Parallel Computing 16 (1990) 183–190
North-Holland

Distributed selectsort sorting algorithms on broadcast communication networks

Jau-Hsiung HUANG * and Leonard KLEINROCK **

* *Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.*

** *Computer Science Department, University of California, Los Angeles, California, USA*

Received 14 May 1990
Revised 16 July 1990

Abstract. In this paper, a distributed *selectsort* algorithm and a *parameterized selectsort* algorithm are presented to be applied on distributed systems for cases when $N \gg P$ where N is the number of elements to be sorted and P is the number of processors in the system. The distributed system considered in this paper uses a broadcasting channel for communication between processors. We show that the number of messages required for the parameterized selectsort algorithm is independent of N and is of complexity $O(P)$, which is optimal in a distributed system with P processors. Furthermore, the amount of communication required in terms of elements is $N + O(P^3)$ and the computation time complexity is $O((N/P)\lg N + P^2 \lg(N/P))$. Hence, when $N \geqslant P^3$, the computation time complexity is $O((N/P)\lg N)$, which is optimal using P processors. In addition, this parameterized algorithm provides us with a parameter K such that by choosing the value of K allows us to trade among processing requirement, memory requirement, and communication requirement. It is shown that this parameterized algorithm can reduce the communication requirements significantly while only slightly increasing the computation requirements.

Keywords. Broadcast, Communication bit complexity, Communication element complexity, Communication message complexity, Computation time complexity, Delimiter.

1 Introduction

For algorithms applied on a distributed system, the time required by the algorithm normally depends on two issues. One issue is the *computation* time requirement and the other is the *communication* time requirement. In many distributed systems, the time taken by computation is much less than the time taken by communication; hence, the communication requirement is usually accepted as the performance measure of distributed algorithms.

Further, the time spent on communication has two major factors. One factor is the time required to prepare messages for sending and to process messages after receiving. We call this the *message processing time*. The message processing time includes packetizing a message before sending and unpacketizing a message after receiving and the time spent on error detection and recovery. Another factor is the time required in transmitting messages. We call this the *message transmission time*, which is the time required for a message to travel across the communication network. Hence, the message processing time required by an algorithm depends on the number of messages created by the algorithm and the message transmission time required by an algorithm depends on the number of bits transmitted by the algorithm. In some systems, the message processing time is no smaller than the message transmission time. Therefore, in measuring the communication complexity of an algorithm, we have to look at both the message

processing time requirement as well as the message transmission time requirement. That is, we would like to reduce both the number of messages required by the algorithm and the number of bits required to be transmitted among processors. In other words, if the number of bits required to be transmitted is the same, the performance of the algorithm will be better if fewer messages are incurred.

The distributed selectsort sorting algorithm presented in this paper wants to sort N distinct elements with P processors assuming that these N elements are evenly distributed among all P processors, i.e. there are N/P elements in each of the processors. We assume there are P processors and P_i denotes the ith processor. Without loss of generality, we assume all N elements are distinct in this paper. In cases when there are elements with the same value, we can append the station number to the element to break the equality. The purpose of this algorithm is to sort these N elements and to store the result in the P processors. That is, all elements in each processor are sorted; further, all elements in P_i are smaller than all elements in P_{i+1}. Hence, P_p contains the largest N/P elements. Therefore, the sorted N elements are placed in processors P_1 to P_p in an increasing order.

For the rest of the paper, we denote the number of messages required to be transmitted by an algorithm as the *communication message complexity* and denote the number of bits required to be transmitted as the *communication bit complexity*. As usual, we denote the time required for computation as the *computation time complexity*. Further, since the basic entity considered in this paper is an element, we define *communication element complexity* as the number of elements required to be transmitted by the algorithm. Clearly, if there are N elements to be sorted, each element can be represented by $k = \lg N$ bits, then the communication bit complexity is simply k times of the communication element complexity. In this paper, we will use the communication element complexity as the performance measure instead of using the communication bit complexity.

There have been many works in distributed sorting algorithms [1–7] among which [1] and [2] will be briefly described here since they are also applied on a broadcast network. [1] gave an algorithm which made use of a broadcast communication network to implement a distributed sorting algorithm. The advantage of their algorithm was that, regardless of the number of processors used, the algorithm had an average communication element requirement as $\frac{3}{2}N$. Note that sorting in a broadcasting network requires a communication element requirement of at least N since every element must be broadcasted at least once in the worst case so that it can be sent to the processor where it should reside in the final sorted distributed lists. The disadvantage of this algorithm is that each broadcast message contains only one element. Hence, this algorithm incurs $\frac{3}{2}N$ messages.

[2] gave another distributed sorting algorithm which also made use of a broadcast communication network using more than one channel. This algorithm achieved higher concurrency by using multiple communication channels. However, this algorithm had a communication element requirement of $4N$ and required $O(N)$ messages.

For the parameterized selectsort sorting algorithm presented in this paper, the communication message complexity is $O(P)$ and the communication element requirement is $N + O(P^3)$. For cases when $N \gg P$, our algorithm requires much less communication requirement than those in [1] and [2]. Moreover, the computation time complexity is $O((N/P)\lg N + P^2 \lg(N/P))$; hence, when $N \geqslant P^3$, the computation time complexity is $O((N/P)\lg N)$, which is optimal using P processors.

This paper is organized as follows. In Section 2, we first present a distributed concurrent selection algorithm. This algorithm is later applied in Section 3 to obtain a distributed selectsort sorting algorithm where the number of messages sent is much smaller than that in [1] and [2]. We then modify this algorithm into a parameterized algorithm in Section 4. The parameter provided in the algorithm allows us to trade among communication requirement,

computation requirement, and memory requirement. It is shown that a slight increase in computation requirement saves us a lot in communication requirement. The conclusion is presented in Section 5.

2 The distributed concurrent selection algorithm

We first describe the distributed concurrent selection algorithm. We assume that each processor contains N/P sorted elements in its local list. This algorithm will select concurrently $P-1$ elements with $P-1$ specified rankings. Some similar works are presented in [8–10]. However, our algorithm selects $P-1$ elements concurrently.

The approach used in this algorithm is a combination of *counting* and *binary searching*. We first explain how counting is used. If P_i wants to find out the overall ranking of its mth locally ranked element with value X, it broadcasts X to all other processors using the communication network. Every processor except P_i will then find out how many elements in its local list are smaller than X and will then send this number in a message to P_i through the broadcast network. After receiving all these $P-1$ messages, P_i adds up all these numbers from these messages to find out how many elements stored in other processors are smaller than X. It then adds m to this number to obtain the overall ranking of X.

In the following we show how binary searching is used. If P_i wants to find out whether its local list contains an element with an overall ranking n, P_i first takes its entire local list as the working list and finds out the overall ranking of the median element in the working list using the counting method just described. If the overall ranking of the median element is greater than n, then we take the first half of the current working list as the new working list and repeat the process. Or, if the overall ranking of the median element is smaller than n, then we take the second half of the current working list as the new working list and repeat the process. This procedure repeats until either the element with an overall ranking n is found or P_i finds out that its local list does not contain the element with an overall ranking n.

To accomplish the selection in this algorithm to find $P-1$ elements with $P-1$ specified rankings concurrently, every processor sends messages in turn according to its station number until the algorithm finishes. Each message sent contains $P(P-1)$ elements grouped into $P-1$ fields with P elements in each field. We denote these $P(P-1)$ elements as $E_{11}, E_{12}, \ldots, E_{1P}, E_{21}, E_{22}, \ldots, E_{2P}, \ldots, E_{(P-1)1}, E_{(P-1)2}, \ldots, E_{(P-1)P}$. The first subscript denotes the field number and the second subscript denotes the ordering of that element in that field. The P elements in the ith field ($1 \leqslant i \leqslant P-1$) in each message are used to locate the element with the ith specified ranking. Since we need to find $P-1$ elements with specified rankings concurrently, we have $P-1$ fields in each message.

The content of element E_{kj} sent by P_i may have two different meanings depending on whether j equals i or not. Element E_{ki} sent by P_i contains an element in P_i's list which P_i likes to find out the ranking of this element among all elements in all processors. P_i achieves this goal using the counting method described above after receiving the following $P-1$ messages. For element E_{kj}, $j \neq i$, sent by P_i, it contains a number which specifies that how many elements in P_i's list is smaller than the element E_{kj} sent by P_j in P_j's last broadcast. This will help P_j in finding the overall ranking of E_{kj} broadcast by P_j in its last broadcast.

To find out the overall ranking of element E_{ki} sent by P_i, P_i simply adds up all the elements E_{ki} in the following $(P-1)$ broadcast messages sent by $P_j (j \neq i)$ and the ranking of E_{ki} in P_i's local list. Whenever a processor, say P_i, finds that element E_{ki} has the kth prespecified ranking, it broadcasts a special message in its next broadcasting. This special message places the element with the kth pre-specified ranking, which is just found, in E_{ki} and a '&' in each

16	39	13	22
19	69	28	31
61	91	47	43
85	122	115	58
109	181	130	64
202	211	147	67
208	232	151	79
226	247	153	101
244	250	175	107
253	259	187	124
269	274	199	133
282	289	205	145
286	304	235	160
311	313	241	169
316	326	262	184
322	331	265	191
329	337	271	196
343	341	292	217
347	352	297	223
349	361	301	228
P1	P2	P3	P4

Fig. 1. Input data for Example 1.

element E_{kj} $(1 \leqslant j \leqslant P,\ j \neq i)$ to let all other processors know that the element E_{ki} is the element with the kth pre-specified ranking to be found.

Example 1. In this example we show how messages are sent using this algorithm assuming $N = 80$ and $P = 4$. The input data is shown in *Fig. 1* and we want to select the 20th, the 40th, and the 60th elements concurrently. The messages sent using this algorithm are shown in *Fig. 2*. As shown in *Fig. 2*, we find that 109, 205, and 282 are the elements to be found.

Lemma 1. *The communication element complexity of the distributed concurrent selection algorithm is* $O(P^3 \lg(N/P))$. *In addition, the communication message complexity is* $O(P \cdot \lg(N/P))$.

Proof. We define a *run* to be that every processor broadcasts in turn once; hence, there are P messages per run. Using binary searching, the number of runs required to complete the algorithm is upper bounded by $\lg(N/P)$. Hence, the total number of messages sent has a complexity of $O(P \cdot \lg(N/P))$. Since each message contains $P(P-1)$ elements; hence, the total number of elements sent across the network in Step 2 has a complexity of $O(P^3 \cdot \lg(N/P))$. □

Lemma 2. *The computation time complexity of the distributed concurrent selection algorithm is* $O(P^2 \cdot \lg^2(N/P))$.

Proof. Before a message is sent, the value of each element has to be determined. For P_i to determine the value of each $E_{ki}(1 \leqslant k \leqslant P-1)$, P_i has to add P number together, which has a computation complexity of $O(P)$. Since there are $P-1$ such E_{ki}s, the total computation complexity incurred is $O(P^2)$. For P_i to determine E_{kj} $(1 \leqslant k \leqslant P-1$ and $j \neq i)$, P_i has to do a binary searching, which has a computation complexity of $O(\lg(N/P))$. Since there are $(P-1)^2$ such E_{kj}s, the total computation complexity incurred is $O(P^2 \cdot \lg(N/P))$. Hence, the computation complexity incurred by a message is $O(P^2 \cdot \lg(N/P))$. Since the number of

P1	253	---	---	---	253	---	---	---	253	---	---	---
P2	9	259	---	---	9	259	---	---	9	259	---	---
P3	14	14	187	---	14	14	187	---	14	14	187	---
P4	20	20	15	124	20	20	15	124	20	20	15	124
P1	109	10	5	5	109	10	5	5	316	10	5	5
P2	3	181	5	4	3	181	5	4	14	326	5	4
P3	3	9	130	4	3	9	262	4	20	20	262	4
P4	9	14	10	64	9	14	20	184	20	20	20	184
P1	109	&	&	&	208	5	10	5	282	16	10	5
P2	&	&	&	&	5	232	10	5	11	289	10	5
P3	&	&	&	&	12	12	205	9	17	17	271	9
P4	&	&	&	&	17	20	17	196	20	20	20	196
P1	&	&	&	&	202	8	6	5	282	&	&	&
P2	&	&	&	&	5	211	5	5	&	&	&	&
P3	&	&	&	&	&	&	205	&	&	&	&	&

Fig. 2. The messages sent in Example 1. Each row represents a message.

messages required by the algorithm is bounded by P lg(N/P), the total computation complexity is O($P^3 \cdot \lg^2(N/P)$). However, since all P processors work concurrently at all time, hence the computation time complexity is O($P^2 \cdot \lg^2(N/P)$). □

3 The distributed selectsort sorting algorithm

The approach of this sorting algorithm is first to find the (iN/P)th ranked elements for all i from 1 to $P-1$. We define these elements as the *delimiters*. Then, elements with values between the values of the $(i-1)$th and the ith delimiters will be sent to P_i for sorting by each processor to complete the algorithm. Accordingly, this algorithm is partitioned into 4 steps as follows. Note that if all P processors can be working at the same time in a step in the algorithm, we add *concurrently* in that step.

Algorithm

Step 1. For $i=1$ to P do *concurrently* P_i sorts its local list using a known optimal sequential sorting algorithm (e.g. quicksort).

Step 2. For $i=1$ to $(P-1)$ do *concurrently* find the (iN/P)th element (i.e. the ith delimiter) using the distributed concurrent selection algorithm described in Section 2.

Step 3. For $i=1$ to P do sequentially P_i broadcasts all elements which should be sent to other processors in a message. At the same time, P_j listens to the broadcasts, and copies all elements with values between $(j-1)$th and the jth delimiters to its local list.

Step 4. For $i=1$ to P do *concurrently* P_i merges all the P sublists sent to it into one sorted list.

Theorem 1. *The computation time complexity of the algorithm is* O($(N/P)lgN + P^2\lg^2(N/P)$) *and the communication element complexity is* O($N + P^3\lg(N/P)$).

Proof. The computational time complexity of Step 1 is $O((N/P)\lg(N/P))$ for each processor since the list in each processor has N/P elements. The computation time complexity and the communication element complexity of Step 2 are provided in Lemmas 1 and 2. In Step 3, each element will be broadcast at most once from the processor it initially resides to the processor it should finally reside. Hence, the communication element complexity is at most N. The computation time complexity of Step 3 is also $O(N)$ to prepare the messages for all elements. The computation time complexity of Step 4 is $O((N/P)\cdot \lg P)$ for each processor since each processor simply merges P sorted sublists such that the computation time complexity is $O((N/P)\cdot \lg P)$. □

Theorem 2. *The communication message complexity of the sorting algorithm is* $O(P\lg(N/P))$.

Proof. We now find the number of messages required by Steps 2 and 3. Step 2 requires $O(P\cdot \lg(N/P))$ messages as stated in Lemma 1. For Step 3, each processor broadcasts a message containing all elements to be sent to other processors. Each processor extracts the part of elements destined for it. Hence, Step 3 requires P messages. Therefore, the total number of messages required by the algorithm is $O(P\lg(N/P)+P) = O(P\lg(N/P))$. □

One advantage of this algorithm is that in Steps 2 and 3 we can send multiple elements per message. As mentioned in the Introduction, this reduces the communication processing time significantly. Also note that all communication is well scheduled so that no collisions will occur if the algorithm is applied on a multi-access broadcast channel. (A multi-access broadcast channel is a broadcast communication channel which can be accessed by all stations. However, a collision will occur if two or more stations transmit at the same time. The well known Ethernet is an example of a multi-access broadcast channel.) Nevertheless, there are some inefficiencies in Step 2 which will be improved in the following section.

4 The distributed parameterized selectsort sorting algorithm

In this section we make a slight change in Step two of the previous algorithm to obtain a parameterized sorting algorithm which reduces the communication requirement dramatically with a slight sacrifice in computation. Since communication is more expensive than computation, this modification gives us a big save. The most significant effect of this parameterized algorithm is that the communication message complexity of Step 2 is no longer a function of N. This makes our algorithm a very good algorithm for large N.

Step 2 is modified such that we do not have to find exactly the (iN/P)th element as the ith delimiter. Rather, we accept any element whose ranking is between $[iN/P-(N/P)\cdot K/100]$ and $[iN/P+(N/P)\cdot K/100]$ as the ith delimiter for a chosen parameter $K(0\leqslant K\leqslant 50)$. Hence, we choose the ith delimiter as the first element found in Step 2 with a ranking between $[iN/P-(N/P)\cdot K/100]$ and $[iN/P+(N/P)\cdot K/100]$ for $1\leqslant i\leqslant P-1$.

Algorithm

Step 1. For $i=1$ to P do *concurrently* P_i sorts its list using a known optimal sequential sorting algorithm.

Step 2. Select a value between 0 and 50 for 'K', for $i=1$ to $(P-1)$ do *concurrently* find the first element with a ranking between $[iN/P-(N/P)\cdot K/100]$ and $[iN/P+(N/P)\cdot K/100]$ as the ith delimiter using the distributed concurrent selection algorithm.

Step 3. For $i=1$ to P do sequentially P_i broadcasts all elements which should be sent to other processors in a message. At the same time, P_j listens to the broadcasts, and copies all elements with values between the $(j-1)$th and the jth delimiters to its local list.

Step 4. For $i=1$ to P do *concurrently* P_i merges all the sublists sent to it into one sorted list.

Lemma 3. *The number of runs required in Step 2 of the parameterized algorithm is upper bounded by* $\lg 50/K$.

Proof. We can group the N elements into $100/2K$ groups where each group contains $N \cdot 2K/100$ elements. Whenever the algorithm finds any of the elements in a group containing the element with the (iN/P)th ranking, this element is regarded as the ith delimiter. Hence, using binary searching on these $100/2K$ groups, the number of searches is upper bounded by $\lg 50/K$. □

Lemma 4. *The communication element complexity of Step 2 of the parameterized algorithm is* $O(P^3)$.

Proof. Since each run contains P messages and each message contains P^2 elements and the number of runs is upper bounded by $\lg 50/K$; hence, the communication element complexity is $O(P^3)$. □

Lemma 5. *The computation complexity of Step 2 is* $O(P^2 \lg(N/P))$.

Proof. This proof follows directly from Lemmas 2 and 3. □

Note that the communication message complexity and the communication element complexity are independent of N; hence the communication processing time can be greatly reduced for large N. However, by this saving in communication, we also incur more computation in Step 4. In Step 4, all processors no longer contain the same number of elements to be sorted as in the original algorithm. Nonetheless, the maximum number of elements to be sorted given to a processor is at most $(1 + 2K/100)\ N/P$. Therefore, the computation time required for Step 4 is at most $2K\%$ more than the original algorithm. Moreover, the memory requirement will also be increased to store these elements. However, the computation time complexity remains unchanged since K is a constant. Furthermore, note that the increase in computation time complexity in Step 4 is dominated by the computation time complexity in Step 1. Hence, this increase in computation in Step 4 has no effect on the overall computation time complexity.

Another advantage of this modification is that by adjusting the value of K, we are able to trade between communication requirement and computation and memory requirement. A larger K will incur less communication and more computation and memory than a smaller K. Moreover, by taking a system configuration (e.g. computational power, communication bandwidth, etc.) into consideration, we can balance the computation requirement and communication requirement by adjusting the value of K.

Theorem 3. *The computation time complexity of the parameterized algorithm is* $O((N/P)\lg N + P^2 \lg(N/P))$ *and the communication element complexity is* $O(N + P^3)$.

Proof. The proof can easily be derived from Theorem 1 and Lemmas 4 and 5. □

Corollary 1. *If* $N > P^3$, *then the computation time complexity is* $O((N/P)\lg N)$ *and the communication element complexity is* $O(N)$. *Both are optimal.*

Theorem 4. *The communication message complexity of the parameterized algorithm is* $O(P)$.

Proof. We now find the number of messages required by Steps 2 and 3. Step 2 requires $O(P \lg(100/K)) = O(P)$ messages. As explained in the proof of Theorem 2, the number of

messages required by Step 3 is at most P. Therefore, the total number of messages required by the algorithm is $O(P)$. □

5 Conclusions

The contribution of the parameterized selectsort sorting algorithm is that the number of messages sent during the algorithm depends only on P and not on N. This will significantly reduce the communication processing time in a broadcast network when N is large. Additionally, we show that when $N \geqslant P^3$, the computation time complexity is $O((N \lg N)/P)$, which is optimal with P processors. More importantly, in the parameterized algorithm, we provide a parameterized way to trade-off among communication requirement, processing requirement, and memory requirement. This allows the algorithm to be fine-tuned under various system configurations.

References

[1] R. Dechter and L. Kleinrock, Broadcast communications and distributed algorithms, *IEEE Trans. Comput.* C-36 (3) (March 1986) 210–219.

[2] J. Marberg, Distributed algorithms for multi-channel broadcast networks, Ph.D. dissertation, Computer Science Department, UCLA, 1986.

[3] D. Rotem, N. Santoro and J.B. Sidney, Distributed sorting, *IEEE Trans. Comput.* C-34 (4) (April 1985) 372–376.

[4] L.M. Wegner, Sorting a distributed file in a network, in: *Proc. 1982 Conf. Information Sci. Systems*, Princeton, New Jersey (March 1982) 505–509.

[5] S. Zaks, Optimal distributed algorithms for sorting and ranking, *IEEE Trans. Comput.* C-34 (4) (April 1985) 376–379.

[6] S.P. Levitan, Algorithms for a broadcast protocol multiprocessor, in: *Proc. 3rd Internat. Conf. on Distributed Computing Systems*, (1982) 666–671.

[7] K.V.S. Ramarao, Distributed sorting on local area networks, *IEEE Trans. Comput.* 37 (2) (Feb. 1988) 239–243.

[8] J. Marberg and E. Gafni, An optimal shout-echo algorithm for selection in distributed sets, in: *Proc. 23rd Ann. Allerton Conf. on Communication, Control, and Computing,* Univ. of Illinois at Urbana Champaign (1985) 283–291.

[9] D. Rotem, N. Santoro and J.B. Sidney, A shout-echo algorithm for finding the median of a distributed set, in: *Proc. 14th S.E. Conf. on Combinatorics, Graph Theory and Computing,* Boca Raton, FL (1983) 311–318.

[10] N. Santoro and J.B. Sidney, A reduction technique for distributed selection: I, Tech. Rep. SCS-TR-23, School of Computer Science, Carleton Univ., Ottawa, Canada, 1983.