tree requesting the computation of the minimum.

Operations $XMAX$ and $INSERT$ for this implementation are described below.

$INSERT$: An $INSERT$ operation initiated at node $z$ first computes the address of the leftmost apex $y$ whose partition has at least one empty node (using $Global\ Min$ operation). The element-priority pair is then sent from node $z$ to the node which contains apex $y$. The pair will be inserted into the partition rooted at apex $y$ according to the procedure described for the second implementation.

The problem with the $INSERT$ operation as given above is that an insertion operation may find a partition (reported to have empty positions) full when it tries to insert a pair into that partition, and therefore blocked and unable to proceed. This is caused by allowing several $INSERT$ operations to search the list of apexes concurrently for their point of insertions, which, as a result more than one $INSERT$ operation, may receive the same apex whose corresponding partition has only one empty position as the point of insertion. In this situation, a new $INSERT$ operation may be reissued at the node which blocked the insertion.

$XMAX$: An $XMAX$ operation initiated at node $z$ first determines the apex which contains the element with the highest priority (using $Global\ Min$ operation) and then sends an operation, called $adjust$, to the node containing that apex. Operation $adjust$ first reports the element to node $z$, and then adjust the banyan heap as described in the previous section.

The problem with $XMAX$ operation as described above is that due to concurrent access to the list of apexes by several processors, the same apex may be reported to several $XMAX$ operations as the holder of the element with the highest priority. This may lead to the situation in which elements returned and subsequently deleted by some of the $XMAX$ operations do not have the highest priority in the banyan heap at the time of removal. One solution to this problem is to send all the $XMAX$ operations (issued at different nodes) to node $p_0$ and let node $p_0$ execute them sequentially. This limits the amount of concurrency obtainable in the system. The amount of obtainable concurrency starts to decrease due to this strategy when the number of elements in the banyan heap exceeds $2^{\log M+1} - 1$ for the first time.

**Theorem 5** Operation $XMAX$ requires $O(\log M)$ time to complete.

Proof: Operation $XMAX$ consist of 3 parts:
1. finding the apex containing the element with the highest element.
2. reporting the element to the node initiated the operation.
3. adjusting the banyan heap.

Each of these steps requires $O(\log M)$ time and hence the total time of $O(\log M)$ for $XMAX$.

**Theorem 6** Operation $INSERT$ requires $O(\log M)$ time to complete.

Proof: Similar to theorem 5.

**Theorem 7** An $XMAX$ operation needs no more than $2(M-1)+2\log M$ communications.

**Theorem 8** Third implementation offers $O(M/\log M)$ throughput.

---

Proof: The third implementation requires $O(\log M)$ time, and

In what follows we first define for concurrent data structure

**Definition 12** The timestamp at a node and the timestamp generated that communication.

**Definition 13** Operations said to be a sequence of operations issued between

**Definition 14** A concurrency of any sequence of operations sequentially.

**Theorem 9** The third implementation

Proof: Consider a sequence the fact that these $XMAX$ timestamps, it is quite possi

**Theorem 10** The third in operation is not started until all their $M+1$ accesses to ti

Below we describe a proto
Proof: The third implementation can perform $M$ operations at a time and each operation requires $O(\log M)$ time, and hence $O(M/\log M)$ throughput.

In what follows we first define a few terms and then introduce the concept of consistency for concurrent data structure.

**Definition 12** The timestamp of an operation is the time at which the operation is initiated at a node and the timestamp of a communication is the timestamp of the operation which generated that communication.

**Definition 13** Operations $O_1, O_2, O_3, \ldots, O_q$ with timestamps $t_1, t_2, t_3, \ldots, t_q$ are said to be a sequence of operations if $t_1 < t_2 < \ldots < t_q$ and operations $O_i, 1 < i \leq q$ are the only operations issued between $t_1$ and $t_q$.

**Definition 14** A concurrent data structure is said to be consistent if the concurrent execution of any sequence of operations on the data structure gives the same result as executing the operations sequentially.

**Theorem 9** The third implementation is not consistent.

Proof: Consider a sequence of $XMAX$ operations waiting in node $p_0$ for execution. Due to the fact that these $XMAX$ operations are executed sequentially in increasing order of their timestamps, it is quite possible for an $INSERT$ operation which have a larger timestamp than any of the $XMAX$ operations to be executed before all the $XMAX$ operations are completed.

**Theorem 10** The third implementation is consistent if an access made to an apex by an operation is not started unless all the operations with the lower timestamps have completed all their $M+1$ accesses to the apex.

Below we describe a procedure for making the third implementation consistent.

An operation is first recorded by all the processors in the hypercube. The processor which initiated the operation broadcasts that operation together with its timestamp to all the processors using its fan out tree. The operation and its timestamp is added to the list of incomplete operations maintained by resident executive of each processor. After the operation is added to the list of incomplete operations by a processor, that processor sends a notification signal to the processor that initiated the operation. The notification signals issued by the processors are combined according to the fanout tree rooted at the processor which initiated the operation; no processor sends a notification signal to its parent in the fan out tree unless all its children have noted that operation. After the root processor received all the notification signals, the execution of the operation starts. During the execution of the operation no access is made to an apex unless all the operations with lower timestamps have completed their access to that apex. When an operation completes its last access to the apexes, it asks all the executives to remove that operation from their list of incomplete operations. The removal operation is performed in the same fashion as the operation of recording an operation. After an operation is removed from the list of incomplete operations of an executive, the next pending operation will be allowed to perform its access to the corresponding apex.
7 Conclusion

We have proposed three concurrent data structures for implementing priority queues on a distributed-memory message passing multiprocessor with hypercube topology. These concurrent data structures can be used by any application running on the hypercube without worrying about all the necessary communications and synchronizations. Priority queue operations each require $O(\log M)$ time to complete, but since $M$ operations may be initiated simultaneously at different processors, $O(M/\log M)$ throughput is achievable as compared to $O(1)$ throughput for sequential data structures.

8 References


16. J. D. Ullman, Computation

17. T. A. Standish, Data Structures

18. E. Horowitz and A. Sahni


20. P. L. Lehman and S. B. ACM Transactions on Design Automation of Electronic Systems


23. J. L. Bentley and H. T. International Conference on Computation


26. H. T. Kung and C. E. L. Computer Science

27. A. K. Somani and V. K. 11th Annual International Conference on Computation


29. H. Schmeck and H. Schulz, Transactions on Computers


31. D. Knuth, The Art of Computation

32. E. M. Reingold and W. J.

33. N. J. Nilsson, Problem Solving

34. F. Dehne and N. Santoro International Conference on Computation

35. L. R. Gokce and G. L. Lipic Proceedings of the Fifth

36. A. L. Tharp, File Organization

37. A. V. Aho, J. E. Hopcroft, Algorithms, Addison Wesley,


figure 1
figure 2
figure 3
Repeat
Parallel

Bruno Ce

1 Int. Comp. Sci. Im
2 Ist. di Elaborazione del

Abstract. Given a n solution x can be con provided that the con
in particular, if μ(A)
result, we reduce the a number of steps ind error 2^{-d}, d = O(1),
achieving the same b more simple.

1 Introduction

In the early seventies it its full generality, time paper appearing in 1971
n, A^{-1} could be comput processors (a polynomi narrowed, but did not
algorithm and the trivi sequence of this, many for parallel matrix inve
1 ≤ α < 2. This kind
solving a linear system, acteristic polynomial I
to matrix inversion in 1 exists for one of them,
bound [5,9,10]. Despite
ial, and it is still an
be inverted in O(log n)

* This work has been pa
“Progetto Finalizzato
dedicated”. Part of this
Elaborazione dell’Infor

5 Ceanky’s result concer
soon extended to arbit