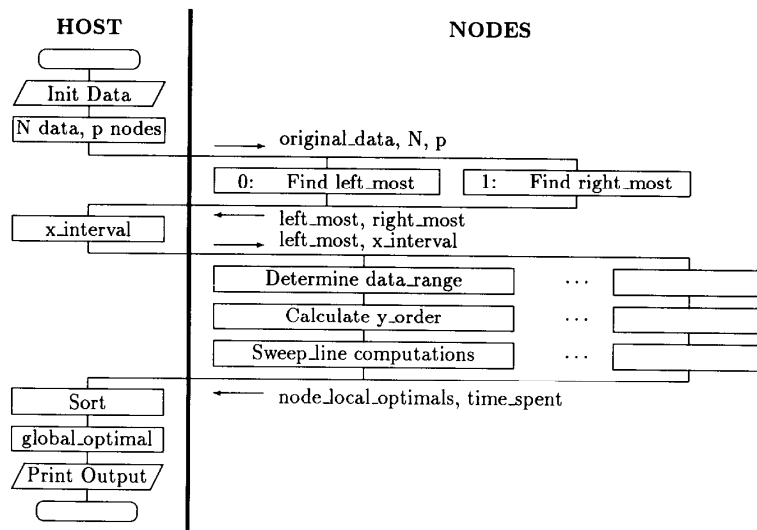
- (10) Go to 1 if HEAP is not empty, that is that if the point in the root of HEAP has not reached the rightmost intersection point (includes the rightmost one).

For each of the $O(n^2)$ new positions of all of the lines involved, the algorithm needs to update only three points of HEAP at a cost of $O(\log n)$ time each. Thus, the entire algorithm runs in time $O(n^2 \log n)$.

The sweep-line algorithm seems to be a very sequential technique. Efficient parallel algorithms can be developed for a given set of line segments on the plane (Atallah and Goodrich, 1986). However, the sweep-line algorithm for LMS is based on a given set of data points. The computations will be performed on the mapped plane, in which the complete set of line segments does not exist. The intersection points of the lines are produced dynamically as the computation proceeds. In other words, every computation needs global information and the next computation depends on the result of the previous step. Thus, it is impossible to statically partition a given set of data on the mapped plane.

The bottleneck can be resolved by observing that the structure LIST is used to maintain the line order along the y -axis, while the structure HEAP is used to maintain the intersection point order along the x -axis. It is clear that on the left plane of the leftmost intersection point, all of lines are ordered along y -axis in increasing order of slopes. Similarly, on the right plane of the rightmost intersection point, all the lines are ordered along the y -axis in decreasing order of slopes. Whenever passing an intersection point, two lines which produced this intersection point need to be swapped. The operation maintains the line order along the y -axis. That is, for an x -coordinate, the lines have a specific order on their y -coordinate values. We call it *y-order*. If the mapped plane is partitioned into several pieces, we can use the x -coordinate of the left boundary of each partition to find the y -order of lines on the left boundary. Then each partition has its own leftmost intersection point and each partition can be swept from its left boundary to its right boundary. This is the basis of the parallel sweep-line algorithm.

- (1) Calculate the leftmost intersection point by ordering the lines in ascending order of slopes. Calculate the rightmost intersection point by ordering the lines in descending order of slopes.
- (2) Divide the x -distance between the leftmost intersection point and the rightmost one by the number of node processors to get the x -interval.
- (3) Send the leftmost point and the x -interval to each of node processors. Each node processor calculates the left boundary based on the formula (leftmost + node number $\times$ x -interval).
- (4) Take the x -coordinate of the left boundary to calculate the y -coordinates of the intersection points formed by the left boundary and all of lines, sort these y -coordinates to find its y -order.
- (5) Store the lines in LIST according to the y -order (thus, each node processor has the same data structure, but different initial line order).



- (6) Each node processor then performs in parallel the sweep-line algorithm mentioned above.

The parallel sweep-line algorithm is depicted in Figure 4. We can write analytical expressions for the sequential and parallel sweep-line algorithm and predict the speedup of the parallel algorithm. Observing Figure 4 and Figure 3, we may say that the parallel sweep-line algorithm will have as good results as the parallel Rousseeuw’s algorithm. Unfortunately, the results of the experimental computation shown in Section 5 do not support this observation since the number of intersection points in each partition is dynamically produced. The experimental computation indicated that almost 99% computation load for the given data concentrates in one node due to severe potential load imbalancing among node processors.

Effort has been made to distribute the working load more evenly. The refined algorithm allows the users to divide the mapped plane into more partitions than node processors available, for example, 16 partitions for 8 nodes. Initially all of 8 nodes work on the first 8 partitions. Whenever one node finishes its computation on one partition, the host submits another partition to it until all of the partitions are processed. This refined partition method does not give much better load balancing but also create additional overhead for reordering the y-order so that the performance is not improved.

4. Tukey's idea

Based on a suggestion of Tukey (Steel and Steiger, 1986, p. 99), an algorithm (named T-sweep algorithm) which statically partitions the original data instead of partitioning the mapped data has been implemented. The algorithm partitions the data by randomly chosen subsets, each having size $\lceil x \rceil$ which is determined by $\binom{x}{2} = \binom{n}{2}/p$. Here p is the number of nodes and n is the size of the original data. Sweep-line algorithm is applied to each subset of the original data to compute exact LMS estimates. An approximate LMS fit is then obtained by averaging the exact fits from subsets.

The sequential sweep-line algorithm requires t_{ss} time to complete its computation on a set of data n , given by

$$t_{ss} = nt_{mp} + \binom{n}{2}t_{one_point} + \Phi(n, p) + \Phi(1, 1), \quad (5)$$

where: t_{mp} – time spent for mapping one point to the mapped plane and initialize two data structures LIST and HEAP; t_{one_point} – time spent for the computation of one intersection point on the mapped plane.

The T-sweep parallel algorithm will spend the time t_{sp} for completing its computation on the given set of n data,

$$t_{sp} = xt_{mp} + \binom{x}{2}t_{one_point} + \Phi(n, p) + \Delta(1, p), \quad (6)$$

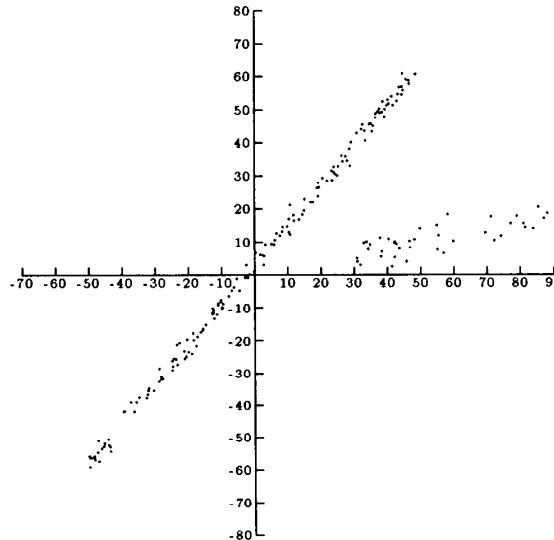


Fig. 5. An input data set of 200 with 80/20% composition.

where $\binom{x}{2} = \binom{n}{2}/p$. The speedup of T-sweep algorithm, thus, is:

$$s = \frac{t_{ss}}{t_{sp}} = \frac{nt_{mp} + \binom{n}{2}t_{\text{one_point}} + \Phi(n, p) + \Phi(1, 1)}{xt_{mp} + \frac{\binom{n}{2}}{p}t_{\text{one_point}} + \Phi(n, p) + \Delta(1, p)} \propto p \quad (7)$$

The value s is very closely proportional to the number of node processors p used since the items $\binom{n}{2}t_{\text{one_point}}$ and $(1/p)\binom{n}{2}t_{\text{one_point}}$ dominate the computation time.

5. Speedup results of an experimental computation

This section gives results of the experimental computation based on two sets of data generated according to the formulas

- (1) $y_1 = 2.1 + 1.2x_1 + \epsilon$,
where $x_1 \sim \text{UNI}(-0.5, 0.5)$ and $\epsilon \sim \text{N}(1, 2)$;
- (2) $y_2 = 0.5 + 0.2x_2 + \epsilon$,
where $x_2 \sim \text{UNI}(0.5, 1.5)$ and $\epsilon \sim \text{N}(0.1, 3)$

One set of data consists of 80% of (x_1, y_1) and 20% of (x_2, y_2) ; the other set consists of 60% of (x_1, y_1) and 40% of (x_2, y_2) . Figure 5 depicts an example of 200 data with 80/20% composition.

The results shown in Table 1 confirm the robustness of LMS regression. The composition percentage of two sets of data is changed from 80/20% to 60/40%, but the slopes and intercepts from the two sets remain close.

The speedups in Table 2 indicate that the parallel capacity of the Rousseeuw's algorithm can be fully exploited. Computations within each node do not have synchronization point. Communication overhead increases with number of nodes, but it can be offset by the increase in number of data points. This explains the gains in speedup, for fixed number of nodes, when n goes from 50 to 200. Certainly this is a very desirable feature of the parallel algorithm.

Table 3 shows that the speedup of parallel sweep-line algorithm is less than 1.00. The major reason is the computation load imbalance. The number of computations done in each node is shown in the Table 4.

From Table 4 we can see that for the set of 50 data, 1207 out of 1224 (98.6%) computations are performed in node 5. The proportion for the data set of size

Table 1
The robustness of LMS regression

	$n = 50$		$n = 200$	
	80/20%	60/40%	80/20%	60/40%
slope	1.22177	1.23035	1.21436	1.21415
intercept	2.87680	2.95833	3.23174	2.63158

Table 2

Experimental computation results of Rousseeuw's algorithm

nodes	$n = 50, 80/20\%$		$n = 200, 80/20\%$	
	node time	speedup	node time	speedup
1	11682	1.00	1028481	1.00
2	5846	1.99	515041	1.99
4	3014	3.87	262224	3.92
8	1697	6.88	132018	7.79

Table 3

Experimental computation results of sweep-line algorithm

nodes	$n = 50, 80/20\%$		$n = 200, 80/20\%$	
	node time	speedup	node time	speedup
1	938	1.00	16468	1.00
2	1076	0.93	18536	0.89
4	1076	0.92	18423	0.90
8	1056	0.92	18390	0.90

200 is 99.9% (19880/19897) done in node 4. The sweep-line algorithm needs extra computations for leftmost and rightmost intersection points (see Figure 4), and the data structures the sweep-line algorithm used consume more time for

Table 4

Load imbalancing in sweep-line algorithm (8 nodes, 8 partitions)

Data	node 0	node 1	node 2	node 3	node 4	node 5	node 6	node 7
$n = 50$	1	0	1	1	11	1207	3	0
$n = 200$	1	0	2	0	19880	12	0	2

Table 5

Total time and computations is each node of refined partitioning method

Data	part.	time	node 0	n1	n2	n3	n4	n5	n6	n7
$n = 50$	8	1616	1	0	1	1	11	1207	3	0
	16	1845	8	0	2	1200	9	3	1	1
	32	1708	9	3	0	4	3	292	908	5
	64	2113	14	8	11	281	4	4	897	5
	128	2962	889	8	6	13	11	37	248	12
$n = 200$	8	19498	1	0	2	0	19880	12	0	2
	16	19766	6	0	9	19871	1	3	7	0
	32	19315	6	1	7	19388	3	2	484	6
	64	19699	21	27	3	5	8	467	4	19362
	128	20514	19314	13	10	437	15	51	34	23

Table 6
Experimental computation results of T-sweep algorithm

data	nodes	slope	intercept	node time	speedup
$n = 50$	1	1.19551	2.65209	938	1.00
80/20%	2	1.24046	4.76830	483	1.94
	4	1.24206	4.80664	248	3.78
	8	1.12025	4.74954	133	7.05
$n = 200$	1	1.21151	3.25983	16468	1.00
80/20%	2	1.19636	2.91181	8700	1.89
	4	1.21039	2.42001	4267	3.85
	8	1.21620	2.57820	2075	7.93

initializing than the one used by Rousseeuw's algorithm. This overhead further increases the computation time.

One effort to resolve the load imbalance is the refined partitioning method mentioned in Section 3. However, the computation experiences reported in Table 5 do not support the method as a solution to the problem.

Table 6 indicates that the T-sweep algorithm is much better than other algorithms since it takes advantage of both a good algorithm $O(n^2 \log n)$ and a balanced load. The major disadvantage of the T-sweep algorithm is that it only gives approximate results. The slope and intercept corresponding to the 1 node case are exact fit. The rest of the cases give approximate fits obtained by averaging the exact fits from partitions. The T-sweep algorithm seems to provide good approximate results when n is large. This is precisely where it is needed. Of course, a genuine Monte Carlo study will improve the approximate results and a further theoretical study is required to validate the procedure.

The performance of sweep-line algorithm presented here may be the worst possible case that can be encountered. Recently Stromberg (1991) has pointed out that LMS computation can be reduced to finding Chebyshev (minimax) solution of inconsistent linear equations. The complexity of the algorithm is $O(n^3)$, but the average time complexity can be improved. Further work using Stromberg's result seems promising.

6. Acknowledgements

We are pleased to thank the Editor and anonymous referees for their invaluable suggestions and comments. We are also grateful to SIUE for providing the Special Research Initiative Award that supported this research.

References

- Atallah, M.J. and M.T. Goodrich, Efficient parallel solutions to some geometric problems, *Journal of Parallel and Distributed Computing*, **3** (1986) 492–507.

- Bailey, D.H., Future requirements for effective use of parallel computers, in: J.L.C. Sanz (Ed), *Opportunities and Constraints of Parallel Computing* (Springer-Verlag, Berlin, 1989) 5–7.
- King, C.-T., W.-H. Chou and L.M. Ni, Pipelined data-parallel algorithms, *IEEE Trans. on Parallel and Distributed Systems*, **1** (4) (1990) 470–499.
- Leroy, A. and P.J. Rousseeuw, PROGRES: A program for robust regression, (Research Report 201, University of Brussels, Centrum Voor Statistiek en Operationeel Onderzoek, 1984).
- Rousseeuw, P.J., Least median of squares regression, *Journal of the American Statistical Association*, **79** (388) (1984) 871–880.
- Rousseeuw, P.J. and A. Leroy, *Robust Regression and Outliers Detection*, (John Wiley, New York, 1987).
- Souvaine, D.L. and J.M. Steele, Time- and space-efficient algorithms for least median of squares regression, *Journal of the American Statistical Association*, **82** (399) (1987) 794–801.
- Steele, J.M. and W.L. Steiger, Algorithms and complexity for least median of squares regression, *Discrete Applied Mathematics*, **14** (1986) 93–100.
- Stromberg, A.J., Computing the exact value of the least median of squares estimate in multiple linear regression, (Technical Report #561, Department of Statistics, University of Minnesota, MN, 1991).