# PARALLEL ALGORITHMS FOR MULTIPROCESSORS USING BROADCAST CHANNEL

Rina Dechter
Leonard Kleinrock
(Formerly Working Paper No. 81002
November 30, 1981)

November 1981
CSD-850025

# WORKING PAPER No. 81002

PARALLEL ALGORITHMS FOR MULTIPROCESSORS

USING BROADCAST CHANNEL

by

RINA DECHTER

& LEONARD KLEINROCK

11/30/81

Advanced Teleprocessing Systems Group
Computer Science Department
University of California
Los Angeles, California 90024

# PARALLEL ALGORITHMS FOR MULTIPROCESSORS USING BROADCAST CHANNEL*

by

Rina Dechter

## ABSTRACT

Two parallel algorithms which use broadcast communication are presented. One algorithm determines the maximum element in a list of k elements, using k processors; the other sorts a list of n elements into a nonincreasing order using k processors. We show that the average processing time of the Max algorithm is O(log k) and the worst case processing time for the SORT algorithm is $O\left(\frac{n}{k}\log\frac{n}{k}+2n-1\right)$, i.e., for large n, the SORT algorithm achieves an asymptotic speed-up ratio of k with respect to the best sequential algorithm.

---

*This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-77-C-0272.

## Two Parallel Algorithms for Broadcast Communication

## INTRODUCTION

In the following pages we are going to discuss two parallel algorithms which take advantage of broadcast communication. The first finds the maximum of k elements using k processors -- this is the MAX Algorithm. The second algorithm utilizes the MAX Algorithm to sort a list of n elements into a nonincreasing order using k processors. For simplicity we assume that all the elements are distinct. The new feature of these algorithms is that they assume BROADCAST Communication among processors which implies that only one processor can transmit at one time and all others can hear the broadcasted message.

The algorithms we proposed are synchronized and can be categorized as a SIMD (Single Input stream, Multiple Data stream) with small module granularity (Kung 79).

We present an analysis of the complexity of the algorithms. We show that the average processing time of the MAX Algorithm is o (log k) and the worst case processing time for the SORT Algorithm is $o(\frac{n}{k}\log\frac{n}{k} + 2n-1)$ i.e. for large n the SORT-Algorithm achieves an asymptotic speed-up ratio of k with respect to the best sequential algorithm, which is optimal in the number of processors used. Another parallel sort Algorithm which exhibit the same performance was presented by Baudet et. al. (Baudet, 78), however, with a different communication scheme among processors.

## 1. MAX ALGORITHM VIA BROADCASTING

### a. The algorithm

Machine: k+1 processors communicate by broadcasting, one of which is a designated as the ''Output processor''.

Input: a list of k distinct nummerically valued elements distributed among the k processors such that each processor has one element.

Output: the maximum value of the elements. This value is stored in the Output Processor.

Procedure: the procedure can be considered as a sequence of steps in which each processor performs two kinds of operations:

a. Listen to broadcasted values by other processors and compare them to its own value,

b. Broadcast its value when its turn comes only if it wasn't dominated by any of the values broadcasted previously.

The sequence ends when all processors have been given their chance to broadcast. The maximum value is the last one to be broadcasted. This value is recorded in the Output Processor which is listening to the whole process.

## b. Complexity of MAX Algorithm

We assume that an empty time slot (which occurs when a processor remains silent during its turn to broadcast) is much shorter then the time to perform a comparison. Thus in the analysis we choose the ''number of comparisons'' as the measure of complexity. This analysis does not take into account variations in computing time among processors, nor does it consider communication time. We see, however, that each ''real'' broadcast is followed by a comparison performed in parallel by some of the processors. Therefore the number of comparisons performed is a reasonable measure of complexity.

Let T(k) be the average number of comparisons performed by the algorithm. T(k) obeys the following recurrence equation,

$$T(k) = 1+\frac{1}{k}\sum_{i=1}^{k} T(k-i) \qquad (1)$$

since if the first element to be broadcast is the $i^{th}$ order element, then exactly i-1 among the rest of the k-1 processors will remain silent and will not participate in the rest of the process. We assume a homogeneous distribution of the elements among the processors and thus the above event has probability of 1/k and the recurrence follows. (1) could be rewritten as:

$$T(k) = 1+\frac{1}{k}\sum_{i=0}^{k-1} T(i)$$

with T(0)=0, T(1)=0. Solving this recurrence yields

$T(k) \leq \log k$

Example

assume we have k=8

| 25 | 17 | 34 | 45 | 12 | 70 | 83 | 75 |
|---|---|---|---|---|---|---|---|
| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ | $P_8$ |

The processors start broadcasting from left to right. In this case 5 processors will broadcast in the order $P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_6 \rightarrow P_7$. The series of elements broadcasted is a nondecreasing series 25, 34, 45, 70, 83 in which the last element to be broadcasted is the max (83). The number of ''real'' broadcasts performed is 5 in this case.

## 2. Parallel MERGE-SORT Via Broadcasting

### a. The algorithm

Machine: k+1 processors communicate by broadcasting, one of which is designated as the ''Output processor''.

Input: a list of n elements distributed among the k processors in such a way that each processor has n/k elements. For simplicity we assume here that n is a multiple of k (it can be extended easily to the general case).

Output: The sorted list available in the Output Processor.

Procedure: the procedure is decomposed into two phases - In phase

1 all processors in parallel sort their own lists. Any efficient sequential sorting algorithm can be used (Quick-sort, merge-sort). In Phase 2 all processors cooperatively participate in the task of merging the sorted lists they possess. This phase is decomposed into cycles. In each cycle the maximum of all the elements at the top of each processor is determined. This element is broadcast to the Output Processor as the next element in the Merged list and it is popped from the processor it belonged to. The implementation of each list in a processor could be by a stack. Each cycle is performed by the MAX-Algorithm presented before, however, there will be some dependency between cycles in the following way:

1) The first cycle is initiated by processor $P_1$.

2) The initiation of cycle i+1 depends on cycles 1 ··· i. Here we distinguish some cases:

   a) if cycle i consists of more than one broadcast, i.e. the cycle can be described by the processors participating in it $P_{i_1} P_{i_2} \cdots P_{i_{m-1}} P_{i_m}$ $m>1$ with the corresponding elements

$$a_{i_1} a_{i_2} \cdots a_{i_{m-1}} a_{i_m}$$

   $a_{i_m}$ is determined to be the Max for cycle i, it is popped from processor $P_{i_m}$ and is recorded as the next element in the Output Processor. In this case (m>1)

cycle i+1 will be initiated by processor $P_{i_{m-1}}$ which will rebroadcast the element $a_{i_{m-1}}$

b) m=1. There are two cases here

b.1) $a_{i_1}$ was not the first element in the first cycle it participated in

b.2) it was

b.1) In case b.1, we have cycle i composed of only one broadcast of element $a_{i_1}$ which is determined to be the Max of cycle i and is popped out of its processor and joins as the next element in the Merged List. In this case cycle i+1 will be initiated by the immediate predecessor of $P_{i_1}$ in the first cycle in which $P_{i_1}$ broadcast $a_{i_1}$.

b.2) If it was the first in its first cycle then cycle i+1 will be initiated by the same processor $P_{i_1}$, if it has any more elements, otherwise by the first processor to its right that has elements in its stack.

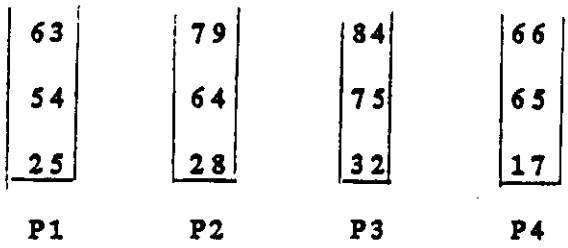
3. The algorithm ends when all processors have empty stacks.

The information needed to implement the above algorithm (phase 2) is simply that each processor will keep the identity of the processor who broadcast after itself in the most recent cycle

in which it participated. Each processor can also detect when a cycle ends and by which processor it was terminated ( a cycle is said to be terminated by processor $P_i$ if it was the last who broadcasted in this cycle), so he knows whether or not he should initiate the next cycle.

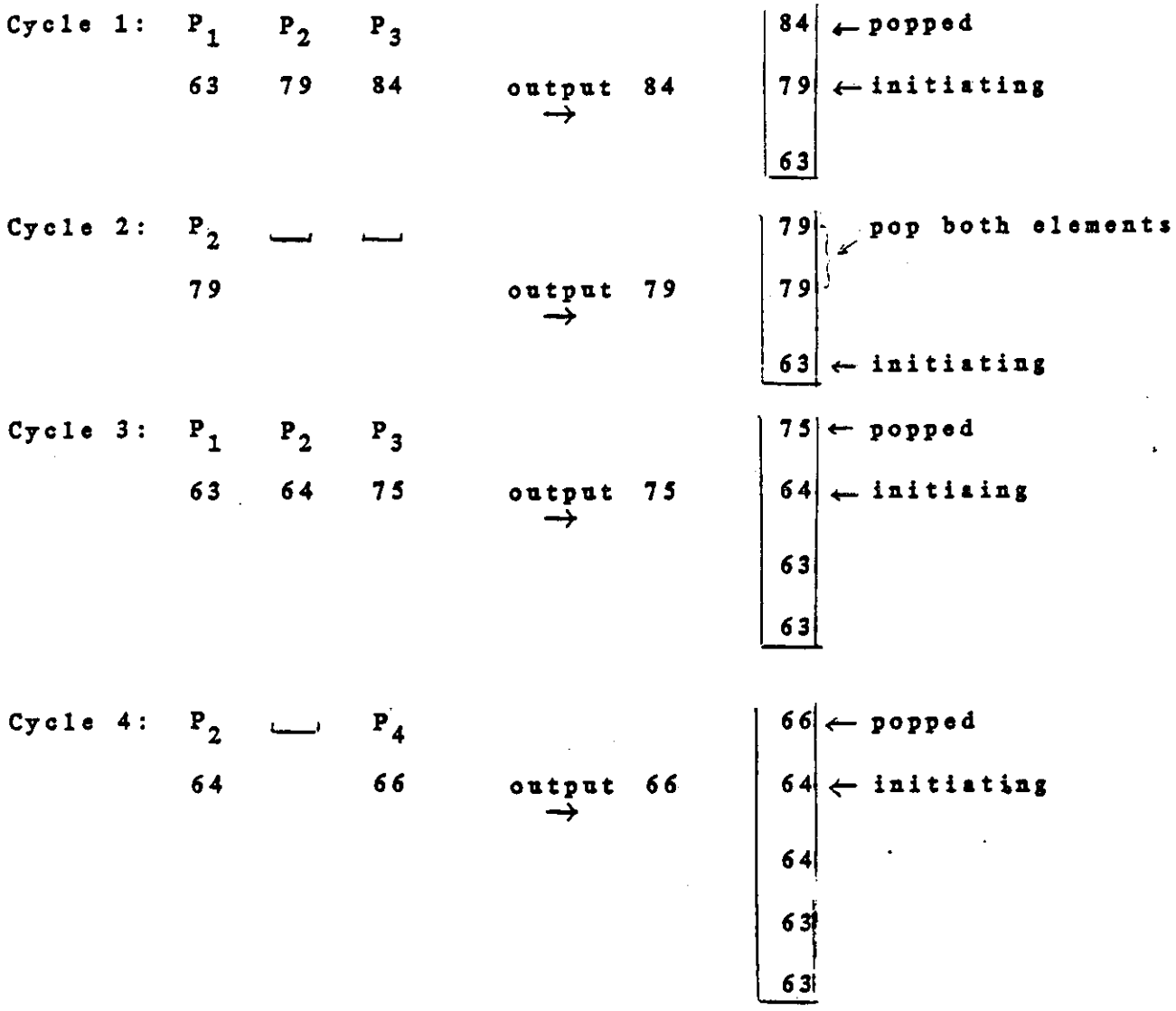
It is most convenient to follow the algorithm's performance by using a stack. We show it by the next example.

Example: Suppose we have n=12, k=4 and the situation in our 4 processors after phase 1 is as follows:




The performance of the algorithm will be shown by giving the sequence of Processors-elements in each cycle, the element which is determined to be the next in the merged list and the stack by which we view the performance of the algorithm. The sign is used to indicate empty time slots, i.e. a processor which remained silent during its turn to broadcast.

The stack pushes broadcasted elements as it ''hears'' them and pops the Max elements and then initiates by the Top element.


Cycle 1: P1 P2 P3
63 79 84
output 84
84 ← popped
79 ← initiating
63
Cycle 2: P2 ___ ___
79
output 79
79
79
pop both elements
63 ← initiating
Cycle 3: P1 P2 P3
63 64 75
output 75
75 ← popped
64 ← initiaing
63
63
Cycle 4: P2 ___ P4
64 66
output 66
66 ← popped
64 ← initiating
64
63
63

Cycle 5: $P_2$ $P_4$

64 65 output → 65

65 ← popped
64 ← initiating
64
64
63
63

Cycle 6: $P_2$

64 output → 64

64
64
64 popped
64
63 ← initiating
63

Cycle 7: $P_1$

63 output → 63

63
63 ← popped
63

Cycle 8: $P_1$

54 output → 54

54 ← popped

Cycle 9: $P_1$ $P_2$ $P_3$

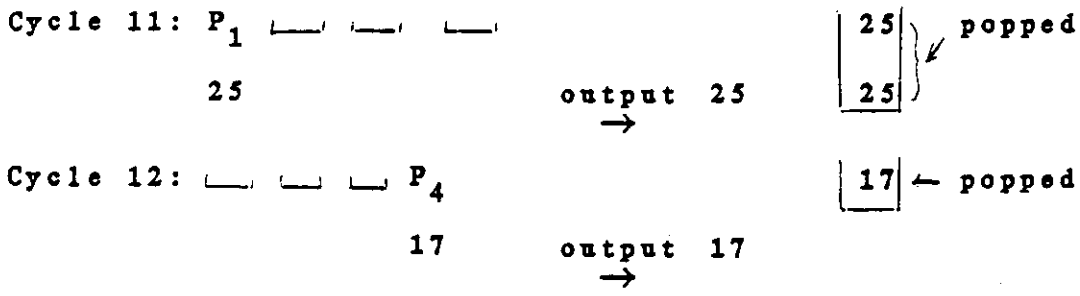
25 28 32 output → 32

32 ← popped
28 ← initiating
25

Cycle 10: $P_2$

28 output → 28

28 popped
28 initiating
25




In the above example we begin with the initial state when phase 2 begins. The first cycle is initiated by $p_1$, and the broadcast turns are in order from left to right. In the first cycle 83 was determined to be the Max element. It is popped from processor P3 the top element of which then becomes 75. The second cycle is initiated by Processor P2 since it broadcasted immediately before P3 in cycle 1 and so on. A processor which has an empty stack remains silent during its turn to broadcast.

## b. Complexity of MERGE-SORT Algorithm

Here we present a worst case analysis when again, we compute the number of comparisons performed by the algorithm. Sorting n/k elements by a processor takes $O(\frac{n}{k}\log\frac{n}{k})$ using either Quicksort or Mergesort - This is the complexity of phase 1. As for phase 2, we next show that the worst case complexity is 2n-1. In order to prove that we consider the following lemmas. Let $\#a_i$ be the number of times element $a_i$ was broadcast.

lemma 1: if $\#a_i > 1$, then $a_i$ initiated at least $\#a_i - 1$ cycles.

This lemma implies that whenever $a_i$ rebroadcasts after the first time, it always initiates a cycle.

lemma 2: each time $a_i$ initiates a cycle, (except for the last cycle which includes only $a_i$) the cycle is terminated by an element never broadcast before.

This lemma implies that for any broadcast of $a_i$ excluding the first and the last we can find a distinct element which broadcast just once. This element is popped immediately after it is broadcast.

Since no two distinct elements initiate the same cycle, we can partition the set $\{a_1, \ldots, a_n\}$ of all our elements to subsets $M = \{S_1, S_2, \ldots, S_t\}$ such that in each subset $S_i$ there is either one element, which broadcast either once or twice or $S_i$ consists of m elements one of them $a_j$ broadcast m+1 times and the rest m-1 elements are those which terminated each initiation of the m-1 cycles by $a_j$.

Conclusion 1:

1) $\forall i,j \quad s_i \sqcap s_j = \phi$

2) if $|S_i| = m$ then the number of times all the elements in $S_i$ broadcasted together denoted by $|S_i|^B \leq 2m$

The conclusion follows immediately from lemma 1, lemma 2 and from the partitioning we created. Let S(n,k) be the number of broadcasts performed by the algorithm. Obviously,

$$S(n,k) \leq \sum_{S_i \in M} |S_i|^B \leq \sum_{S_i \in M} 2|S_i| = 2 \sum_{S_i \in M} |S_i|$$

$$= 2n$$

We conclude with the following theorem.

Theorem 1 $\quad n \leq S(n,k) \leq 2n-1$

Proof:

It is obvious that $n \leq S(n,k)$ since each element has to be broadcast at least once. The second part $S(n,k) \leq 2n-1$ follows quite immediately from all the results given above. The first element which is determined in the first cycle is broadcast just once. We are left with n-1 elements for which we have shown that the upper bound for their total broadcast time is 2(n-1), which yields:

$$S(n,K) \leq 2(n-1)+1 = 2n-1$$

□

We now give proofs for lemmas 1 and 2.

Proof of lemma 1

Assume to the contrary that there are two cycles in both of which $a_i$ is not the initiating element.

$$C_{i_1} \quad a_1 \cdots a_i \cdots a_t$$

$$C_{i_2} \quad a_r \cdots a_i \cdots a_j$$

and assume that cycle $C_{i_1}$ was created before cycle $C_{i_2}$. However,

by the nature of the algorithm after cycle $C_{i_1}$, no element to the left of $a_i$ in cycle $C_{i_1}$ can be broadcast before $a_i$ is popped and merged to the output list. So we get a contradiction.

Proof of lemma 2

Assume $\#a_i > 2$ and we arbitrarily choose the $m^{th}$ broadcast of $a_i$ $1<m<\#a_i$. The cycle which is initiated by the $m^{th}$ broadcast of $a_i$ is denoted by $C_i^m$ and the element to terminate the cycle is denoted by $t_i^m$. So cycle $C_i^m = a_i \cdots a_j \cdots t_i^m$. However, if $t_i^m$ participated previously in a cycle, it should initiate all other cycles according to lemma 1, a contradiction.

Conclusion:

The worst case performance of Algorithm MERGE-SORT presented here is given by

$$\frac{n\log n}{k} - \frac{n\log k}{n} + 2n-1$$

or simply

$$T = (n\log n)/k + o(n)$$

We know that the maximum number of comparisons required for sorting a sequence of n elements on a sequential computer is asymptotically nlogn. Therefore, when k is smaller than log n (in order of magnitude), the asymptotic speed-up ratio of our algorithm over the optimal sequential algorithm is k, which is optimal. In particular, when k=log n, the ratio of this parallel

algorithm to the optimal sequential algorithm is of order log n, the number of processors. On the other hand, when k is greater than log n, the total execution time required is asymptotically linear in n.

NOTE: The MAX algorithm and the MERGE-SORT algorithm presented here could be modified to work asynchronously. This is a result of the fact that the order by which processors broadcast is maintained only for the purpose of preventing collisions; however, for the correctness of the algorithm, the order by which processors broadcast in each cycle is not affecting the results. The only important thing is to be able to determine 1) when a cycle ends and 2) who initiates the next cycle.

References


[1] Kung, H.T. Synchronized and asynchronous parallel algorithms for multiprocessors in Algorithms and Complexity: New Directions and Recent Results, (T. F. Troub, ed.), Academic Press, New York, pp. 153-200, 1976.

[2] Baudet, Gerard and David Stevenson. ''Optimal Sorting Algorithms for Parallel Computers'', IEEE Transactions on Computers, Vol. C-27, No. 1, January 1978.

[3] Deo, Narsengh, C. Y. Pang and R. E. Lord. ''Two Parallel Algorithms for Shortest Path Problems'', proceedings of 1980 international conference on parallel processing.

[4] Kung, H.T. ''The Structure of Parallel Algorithms '' Advances in Computers, Vol. 19.