AN INTERCONNECTION NETWORK SUPPORTING RELATIONAL JOIN OPERATIONS

B. Menezes*, D. Brant**, D. Loewi,
A. G. Dale, R. Jenevein

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-25 June 1987

Abstract

The use of the KYKLOS Interconnection Network as the heart of an I/O Engine is investigated. Its tree architecture combined with excellent leaf node to leaf node traffic propeties provided the motivation to determine its suitability in support of database operations. The network provides the linkage for multiple hosts and I/O nodes with hosts at the roots and I/O nodes at the leaves. From a database point of view, join processing is distributed to all nodes in the network. The KYKLOS-II Network is shown to have significant cost/performance advantages over other tree-based networks.

---

*Department of Electrical and Computer Engineering

**Applied Research Laboratories

1 Introduction

Recent research efforts have focussed on techniques for improving computational parallelism. Issues such as the degree to which a system approaches linear speed-up for a given algorithm on a specific architecture constantly surface. If not careful, we will introduce an imbalance between computational power and secondary memory (external memory) as the number of processing components increase. Put simply, the service rate (of data) of the external system must equal or exceed that of the computational absorption rate (of input data) or the parallel ensemble's performance will not be determined by the processing subsystem. It is important to note that balancing processor and memory rates will be driven by the traffic flow in the external memory sub-system itself. It is to this end that we chose to direct our efforts towards the development of a highly parallel external memory sub-system. In this way, if successful, the external memory system could be designed so that the memory bandwidth matched that of the computational power which in turn would promote a balanced design.

To meet the objectives of developing an external memory system commensurate with the computational power of the next generation of host machines, we have been investigating[1] and refining an architecture initially proposed in [BROW85]. As noted in that paper, the architecture lends itself to *parallel access of databases*, and to *parallel operations on data objects* that are being streamed from secondary storage toward the host.

The gross architecture of the I/O Engine is shown in Fig. 1. The architecture is partitioned into four major levels:

- Host processors, which can be either general purpose or specialized processors
- A set of "Node Mappers" which make an associative translation from requested object names (ex. relation names, attribute names) to leaf (I/O) nodes where the objects are stored by generating a routing tag for the Interconnection Network.
- An Interconnection Network(ICN) which couples host and leaf(I/O) processors, and also interconnects leaf processors. The ICN topology proposed is based on the KYKLOS[MENE85b] multiple-tree topology. The non-leaf nodes in this network also incorporate logic and buffering to support merge operations on data streams.
- I/O nodes, each consisting of a general purpose microprocessor, associative disk cache, a sort engine, and an intelligent controller for associated conventional moving-head disks.

In the next section we consider in detail the motivation for the KYKLOS-II ICN topology and describe a strategy for shortest path routing between the leaf nodes of the ICN. We focus

[1]Work reported in this paper was partially supported under ONR grant N00014-86-K-0499

attention on the ICN since this is the key module of the I/O Engine in which the performance-limiting join operation must be executed.

Section 3 of this paper outlines an algorithm for parallel processing of the relational join operations within the proposed ICN. This algorithm exploits the properties of the ICN to provide

- efficient parallel processing for an operation that is a performance-limiting bottleneck for relational database systems and
- efficient utilization of the bandwidth of the I/O Engine.

Finally, traffic through the network and load balancing is studied and the effect of network topology on system performance is investigated.

2 Tree Topologies in Database Architectures

Tree architectures have been used in a variety of database machines. The Non-Von Database Machine[SHAW79] used a hierarchical associative architecture to execute relational algebra primitives more efficiently than the single level associative processor designs. In this design, the primary associative memory(PAM) was organized as a tree machine with a large number of processors implementable in VLSI.

The binary tree has also been used in such commercially available machines as Teradata Corporation's Database Computer[EHRE84]. Highly Concurrent Tree Machines [BROW78, SONG80] have been used to solve problems in DBMS. For example, two complete binary trees connected in a mirror image fashion have been employed to perform sorting in O(log N) time. One of the trees broadcasts streams of data and instructions and the other tree is used to combine and route outputs (Fig. 2).

Many database specific algorithms executed on tree-based architectures involve some sort of a merge operation on sorted streams of data. This technique has been widely used for performing joins, duplicate elimination, global sorts, intersections, unions, etc. The effect of this technique is to meet the need to compare one leaf node's data with that of many other leaf nodes. The basic problem with these operations is traffic congestion near the root which saturates the bandwidth of links at or near the root. Thus the advantages of logarithmic delay and the recursive, scalable structure of a tree are offset by the traffic bottleneck problem. Solutions to this problem have appeared in the literature [DESP78, GOOD81]. To relieve traffic congestion and provide adequate levels of tolerance to node faults, a multiple-tree architecture, KYKLOS-II, has been proposed (referred to as Scheme 2 in [MENE85b]).

2.1 The KYKLOS ICN

The simplest version of KYKLOS is the simple Double Tree with $N=2^n$ leaf or I/O nodes (n=integer), called KYKLOS-I. Each tree has n levels of non-leaf nodes. Nodes and their descendant links are identified by a level number as shown in Fig.3(a). A modified version of this structure called KYKLOS-II is shown in Fig.3(b). The topology is constructed so that the ordering of level -i nodes, $1 \leq i \leq n$, is a perfect shuffle on $(0,1,...,2^n-1)$. This means that the bottom tree in KYKLOS-II may be redrawn without edge intersections (Fig.3(c)) so that the sequence of nodes at each level is a perfect shuffle on $(0,1, ..., 2^m-1)$, $0 \leq m \leq n$. It must be noted that there are possibilities for more than two trees. In particular, the case of a logN-tree KYKLOS is explored in [MENE87a]. In this paper, however, the discussion is confined to the 2-tree case.

Some key properties of the KYKLOS-II Network are

1. Network Cost (as measured by the number of non-leaf nodes) is a linearly increasing function of the number of leaf nodes. Also the degree (fanout) of each non-leaf node is uniformly 3 or less.

2. The maximum traffic density in KYKLOS-II is $O(N^{1.5})$ [JENE86] as compared with the $O(N^2)$ maximum traffic density in the case of KYKLOS-I or a simple binary tree.

3. Average Communication delay (leaf-to-leaf) is further reduced over that in the simple binary tree[MENE87b].

4. Fault tolerance is improved by virtue of the multiple-tree nature of this structure. Further, connectivity in the case of KYKLOS-II is superior to that of KYKLOS-I [MENE86].

5. Routing is straightforward. Also, there are several Routing Strategies[MENE85b], each of which map to a different set of distance and traffic characteristics.

We conclude this section by outlining a strategy to route a message between leaf nodes. This will be made use of in describing the Join Algorithm of Section 3.

2.2 P-II Routing Strategy

We have previously reported H-II and M-II[JENE86] routing for KYKLOS-II networks. In [MENE85a], the P-II routing strategy using the shortest path between leaf nodes was developed. This will be used to support join operations on KYKLOS-II.

Consider routing a message from node 1 to node 8 in a KYKLOS-II ICN with 16 leaf nodes (Fig. 4). If the top tree alone were to be used, a path using 8 link traversals would be necessary. If the bottom tree alone were employed, the path from 1 to 8 would have to include the root of the

bottom tree with a pathlength of 8. However both trees may be employed with a 50% reduction in pathlength by crossing over at node 8.

The ability of passing between trees, termed passthrough here, often (though not always) results in a reduced pathlength (hence reduced communication delay). Further, it has been shown that the shortest path between a given pair of leaf nodes need never involve more than one passthrough if any at all[MENE87b]. The routing strategy which makes this possible is termed P-II. The "P" here implies that there is no restriction on passthrough, the "II" is because the strategy prescribed is for KYKLOS-II.

2.2.1 Routing Algorithm

Let S and D be the source and destination addresses (n-bit strings) of the message.
Let $X = S \oplus D$.
Let t and b be the number of consecutive zeros begining at the MSB and LSB end of X, respectively.
Let p be the maximum number of consecutive zeros between the first nonzero bit on the left and the first nonzero bit on the right.
Let z_1 be the bit position of the first bit to the right of this string of zeros and z_2 be the bit position of the first bit to the left of this string of zeros.
Let lev_num be a variable which may be initialized or modified by a leaf node only.
Table 1 displays each of these values for two different source-destination pairs.

S	D	X	t	b	p	z_1	z_2
0001	1000	1001	0	0	2	0	3
1110	1010	0100	1	2	0	-	-

Table 1: Display of variables for Routing

Note that in the second example of Table 1, there is no cluster of zeros between the first nonzero bit on the left and the first nonzero bit on the right of X. Hence p=0 and z_1 and z_2 are not defined in this case.

The following are the steps involved in the P-II Routing Strategy.

At leaf node initiating Message
if $t \geq p$ and $b \geq p$,
{
  if t > b
    {
      S sets lev_num =n-t,
      Message is shipped out through upper tree link
      incident on S
    }

```
else if b > t
        {
          S sets lev_num = n-b,
          Message is shipped out through lower tree link
          incident on S
        }
 else
        {
          S sets lev_num = n-t,
          Message is shipped out through either upper
          or lower tree link
        }
}
else
{
  S sets lev_num = z1+1,
  upper tree link incident S is used for transmission.
}
```

At leaf node receiving a Message
If the destination tag (D) is equal to its own ID, it keeps the message, otherwise it does lev_num = n-z_2 and forwards the message via the lower tree link incident on it.

At a non-leaf node if message is rootward bound
Let L be the level number of the non-leaf node. Then if L ≠ lev_num, message is forwarded rootward (toward parent) else message is reflected toward other child.

At a non-leaf node if message is leafward bound
If L > 0, it forwards message via left child if bit L-1 of destination address=0, else message is forwarded to right child
else
if L < 0, it forwards message via left child if bit n+L of D = 0, else message is forwarded via right child.

Using the strategy just described, a message from node 1 to 9 would use the top tree first to reach node 8, whence the bottom tree would be employed to reach its final destination (dark lines in Fig. 4). The other example of Table 1 (Node 14 to 10) does not need passthrough. In fact, it could use 4 link traversals in the bottom tree to reach its destination.

2.2.2 Multiple Shortest Paths

Note that there are possibilities for multiple shortest paths. For example, the shortest path for a message from node 1 to 8 may involve the bottom tree first followed by the top tree as shown in the broken lines in Fig. 4. In general, if the shortest path between 2 leaf nodes involves passthrough there will be at least two distinct shortest paths [MENE87b].

In addition, there may be two shortest paths, one involving say, the top tree only and another involving passthrough (i.e. t=p). In such an event, the P-II Routing Strategy would use the top tree exclusively thus foreclosing the passthrough option. The reason for this is the philosophy that routing was primarily a function of the non-leaf nodes and consequently should involve as little intervention of the leaf nodes as possible.

We next describe a join algorithm that exploits the interconnection structure of KYKLOS-II and uses the P-II Routing Algorithm. Before that we briefly describe the data model employed.

3 Distributed Join Processing in KYKLOS

Although the goal of the I/O Engine is to develop a general purpose solution to the I/O bottleneck problem, the KYKLOS-II architecture lends itself quite naturally to relational database processing. Thus the parallelism and concomitant speedup are due to

- parallel access of data at the I/O node sites and
- performing parallel database operations over the entire network (non-leaf nodes).

The following sections illustrate this point by describing how an efficient join operation on a distributed database can be performed. We note here that other join algorithms can be implemented, and we are currently studying their computational and traffic generating characteristics in a variation of the KYKLOS-II architecture. Analytical results for network traffic loads for algorithms in addition to the one described in this paper are given in [MENE87a].

3.1 Data Storage

To produce a machine capable of very high speed parallel access to data, the distribution of that data over the set of leaf nodes is crucial. To make effective use of the architecture, the data should be distributed to the leaf nodes, such that, given a sufficiently large quantity of data to be processed, the entire collection of network resources can be applied to that processing. The following data storage description is based on the relational model. The principle motivating factor in the design has been to provide the highest possible degree of useful parallelism in processing relational algebraic operations. Accordingly, each relation is partitioned and distributed among the leaf nodes of the architecture. A relation, R, may be viewed as a table with rows as tuples and columns as attributes.There are two apparent methods for partitioning that relation. A *partitioning by rows* produces horizontal fragments r_i such that R is the union of all r_i

i.e. $R = \cup_i r_i$.
A *partitioning by columns* produces vertical fragments ρ_j such that R is equal to the natural join of all ρ_j
i.e. $R = \bowtie_i \rho_i$.
These vertical fragments may then be subject to horizontal fragmentation.

The model chosen for the I/O Engine makes use of both of these types of partitioning. The horizontal fragmentation, or *tuple-based schema* (TBS), is the primary storage model. Each fragment of the TBS is mapped to a particular leaf node by the Node Mapper. The size of the partitions will determine the number of fragments. If the fragments are too large, a loss of parallelism will result. If they are too small, inefficiency may result due to the increased system overhead in managing the fragments. Current analytical studies suggest that the appropriate partition size is several disk tracks (i.e. for current disks, 64K-1024K bytes). The vertical fragmentation, or *attribute-based schema* (ABS), is also implemented to provide both, fault tolerance and improved performance. The ABS is a vertical fragmentation followed by a horizontal fragmentation i.e. each of the vertical fragments is then partitioned horizontally and distributed to the leaf nodes in a manner similar to the TBS. Further details of these schemes are included in the appendix.

3.2 The Processing Model for Database Operations in KYKLOS

It has been recognized that on-the-fly and pipelined data stream processing of data are important factors in improving the speed of many relational operations [GARD83, TANA84, SHIB84]. On-the-fly processing refers to the technique of performing relational operations on data as it is being moved from one network node to another. One common operation performed in this manner is to select the appropriate tuples by use of filters. Pipelined-data-stream processing is based on applying the concept of pipelined parallel processing to streams of data. This technique can be used as data is made to traverse several levels of the architecture, with each node along a particular path performing a portion of the required processing. Since many data streams can be in transit at any one time, parallel pipelined-data-stream processing is possible By endowing non-leaf nodes with the required functionality, eg. comparison, concatenation, and buffering, these concepts can be extended to all levels of the KYKLOS architecture. The goal then, is to have a continuous flow of relevant data through the network. The output of this data flow will be either to the host through the roots, or distributed to the leaf nodes for the next operation. We next look at how relational join operations may be distributed over the KYKLOS ICN and its implications to performance.

3.3 Semi-Join Algorithm

This discussion assumes that a natural join is to be computed on relations R and S, each of which is evenly distributed (horizontally) across the N leaf nodes using the TBS. The semi-join algorithm discussed below is only one of several parallel join algorithms that can be implemented on the architecture. It is used here as an example of how the architecture can

support operations that require operands from different I/O nodes.

Let r_i and s_i denote the fragments of relations R and S at node i.
Let C denote the set of common attributes between R and S i.e. $C = R \cap S$. The semi-join algorithm may be thought of as a two-step process

- Phase 1: Each leaf node, i, broadcasts its lists of common attribute values $c_{ri} = \pi_C r_i$ and $c_{si} = \pi_C s_i$ to every other node. On receipt of c_{rj} and c_{sj}, $j \neq i$, each leaf node, i, computes the semi-joins
  $r_{ij}' = r_i \ltimes c_{sj} = r_i \ltimes s_j$
  $s_{ij}' = s_i \ltimes c_{rj} = s_i \ltimes r_j$
  Also it computes the join between its local fragments i.e.
  $r_i \bowtie s_i$
- Phase 2: Each pair of leaf nodes, i,j, $i \neq j$ ships the results of the above semi-joins to a rendezvous determined by mutual consent, at which point a partial join
  $r_{ij}' \bowtie s_{ji}' \cup r_{ji}' \bowtie s_{ij}'$ is performed.

The first phase of the algorithm is particularly straightforward: The two trees in KYKLOS may be used to broadcast the c_{si}'s and c_{ri}'s, $0 \leq i < N$. These lists are used to compute that subset r_{ij}' of r_i which will participate in the partial join between the fragments of i and j. It is the second phase of the algorithm that uses the interconnection structure of KYKLOS-II to achieve a significant improvement in load balancing and network traffic over a single binary tree or KYKLOS-I. This is discussed in the next two subsections.

3.4 The Mid-point Strategy for performing Partial Joins

Consider a 64-node KYKLOS-II (Fig. 5) and consider the partial join between the semi-join fragments of nodes 11(001011) and 46(101110). The shortest paths between these two nodes as prescribed by the P-II Routing Strategy of Section 2 is sketched in Fig. 5. To minimize the traffic, it makes sense to select a rendezvous for the partial join between these node fragments *somewhere* on the shortest path. If the results of the join were to be sent to the host (located at the root), the partial join should be done at the node closest to the root as shown in Fig. 5. However, it is often the case that the results of the partial join are an intermediate result which needs to be returned to the leaf nodes as input for the next phase of a computation. As such, a reasonable compromise between these conflicting requirements would be to perform the join at the mid-point of the path connecting them.

For the example of Fig. 5, the point on the path between leaf nodes 11 and 46 that is closest to the (top) root is a level 3 node. However the strategy being considered would select the mid-point for performing the partial join. Thus a level 2 node is selected which is closer to the leaf nodes. Intutively, an added advantage of performing the partial join at the mid-point and hence at sites closer to the leaves is that as the level number in the top tree decreases, the number of

nodes available for processing the partial joins increases. As a result, the workload gets more evenly distributed.

3.5 Case Study and Analysis

For the purpose of analyzing and comparing traffic using the Mid-point Strategy, a case study using KYKLOS-I and KYKLOS-II is presented, with each network composed of two binary trees.

3.5.1 Distribution of Partial Joins

In the KYKLOS-I case, the Mid-point Strategy is equivalent to performing the partial join at the root of the smallest subtree containing both i and j, an approach commonly employed for a single binary tree. Of course, half of the partial joins can be performed in each tree of KYKLOS-I. Fig. 6 is an example of this approach with N=8. The notation i,j at the nodes indicates the joining of the corresponding fragments of the relations at nodes i and j. Note that there are a total of 8 pairs of leaf nodes whose fragments join at a root which is the busiest node in the network.

Joining the partial fragments at the mid-point of the path using P-II routing in KYKLOS-II produces the partial joins shown in Fig. 7. Employing the same resources but a different interconnection strategy, the KYKLOS-II Network reduces the maximal load to 3.

Table 2 shows the maximum workload i.e. the number of partial joins performed at the busiest site. (Recall that for each pair of nodes i,j, $i \neq j$, the Semi-join Algorithm requires 2 partial joins to be computed). For KYKLOS-I, it is easy to see that the roots are the sites for the partial joins of every pair of nodes separated by a distance of 2logN. Since there are $N^2/4$ such pairs of nodes and 2 partial joins per pair, there are $N^2/2$ partial joins to be computed at the roots or $N^2/4$ partial joins per root. This means that over half of the total number of partial joins are done at the root nodes. By comparison, the maximum workload for KYKLOS-II is at least an order of magnitude less for N>32. Also, the workload in KYKLOS-II is spread out more evenly over the whole network.

3.5.2 Response Time

Another benefit from the KYKLOS-II interconnection strategy is the reduction in the average distance that the semi-join outputs must travel, thereby improving response time. For instance, for N=8, the average distance is 1.9 links traversed as compared with 2.4 in the case of KYKLOS-I, an improvement of about 20%.

3.5.3 Network Traffic

Three categories of communication requirements may be identified depending on the task at hand. The tasks requiring communication in the semi-join algorithm are

- Global broadcast of attribute values to all other leaf nodes
- Broadcast of semi-join outputs to the predetermined site for computing the partial

N	KYKLOS-I	KYKLOS-II
8	16	6
16	64	12
32	256	28
64	1024	88
128	4096	288
256	16,384	928
512	65,536	3008
1024	262,144	9728

Table 2: Maximum Number of partial joins at a node site

joins and

- Transfer of semi-join output values to the host or to the leaf nodes.

The global broadcast is well suited to a tree-structured network. Both KYKLOS-I and KYKLOS-II will exhibit the same performance characteristics for this operation.

The second task necessiates transmission of tuples along the shortest path between every pair of leaf nodes. It has been shown [MENE87b] that the maximum traffic density for this task is $O(N^{1.5})$ if the H-II Routing Strategy is used. Further, the P-II Routing Strategy has a lower asymptotic bound on maximum traffic density of $O(N^{1+log\phi}) \sim N^{1.7}$ where $\phi = (1+\sqrt{5})/2$. However, upto N=16,000, the constants in the expressions for traffic density are such that P-II actually outperforms H-II Routing. By comparison, the maximum traffic in KYKLOS-I is $O(N^2)$.

Finally, the third task could involve transmission of results to the leaf nodes. Table 3 shows the maximum traffic densities for both, KYKLOS-I and KYKLOS-II as a function of N. Note that as in the case of the distribution of partial joins, the maximum traffic in KYKLOS-II is only a tiny fraction of that in KYKLOS-I. This shows that congestion is a far more serious problem with KYKLOS-I. Note that the improvement in traffic characteristics in KYKLOS-II has been brought about by no addition of hardware resources, only by altering the interconnection strategy.

If the result relation must be streamed to the host, regardless of which KYKLOS topology is used, the link from the root to the host is a potential bottleneck. However, if we assume a balanced design, such that the bandwidth of the maximally congested links are at least that of the host's input channel, then the network can deliver data to the host as fast as the host can accept it. This effectively moves any bottleneck from the network to the host.

N	KYKLOS-I	KYKLOS-II
8	8	4
16	32	8
32	128	22
64	512	60
128	2048	176
256	8192	528
512	32,768	1696
1024	131,072	5664

Table 3: Maximum Link Traffic of partial joins at a node site

4 Conclusion

The KYKLOS-II network has been shown to be superior to the standard binary tree topology or KYKLOS-I for computing natural joins in a parallel I/O Engine. By increasing the utilization of the same set of resources, KYKLOS-II enables a significant improvement in the cost/performance ratio over the often used KYKLOS-I type interconnection schemes. Moreover, the routing in KYKLOS-II is straightforward and generalized for all N. If even better performance is demanded of the system, the KYKLOS-II scheme can be extended to incorporate $\log_2 N$ trees for $N \geq 8$. The resulting topology yields a hypercube between levels 1 and -1 of the trees. This HyperKYKLOS offers interesting possibilities for performance-driven applications and is the subject of a recent Technical Report [MENE87a].

APPENDIX

For various reasons associated with integrity constraints and the ABS, each conceptual relation is augmented with a system assigned attribute called a tuple identifier (TID). Each TID value also identifies the relation to which a tuple belongs. The TBS is defined as follows:

1.Given a conceptual relation R composed of attributes $A_1 ... A_n$.
2.Augment R with attribute A_0, the TID.
3.Horizontally fragment the resulting augmented relation, $R(A_0,A_1,...A_n)$ across the N leaf nodes.

The ABS is defined as follows:

1.Let $R(A_0,A_1,...A_n)$ be an augmented relation with A_0 being the TID.
2.Vertically fragment R such that the vertical partitions are of the form $R_1(A_0,A_1)$, $R_2(A_0,A_2),...R_n(A_0, A_n)$.
3.Horizontally fragment each $R_i(A_0,A_i)$ across the N leaf nodes.

The data distribution defined above may be static or dynamic. If application specific dynamic load balancing is desired, then the sizes and contents of the fragments can be adjusted as needed. This is facilitated by dividing the fragments into smaller logical buckets using various dynamic hashing techniques.

REFERENCES

[BROW78] Browning, S.,"Hierarchically Organized Machines" *In C.A. Mead and L.A.Conway, eds., Introduction to VLSI Systems*, Reading, Mass.: Addison-Wesley, 1978.

[BROW85] Browne, J., A.Dale, C.Leung and R.Jenevein,"A Parallel Multi-Stage I/O Architecture with a Self-Managing Disk Cache for Database Management Applications", *Proceedings of the Fourth International Workshop on Database Machines*, March 1985, pp. 330-345.

[DESP78] Despain, A.M. and D.A.Patterson,"X-Tree: A Tree Structured Multi-processor Computer Architecture", *Proceedings of the Fifth International Symposium on Computer Architecture*, pp. 144-151, 1978.

[EHRE84] Ehrensberger, M.J., "The DBC/102 Data Base Computer's System-Architecture, Components, and Performance" *Paper presented at the Minnowbrook Workshop on Database Machines*, 1984.

[GARD83] Gardarin, G., et al., "SABRE: A Relational Database System for a Multimicroprocessor Machine," *Advanced Database Machine Architecture*, D. Hsiao (ed), Prentice-Hall, 1983.

[GOOD81] J.R.Goodman and C.H.Sequin,"Hypertree: A Multiprocessor Interconnection Topology", *IEEE Transactions on Computers*, vol. C-30, pp 923-933, Dec 1981.

[JENE86] Jenevein, R. and B.Menezes, "KYKLOS: Low Tide High Flow" *Proceedings of the Sixth International Conference on Distributed Computing*, pp. 8-15, May 1986.

[MENE85a] Menezes, B., "The KYKLOS Interconnection Network" *PhD. Dissertation Proposal*, Dept. of Electrical and Computer Eng., Univ. of Texas at Austin, July 1985.

[MENE85b] Menezes, B. and R.Jenevein, "KYKLOS: A Linear growth Fault-tolerant Interconnection Network" *Proceedings of the International Conference on Parallel Processing*, pp. 498-502, August 1985.

[MENE86] Menezes, B., R.Jenevein and M.Malek, "Reliability Analysis of the KYKLOS Interconnection Network" *Proceedings of the Sixth International Conference on Distributed Computing*, pp. 46-51, May 1986.

[MENE87a] Menezes, B., K.Thadani, A.Dale and R.Jenevein, "Design of a HyperKYKLOS-based Multiprocessor Architecture for High-Performance Join Operations", *Dept. of Computer Sciences, Technical Report TR-87-18*, the University of Texas at Austin, Austin, Tx., May 1987.

[MENE87b] Menezes, B., "Interconnection Strategies, Properties and Applications of the KYKLOS Interconnection Network" *Dissertation ,in preparation*, Dept. of Electrical and

Computer Eng., University of Texas at Austin, 1987.

[SHAW79] Shaw, D.E., "A Hierarchical Associative Architecture for the Parallel Evaluation of Relational Algebraic Database Primitives" *Dept. of Computer Science, Technical Report STAN-CS-79-778*, Stanford University, Stanford, Ca., 1979.

[SHIB84] Shibayama, S., et al., "A Relational Database Machine with large Semiconductor Disk and Hardware Relational Algebra Processor," *New Generation Computing*, No. 2, 1984, pp. 131-155.

[SONG80] Song, S.W., "On a High-Performance VLSI Solution to Database Problems", *Ph.D. Thesis*, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, Pa., 1980.

[TANA84] Tanaka, Y., "MPDC: Massive Parallel Architecture for Very Large Databases," *Proceedings of the International Conference on Fifth Generation Computer Systems*, pp. 113-137, 1984.

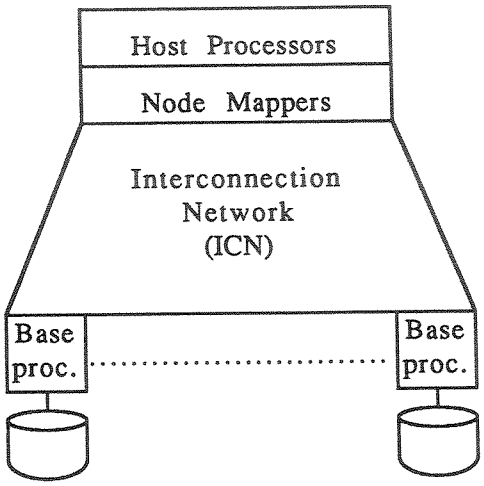

Figure 1. System Overview

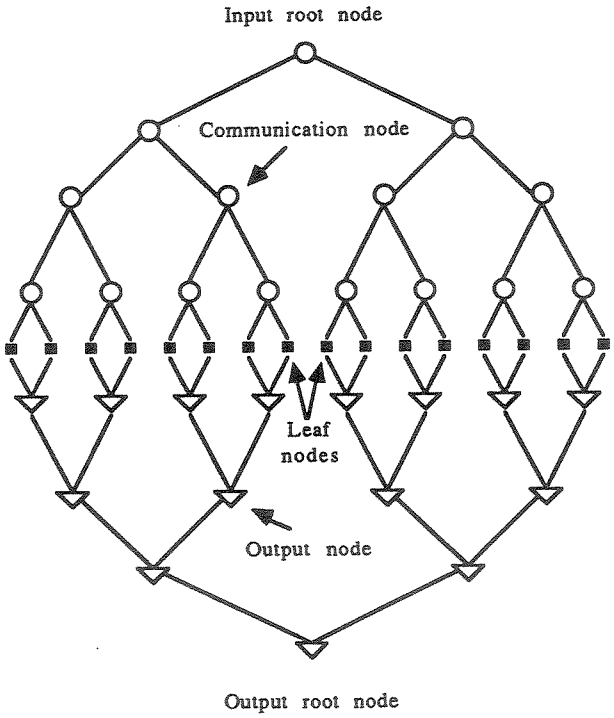

Figure 2. Double binary tree for broadcast/merge

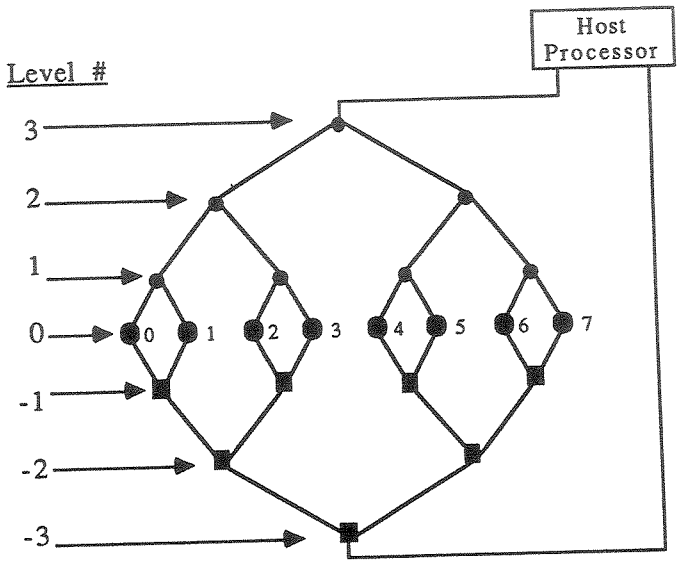

Figure 3a. KYKLOS-I

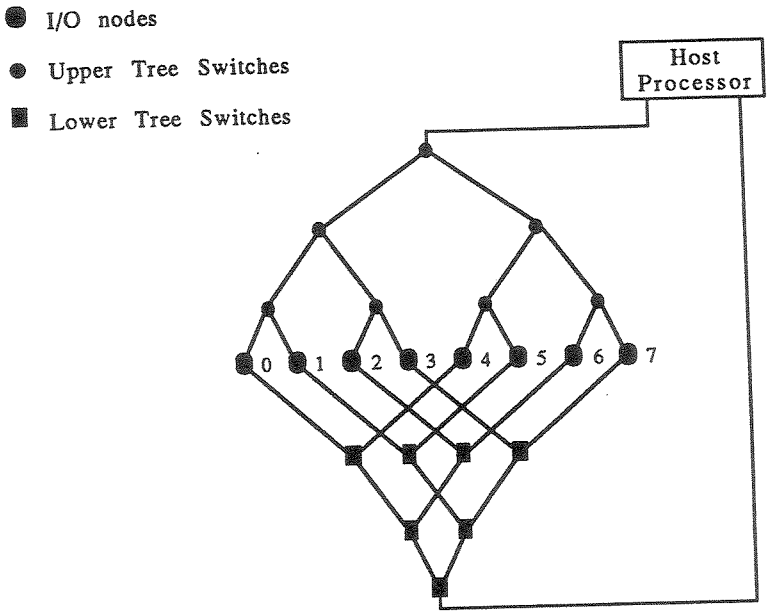

Figure 3b. KYKLOS-II

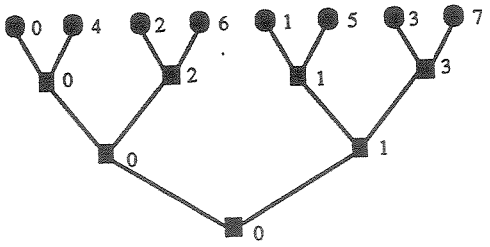

Figure 3c. Redrawn bottom tree in KYKLOS-II

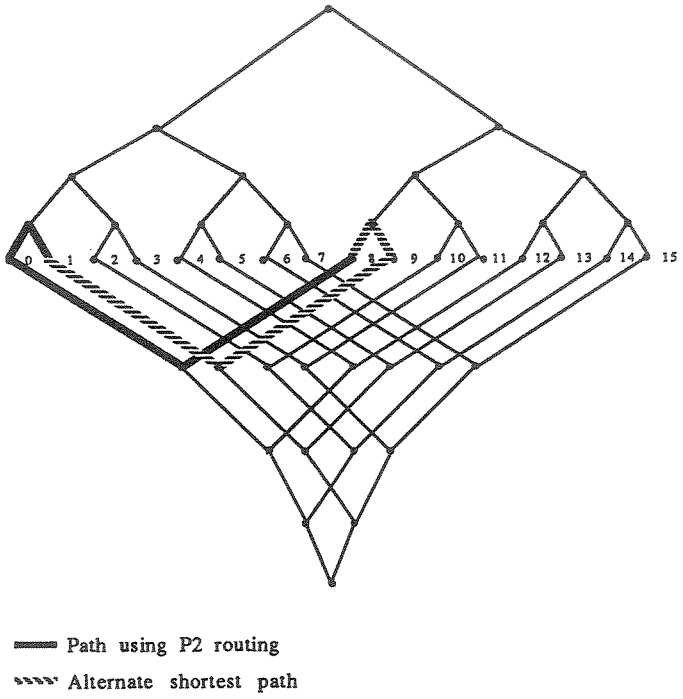

Figure 4. Illustrating P-II Routing in KYKLOS-II (Source = 1, Destination = 8)

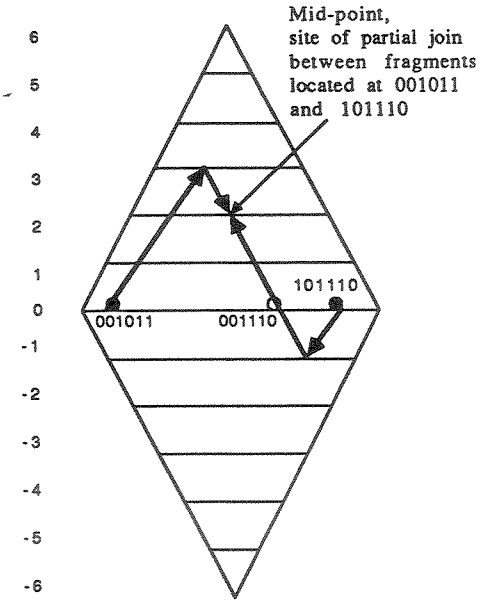

Figure 5. Illustrating Mid-point Strategy in KYKLOS-II with 64 leaf nodes

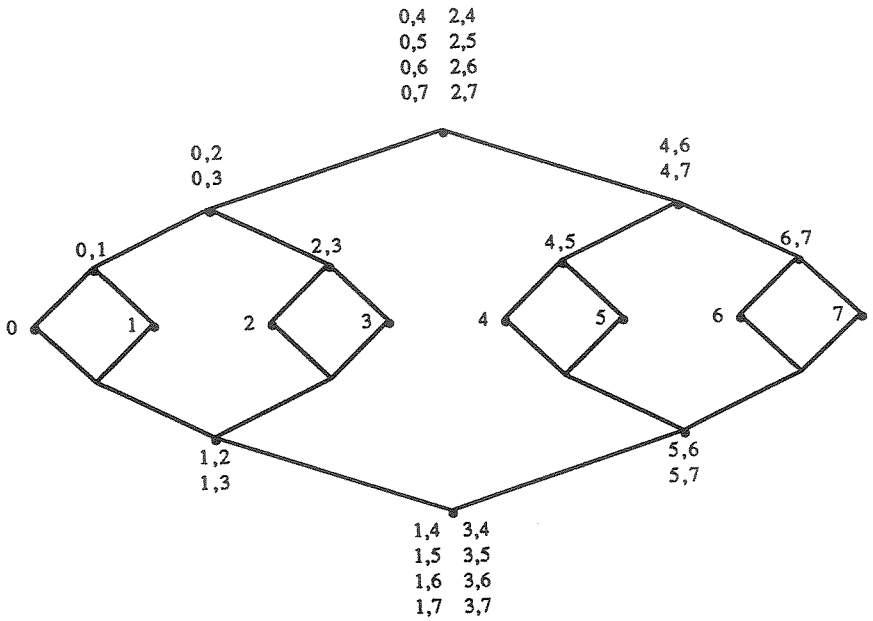

Figure 6. Distribution of partial joins in KYKLOS-I (N = 8)

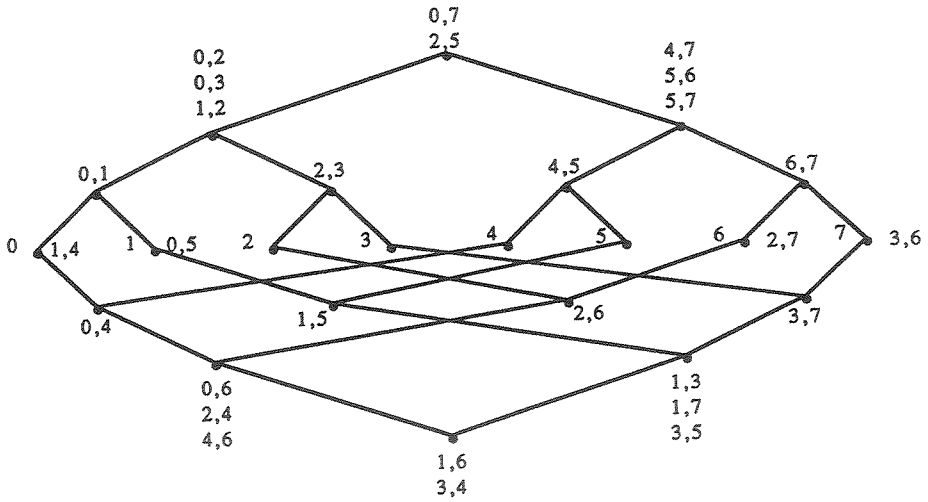

Figure 7. Distribution of partial joins in KYKLOS-II (N = 8)