tree requesting the computation of the minimum.

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

INSERT: An *INSERT* operation initiated at node x 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 x 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 x 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 x, 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* operation 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 $2^{\log M+2}-1$ 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 $O(logM)$ 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 implement
requires $O(\log M)$ time, and

In what follows we first de
for concurrent data structur

Definition 12 *The timesta
at a node and the timestam
generated that communicati*

Definition 13 *Operations
said to be a sequence of ope
only operations issued betwe*

Definition 14 *A concurre
tion of any sequence of ope
the operations sequentially.*

Theorem 9 *The third imp*

Proof: Consider a sequence
the fact that these *XMAX*
timestamps, it is quite possi
any of the *XMAX* operation

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

Below we describe a proc

An operation is first re
which initiated the operati
the processors using its fa
list of incomplete operatio
operation is added to the l
a notification signal to the
issued by the processors ar
which initiated the operati
out tree unless all its child
all the notification signals,
the operation no access is
have completed their acces
the apexes, it asks all the
operations. The removal
recording an operation. Af
of an executive, the next
corresponding apex.

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 $O_1, O_2, O_3, ...,$ and O_q with timestamps $t_1, t_2, t_3, ...,$ and t_q are said to be a sequence of operations if $t_1 < t_2 < .... < t_q$ 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 apexes.*

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

1. J. Biswas and J. C. Browne, "Simultaneous Update of Priority Structures," Proceedings of International Conference on Parallel Processing, August 1987, pp. 124-131.

2. M. J. Carey and C. D. Thompson, "An efficient Implementation of Search trees on $\lceil \log N+1 \rceil$ processors," IEEE Transactions on Computers, Vol. C-33, No. 11, Nov. 1984, pp. 1038-1041.

3. Y. Saad and M. Schultz, "Topological Properties of Hypercubes," IEEE Transactions of Computers, Vol. 37, No. 7, July 1988, pp. 867-872.

4. M. R. Meybodi, "New Designs for Priority Queue Machine," Proceedings of PARBASE-90: Conference on Databases, Parallel Architectures, and their Applications, Miami Beach, Florida, March 6-9, 1990, pp. 123-128.

5. M. R. Meybodi, "Tree Structured Dictionary Machines for VLSI-A Survey," CS Technical Report, Ohio University, Athens, Ohio, 47501.

6. M. R. Meybodi, "Implementing Priority Queues on Hypercube Machine," Annual Parallel Processing Symposium, Fullerton, Califonia, April 4-6, 1990, pp. 132-157.

7. C. Moler and D. S. Scott, "Communication Utilities For The iPSC," iPSC Tech. Report, August 1986.

8. J. E. Brandenburg and D. S. Scott, "Embeddings of Communication Tree and Grid into Hypercubes," iPSC Technical Report, August 1986.

9. V. N. Rao and V. Kumar, "Concurrent Access of Priority Queue," IEEE Transactions on Computers, Vol. 37, No. 12, Dec. 1988, pp. 1657-1665.

10. W. J. Dally, A VLSI Architecture for Concurrent Data Structures, Kluwer Academic Publishers, 1987.

11. M. J. Quinn, Designing Efficient Algorithms for parallel Computers, McGraw Hill, 1987.

12. C. D. Thompson, "The VLSI Complexity of Sorting," IEEE Transactions on Computers, Vol. C-32, No. 12, Dec. 1983, pp. 373-386.

13. A. R. Omondi and J. D. Brock, "Implementing a Dictionary on Hypercube Machine," Proceedings of International Conference on Parallel Processing, August 1987, pp. 707-709.

14. K. H. Cheng, "Efficient Designs of Priority Queue," Proceedings of International Conference on Parallel Processing, August 1988, pp. 363-366.

15. J. H. Chang, O. H. Ibarra, M. J. Chang, and K. K. Rao, "Systolic Tree Implementation of Data Structures," IEEE Transactions on Computers, Vol. 37, No. 6, June 1988, pp. 727-735.

16. J. D. Ullman, Computati

17. T. A. Standish, Data Str

18. E. Horowitz and A. Sahn

19. H. T. Kung and P. L. L
Transactions on Databas

20. P. L. Lehman and S. B.
ACM Transactions on D

21. C. S. Ellis, "Concurrent
puters, Vol. C-29, No. 9

22. C. S. Ellis, "Concurrent
pp. 63-86.

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

24. C. E. Leiserson, "Systol
University, Pittsburgh, F

25. T. A. Ottmann, A. L. R
IEEE Transaction on Co

26. H. T. Kung and C. E. L
on Sparse Matrix Compu

27. A. K. Somani and V. K.
11th Annual Internation

28. M. J. Atallah and S. R.
transactions on Compute

29. H. Schmeck and H. Schr
Transaction on computer

30. A. L. Fisher, "Dictionar
ternational Symposium c

31. D. Knuth, The Art of Co

32. E. M. Reingold and W. J

33. N. J. Nilsson, Problem S

34. F. Dehne and N. Santoro
International Conference

35. L. R. Goke and G. L. Lip
Proceedings of the First

36. A. L. Tharp, File Organi

37. A. V. Aho, J. E. Hopcrof
rithms, Addison Wesley,

16. J. D. Ullman, Computational Aspect of VLSI, Computer Science Press, 1984.

17. T. A. Standish, Data Structures Techniques, Addison Wesley, 1980.

18. E. Horowitz and A. Sahni, Fundamentals of Data structures, Computer Science Press, 1983.

19. H. T. Kung and P. L. Lehman, "Concurrent Manipulation of Binary Search Trees," ACM Transactions on Database Systems, Vol. 5, No. 3, Sept. 1980, pp. 354-382.

20. P. L. Lehman and S. B. Yao, "Efficient Locking for Concurrent Operations on B-Trees," ACM Transactions on Database Systems, Vol. 6, No. 4, Dec. 1981, pp. 650-670.

21. C. S. Ellis, "Concurrent Search and Insertion in AVL Trees," IEEE Transactions on Computers, Vol. C-29, No. 9 September 1980, pp. 811-817.

22. C. S. Ellis, "Concurrent Search and Insertion in 2-3 Trees," Acta Information, Vol. 14, 1980, pp. 63-86.

23. J. L. Bentley and H. T. Kung, "A Tree Machine for Searching Problems," Proceeding of the International Conference on Parallel Processing, 1979.

24. C. E. Leiserson, "Systolic Priority Queues," Dept. of Computer Science, Carnegie Melon University, Pittsburgh, PA, Report CMU-CS-115,1979.

25. T. A. Ottmann, A. L. Rosenberg, and L. J. Stockmeyer, "A Dictionary Machine for VLSI," IEEE Transaction on Computers, Vol. C-31, No. 9, Sept. 1982, pp. 892-897.

26. H. T. Kung and C. E. Leiserson, "Systolic Arrays (for VLSI), " Proceedings of Symposium on Sparse Matrix Computations and their Applications, Nov. 1978, pp. 256-282.

27. A. K. Somani and V. K. Agarwal, "An Efficient VLSI Dictionary Machine, " Proceedings of 11th Annual International Symposium on Computer Architecture, 1985, pp. 142-150.

28. M. J. Atallah and S. R. Kosaraju, "A Generalized Dictionary Machine for VLSI," IEEE transactions on Computers, Vol. C-34, No. 2, Feb. 1985, pp. 151-155.

29. H. Schmeck and H. Schroder, "Dictionary Machines for Different Models of VLSI," IEEE Transaction on computers, Vol. C- 34, No. 5, May 1985, pp. 472-475.

30. A. L. Fisher, "Dictionary Machines with Small Number of Processors," Proceedings of International Symposium on Computer Architectures, 1984, pp. 151-156.

31. D. Knuth, The Art of Computer Programming, Vol. 3, 1973.

32. E. M. Reingold and W. J. Hansen, Data Structures, Little, Brown and Company, 1983.

33. N. J. Nilsson, Problem Solving Methods in Artificial Intelligence, McGraw Hill, 1971.

34. F. Dehne and N. Santoro, "Optimal VLSI Dictionary Machines on Meshes," Proceedings of International Conference on Parallel Processing, August 1987, pp. 832-840.

35. L. R. Goke and G. L. Lipovski, "Banyan Networks for Partitioning Multiprocessor Systems," Proceedings of the First Annual Symposium on Computer Architecture, 1973, pp. 21-28.

36. A. L. Tharp, File Organization and Processing, John Wiley and Sons, 1988.

37. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison Wesley, 1974.

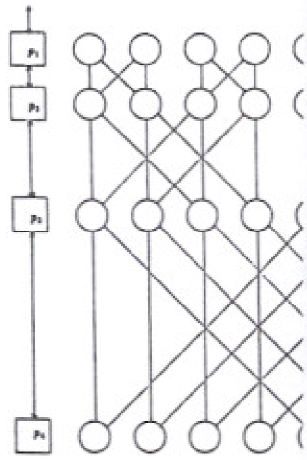
38. R. Nix, "An Evaluation of Pagodas," Tech. Rep. 164, Computer Science Dept. Yale Univ.

39. D. D. Sleator and R. E. Tarjan , "Self Adjusting Heaps," SIAM J. Comput. Vol. 15, No 1, Feb. 1986, pp. 52-59.

40. J. Vuillemin, "A Data Structure for Manipulating Priority Queues," Comm. ACM, Vol. 21, No. 4, 1985, pp. 309-315.

41. M. R. Meybodi, "Binary Search Mesh: A Concurrent Data Structure for Hypercube," Computer Science Technical Report, Ohio University, Athens, Ohio, Jan. 1992.

42. M. R. Meybodi "Banyan Heap Machine," Proceedings of Sixth International Parallel Processing Symposium, University of Southern California, Los Angeles, CA, March 1992.



figure

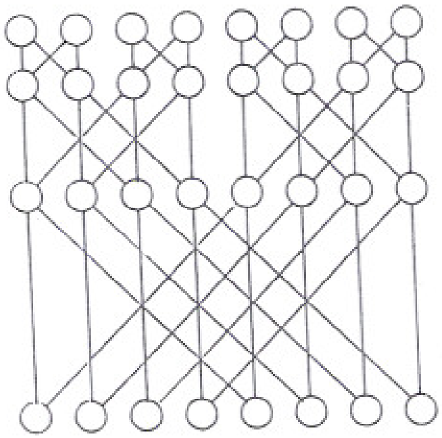


figure 1

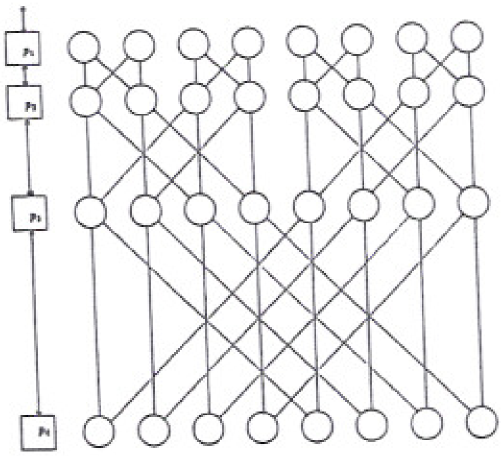


figure 2

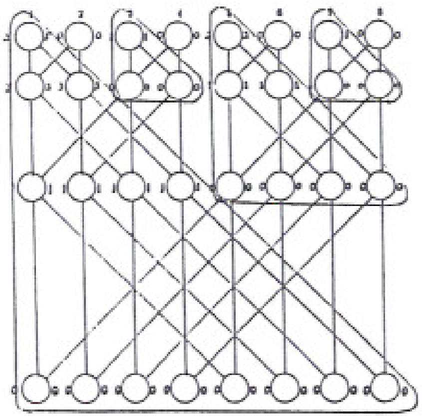


figure 3

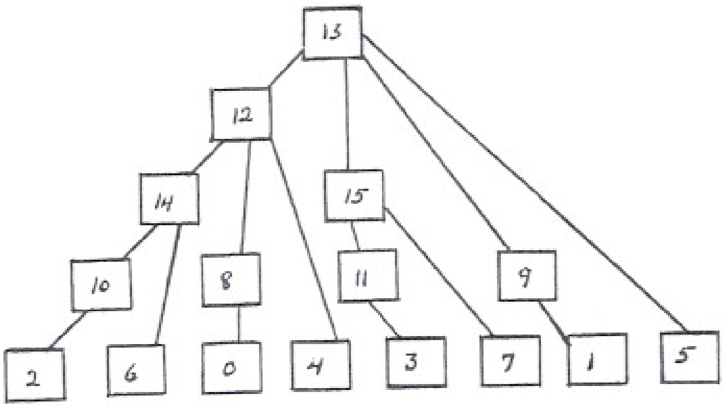


figure 4

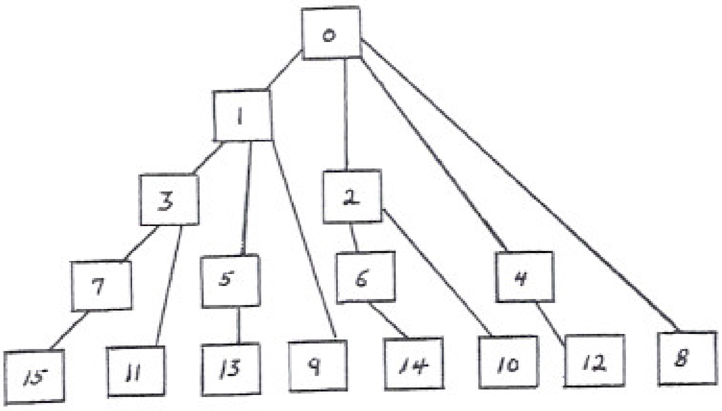


figure 5

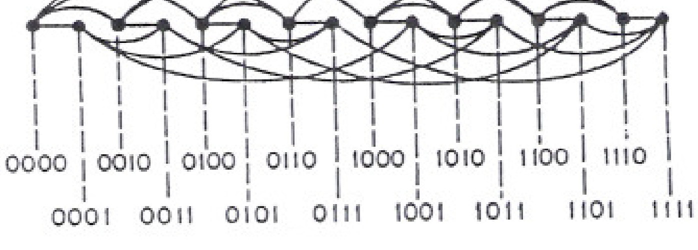


figure 6

Repeat
Parallel

Bruno Cc

[1] Int. Comp. Sci. In:
[2] Ist. di Elaborazione del

Abstract. Given a n
solution **x** can be con
provided that the con
In particular, if $\mu(A)$
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 197
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
$1 \leq \alpha < 2$. This kind
solving a linear system,
acteristic polynomial. I
to matrix inversion in
exists for one of them,
bound [5,9,10]. Despite
tially, and it is still an
be inverted in $O(\log n)$

* This work has been pa
"Progetto Finalizzato
dedicati". Part of this
Elaborazione dell'Info

[3] Csanky's result concer
soon extended to arbi