Design of a HyperKYKLOS-Based Multiprocessor Architecture for High-Performance Join Operations

B. L. Menezes,* K. Thadani,
A. G. Dale, R. Jenevein

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

TR-87-18 May 1987

Abstract

The traffic characteristics of various distributed join algorithms on the Hypercube are analyzed. It is shown that, regardless of which join strategy is employed, the network bandwidth requirements of the computation and collection phases are radically different. This imbalance prevents these two phases from being pipelined (overlapped). To alleviate this problem, the HyperKYKLOS Network is proposed. The topology of this network is defined and a brief description of the I/O nodes presently under construction is included.

*Department of Electrical and Computer Engineering, University of Texas at Austin.

1 Introduction

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 objects (eg. relation names, attribute name-value pairs and tuples) to base (I/O) nodes where the objects are stored by generating a routing tag for the Interconnection Network.
- An Interconnection Network which couples host and base(I/O) processors, and also interconnects base processors. The ICN topology proposed is based on the KYKLOS[MENE85a] multiple-tree topology. The switch 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 associated conventional moving-head disks.

The ICN topology, as discussed in Section 3 of this paper, allows the I/O nodes to be interconnected as a Hypercube, with tree connections from this level of the system to the host level.

Two features of the Hypercube topology are attractive in considering parallel database processing such as the join operation on relations that are partitioned and distributed over the I/O nodes:

1. *Topological Properties:* The average and worst case distances in the Hypercube are bounded by $n=\log_2 N$ in a cube with N nodes[2]. Also, the maximum traffic carried by a link under the assumption of uniform message distribution is O(N).

---

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

[2]n and logN are used interchangeably

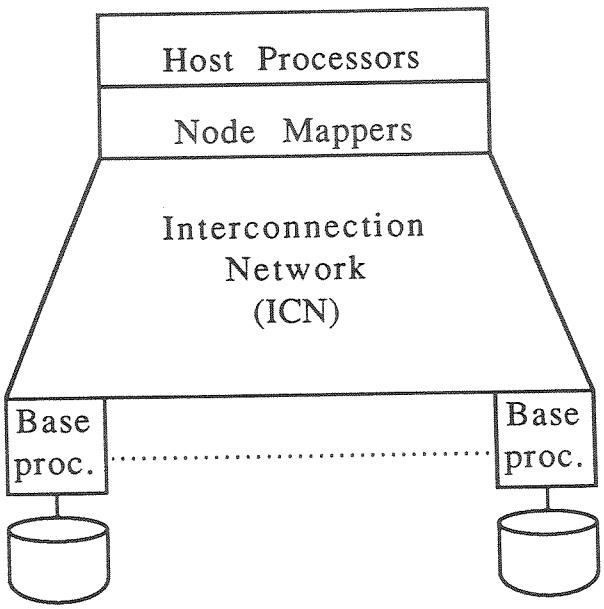


Figure 1. System Overview

2. *Degrees of Freedom:* The fanout of the Hypercube is uniformly n i.e. any given node has n nearest neighbors, one per dimension. This facilitates broadcast of attribute values or relation fragments in O(log N) time by using one dimension per transmission cycle.

Although a Hypercube lends itself to efficient parallel computation of the basic relational operations (selection, projection, join, set operations), it presents major potential problems during the collection phase of database transactions when the partial results must be collected at a given node.

Section 2 of this paper analyzes the traffic properties of various join methods in a Hypercube and compares traffic loads during the compute phase of a database transaction with traffic loads generated during the collection phase. This analysis shows the existence of a serious imbalance in traffic loads between the two phases, and motivates our proposal for the HyperKYKLOS topology discussed in Section 3.

We are pursuing further studies and development of a HyperKYKLOS organized database engine. In Section 4 of this paper, we describe the architecture of a prototype I/O node which is now under development.

2 Traffic Properties of Join and Merge Operations in a Hypercube

2.1 Preliminaries

Every node in the N-cube consists of a processor, main memory and secondary storage; it has n incoming channels and n outgoing channels. It is assumed that one or more host machines is interfaced to a Hypercube node. The relational database is divided into disjoint horizontal fragments and is distributed uniformly among the N processors.

Formally, the database D_x may be viewed as a set of relations, $\{R_1, R_2, \ldots, R_x\}$. Each relation R_i is partitioned int N disjoint horizontal fragments which are stored one per node. We denote by r_{ij}, the j^{th} fragment of R_i stored at node j satisfying

$R_i = \cup_{j=0,N-1} r_{ij}$, i=1, ..., x and

$r_{ij} \cap r_{ik} = 0$, $j \neq k$.

Evaluation of a database query proceeds in three major phases:

- *Phase 1* Transmission of the optimized query to all the processors. This is followed by access to disk or cache to obtain the desired relation(s).
- *Phase 2* (Computation Phase) Evaluation of partial results at each processor.
- *Phase 3* (Collection Phase) Transmission of the results to the host.

The transmission involved in Phase 1 is negligible and hence will not be factored into the traffic computation. Since most queries require the results to be sorted, Phase 3 requires a global sort of the partial results.

Any query can be decomposed into a sequence of selections, projections and joins. Of these, the join is the only operation that requires transmission between processors and hence will be considered in some detail.

We next discuss the details of the implementation of widely used join algorithms [VALD84] on the Hypercube. Under consideration will be the join between R_m and R_n where $|R_m| \leq |R_n|$ and the joining attribute is y. We use |y| to denote the size of the join attribute. Also t_m and t_n will be used to denote the width of tuples of relations R_m and R_n. Finally, by broadcast of a data item x is meant the transmission of x from a node i to every other node in the network. An N-broadcast of a set of data items $\{X_0, X_1, \ldots, X_{N-1}\}$ is the transmission of X_j to network node j, $0 \leq j < N$. Note that a broadcast is a special case of an N-broadcast in which $x=X_i$, $0 \leq i < N$.

2.2 Join Strategies

2.2.1 Nested Loop Join

i) Every node, i, broadcasts r_{mi} using shortest path routing.

ii) At each node i:

- Compute $r_{ni} \bowtie r_{mi}$
- On receipt of r_{mj} compute $r_{ni} \bowtie r_{mj}$, $0 \leq j < N, j \neq i$.
- Compute $\cup_{j=0,N-1} [r_{ni} \bowtie r_{mj}]$.

Analysis Step i) involves communication and Step ii) involves computation. Hence the message traffic in this algorithm is due to Step i). Since broadcast of each fragment involves N-1 links and there are N such broadcasts, the average traffic through a single link in an n-cube

(with Nn links[3]) is given by

$$T_{nl} = \frac{N(N-1)}{(Nn)}$$

$$= \frac{(N-1)}{n} \quad \textit{fragments/link}$$

Since R_m has t_m bytes/tuple and the whole relation is distributed over the N nodes of the Hypercube, the size of each fragment is $(|R_m|t_m) / N$. Hence

$$T_{nl} = \frac{(N-1)|R_m|t_m}{Nn} \quad \textit{bytes/link.}$$

For N >> 1,

$$T_{nl} \sim \frac{|R_m|t_m}{n} \quad \textit{bytes/link.}$$

2.2.2 Sort Merge Join

This algorithm is similar to the Nested Loop Join in that fragments r_{mi} are broadcast. However, both r_{mi} and r_{ni} are sorted at each site i before step 1 of the Nested Loop Join Algorithm.

Analysis While the computational requirements are somewhat different, the traffic, T_{sm} for this algorithm is identical to that for the Nested Loop Join Algorithm i.e.

$T_{sm} = T_{nl}$.

2.2.3 Semi-Join based Algorithm

i) Every node, i, computes $\pi_y(r_{mi})$ and $\pi_y(r_{ni})$ and broadcasts each of these lists to every other node.

ii) At each node i:

- On receipt of $\pi_y(r_{mj})$ and $\pi_y(r_{nj})$, $j \neq i$, compute the semijoins
  $r_{ni} \ltimes \pi_y(r_{mj})$ and $r_{mi} \ltimes \pi_y(r_{nj})$
- Perform an N-broadcast of
  $\{r_{mi} \ltimes \pi_y(r_{n0}),\ r_{mi} \ltimes \pi_y(r_{n2}),\ \ldots,\ r_{mi} \ltimes \pi_y(r_{nN-1})\}$
- On receiving $r_{mj} \ltimes \pi_y(r_{ni})$, $i \neq j$, compute $[r_{ni} \ltimes \pi_y(r_{mj})] \bowtie [r_{mj} \ltimes \pi_y(r_{ni})]$

---

[3]we assume that each edge in the Hypercube is comprised of two bidirectional links or channels

- Finally compute $\cup_{j=0,N-1}\{[(r_{ni} \ltimes \pi_y(r_{mj})] \bowtie [r_{mj} \ltimes \pi_y(r_{ni})]\}$

Analysis Let |y| be the size in bytes of the join attribute y. Part (i) of this algorithm involves a broadcast of y. Hence the average traffic through this phase, T_{SJ1} may be derived in a manner similar to that for T_{nl}

$$T_{SJ1} \sim \frac{|y||R_m|\sigma_y}{n} \quad bytes/link$$

$$where \quad \sigma_y = \frac{Number \ of \ distinct \ values \ for \ y \ in \ r_m}{|r_{mi}|}$$

i.e. $|r_{mi}|\sigma_y$ is the number of distinct attribute values for y in r_{mi}.

Part (ii) of this algorithm involves an N-broadcast. Let each of the N-1 items being broadcast traverse n' links on the average. Since there are N such broadcasts, the average traffic per link, T_{SJ2}, is

$$T_{SJ2} = \frac{(N-1)n'N}{(Nn)} \quad items/link.$$

The average distance between a pair of Hypercube nodes is

$$n' = \sum_{i=1}^{n} i\binom{n}{i}/(N-1) = \frac{Nn}{2(N-1)}$$

Upon substitution of n',

$$T_{SJ2} = N/2 \quad items/link.$$

Also each item is a semi-join output i.e. the tuples of R_m that participate in a join at another site. Hence the traffic in this phase, T_{SJ2} is given by

$$T_{SJ2} = \frac{|R_m|t_m\sigma_k}{2} \quad bytes/link.$$

where σ_k is the join selectivity

$$i.e.\ \sigma_k = \frac{|r_{mi} \bowtie r_{nj}|}{|r_{mi} \ X \ r_{nj}|}$$

2.2.4 Hash-based Join

This method essentially involves applying a hash function to the fragments of both relations at each node. For each relation, the function yields a set of hashed fragments, one for each node in the system.

These fragments are then N-broadcast. Since the hashing is done on the join attribute, a fragment of R_m arriving at node i joins only with fragments of R_n arriving at that node and no other.

Analysis Assuming that the hash function splits the relation fragments uniformly, each N-broadcast will involve a fragment of size

$$\frac{|R_m|t_m}{N^2} \ or \ \frac{|R_n|t_n}{N^2}.$$

Since the traffic for each of the 2 N-broadcasts is N/2 hash buckets per link per relation as explained in the derivation of T_{SJ2}, the traffic T_{HB} for this algorithm is

$$T_{HB} = \frac{|R_m|t_m+|R_n|t_n}{2N} \quad bytes/link.$$

2.3 The Collection Phase

Assuming a join selectivity of σ defined by

$$\sigma = \frac{|R_m \bowtie R_n|}{|R_m \ X \ R_n|},$$

the total traffic to the host will be $\sigma^2|R_m||R_n|$ tuples. Noting that there are n connections to the host, the traffic, T_{CP} through the maximally congested links in this phase is at least

$$T_{CP} = \frac{\sigma^2|R_m||R_n|t_m t_n}{n} \quad bytes/link$$

2.4 Comparison of Traffic in Phases 2 and 3

Tables 1-3 show the ratio of traffic in Phase 3 to that in Phase 2. We have assumed tuples sizes of 208 bytes[4] and the join attribute length of 52 bytes. Finally, we have also assumed worst-case traffic in Phase 2 i.e. σ_k=1, σ_y=1. Calculations are for a Hypercube of 32 nodes (n=5). As can be seen, the traffic ratio is always greater than 1 for the range of relation sizes and selectivities under consideration. Further this is true regardless of which join strategy is employed, though this ratio is highest for the Hash-based strategy and is lowest for the semi-join based algorithm.

Note that the traffic ratio increases linearly with relation size and quadratically with join selectivity. The fact is, that even at low selectivity and a relation size of 10,000 tuples, the traffic ratio is between 2 and 4 orders of magnitude. Also as N increases this ratio gets worse (increases) in the case of the hash-based join though the ratio in the other two cases are independent of network size.

3 HyperKYKLOS: The augmented Hypercube

From the tables in Section 2, it is clear that there exists a great imbalance between maximum link traffic in the compute phase and that in the collection phase. For example, even at σ=0.1, the traffic ratio is 17.5 for only 1000 tuple sized relations. Further, this is regardless of which algorithm is actually implemented. If query processing were to be viewed as a three-stage pipeline as shown in Fig. 2, the latencies through stages 2 and 3 would be related to the maximum traffic which as explained above is severely imbalanced. The crux of the problem is that the same physical hardware viz. Hypercube links and switches are used in both stages of the pipeline while the demands, in terms of bandwidth, are radically different in the two phases. What is needed is to expeditiously collect the results of the partial joins and present them as a sorted list (of tuples) to the host.

A solution is to construct a binary tree external to the existing Hypercube with the I/O nodes of the Hypercube as the leaves. Where multiple joins have to be performed or where the bandwidth requirements of Phase 3 are not adequately met, two or more trees may be employed. The use of multiple trees (the tree replication factor is denoted r) sharing the same set of leaf nodes to

---

[4] as in the extended Wisconsin Benchmarks [DeWI87]

σ	$\|R_n\|=10^3$	$\|R_n\|=10^4$	$\|R_n\|=10^6$
.1	17.5	175	17,500
.2	70.0	700	70,000
.5	437.5	4375	437,500
1.0	1750.0	17,500	1,750,000

Table 1: Traffic Ratio in Nested Loop Case

σ	$\|R_n\|=10^3$	$\|R_n\|=10^4$	$\|R_n\|=10^6$
.1	6.36	63.6	63,600
.2	25.4	254.4	254,400
.5	159.0	1590	1,590,000
1.0	636.0	6360	636,000

Table 2: Traffic Ratio in Semi Join Case

σ	$\|R_n\|=10^3$	$\|R_n\|=10^4$	$\|R_n\|=10^6$
.1	112	1120	112,000
.2	448	4480	448,000
.5	2800	28,000	2,800,000
1.0	11200	112,000	11,200,000

Table 3: Traffic Ratio in Hash based Case

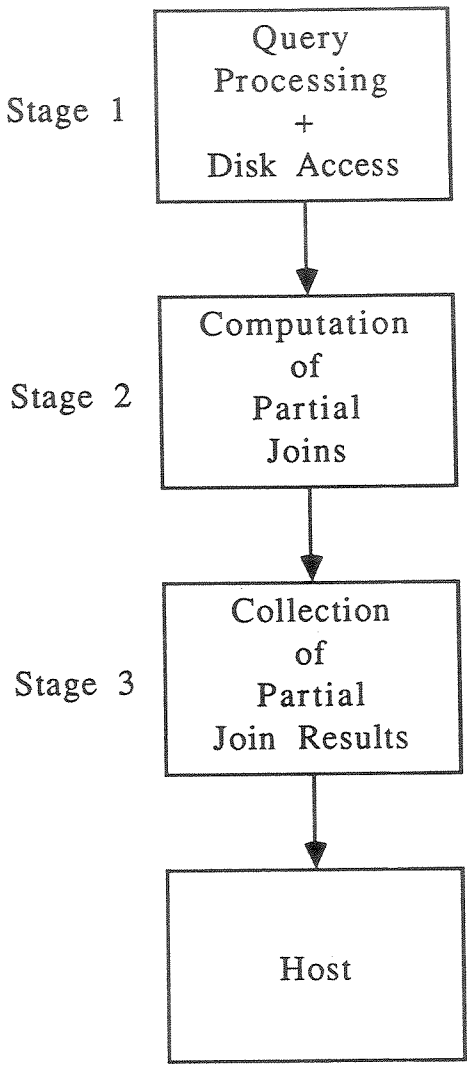


Figure 2. 3-Stage Pipeline Representation of Distributed Join Processing in a Hypercube.

improve performance and provide fault tolerance has been studied as the KYKLOS topology [MENE86,JENE86].

The advantages of such constructions are obvious: the properties of the composite topology are a superset of the properties of its constituent parts. Algorithms that require message transmission from each node to every other node would be well suited to the Hypercube; such algorithms would result in O(N) maximum traffic density. On the other hand for operations that require merging or sort-merge algorithms the use of tree structures is highly desirable. Though a tree may be mapped onto a Hypercube, a single connection from the root of this tree to the host could result in serious traffic bottlenecks at the root with concomitant imbalance in bandwidth utilization. This would result in a system that is network-bound (i.e. the input capacity of the host or hosts exceeds the bandwidth of the fastest link(s)) especially in the event that cartesian products or low selectivity joins be required at the host. As such, we are investigating the possibility of providing multiple trees - an alternative which seems attractive in the light of the fact that there may be many hosts and multiple queries to be processed. One such alternative is a derivation of the KYKLOS Network explained below.

A special case of the KYKLOS Network is one where r=logN (Fig. 3(a)). Because this consists of trees built on top of a Hypercube, it has been christened HyperKYKLOS.

Topology Definition We have defined KYKLOS as a multiple-tree structure sharing a set of leaf nodes. We could redraw each tree separately as a full binary tree with no link crossovers. (Fig. 3(c)). Each tree may then be characterized by a Labelling Sequence (LS) [MENE86] of leaf nodes. As an example, the LS's for each tree of Fig. 3(c) are

$L_0 = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7$

$L_1 = 0\ 2\ 4\ 6\ 1\ 3\ 5\ 7$

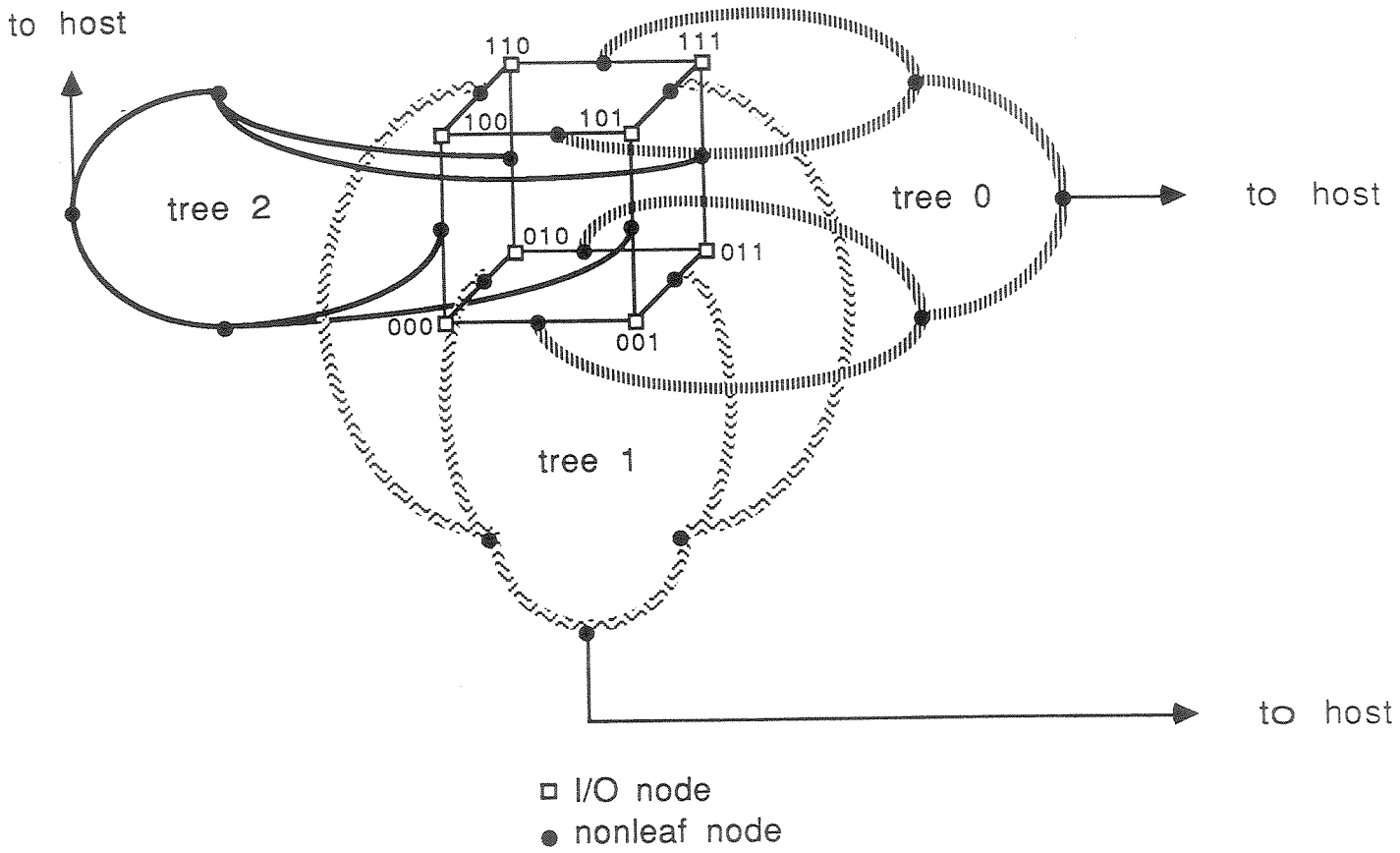
$L_2 = 0\ 4\ 1\ 5\ 2\ 6\ 3\ 7$

Since a HyperKYKLOS of N leaf nodes has n = logN trees, the interconnection strategy for such a structure could be defined by means of the tuple

$<L_0, L_1, \ldots L_{n-1}>$

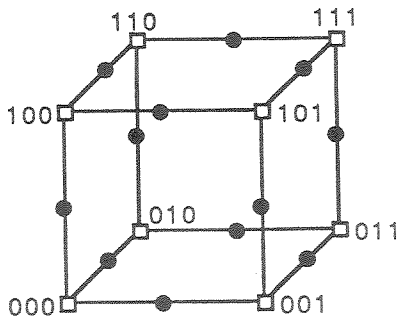
where L_i is the LS for the i^{th} tree. We have chosen a simple interconnection strategy for the version of HyperKYKLOS presented in this paper which we call HyperKYKLOS-I. The LS's for HyperKYKLOS-I are defined below

Let $L_i(j)$ be the j^{th} term of L_i. Then



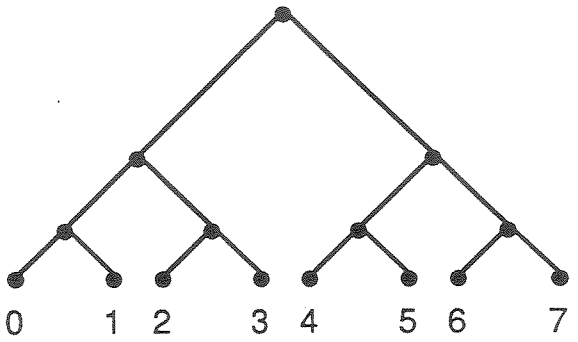
HyperKYKLOS with 3 Trees

Figure 3(a)

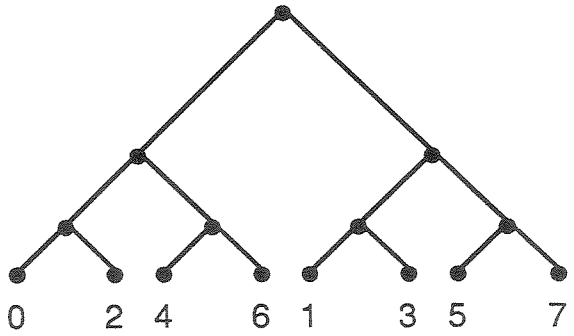


Modified Hypercube

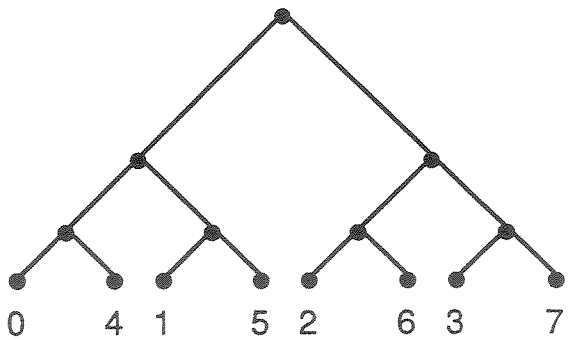
Figure 3(b)



LS for Tree 0



LS for Tree 1



LS for Tree 2

Labeling scheme for 3 trees.

Figure 3 (c)

$L_i(j) = \rho_i(j)$ where

j is an n-bit binary string and

$\rho_i(j)$ is the number obtained rotating j a total of i bits to the left.

In HyperKYKLOS, there are logN paths (through the roots of each tree) to the host. This provides a logN-fold improvement in traffic to the host especially when a low-selectivity join is to be performed with the result returning to the host. Where selectivity is low and cost considerations dominate, only a small subset of trees may be used (i.e. $r < \log N$). In this case any subset of r LS's may be used to define the r trees.

Finally we emphasize that if Stage 3 of the pipeline (Fig.2) is to cease to be a bottleneck it is not sufficient to merely have the extra tree(s). In fact, two requirements of the tree structure are clearly identifiable:

- High speed comparator logic at each non-leaf node to perform on-the-fly merge operations.
- High-speed links especially at the root of the tree

Given that the host may be a high-speed computer (or supercomputer), the onus of providing high performance lies with the designer of the I/O system.

4 The Architectural Prototype

We have identified at the gross system level the two major components of the architecture viz.

- The I/O nodes linked in a Hypercube configuration
- The sort/merge trees for Phase 3.

We next present a brief description of the architecture of the I/O node prototype under construction (Fig. 4).

4.1 I/O Node Prototype

Each I/O node will be composed of the following blocks (shown in dashed lines).

(a) Control Processor (CP)

(b) Sort/Search/Set Engine (S^3E)

(c) Disk System (DS) including disk cache, MC68020 and Mass Storage

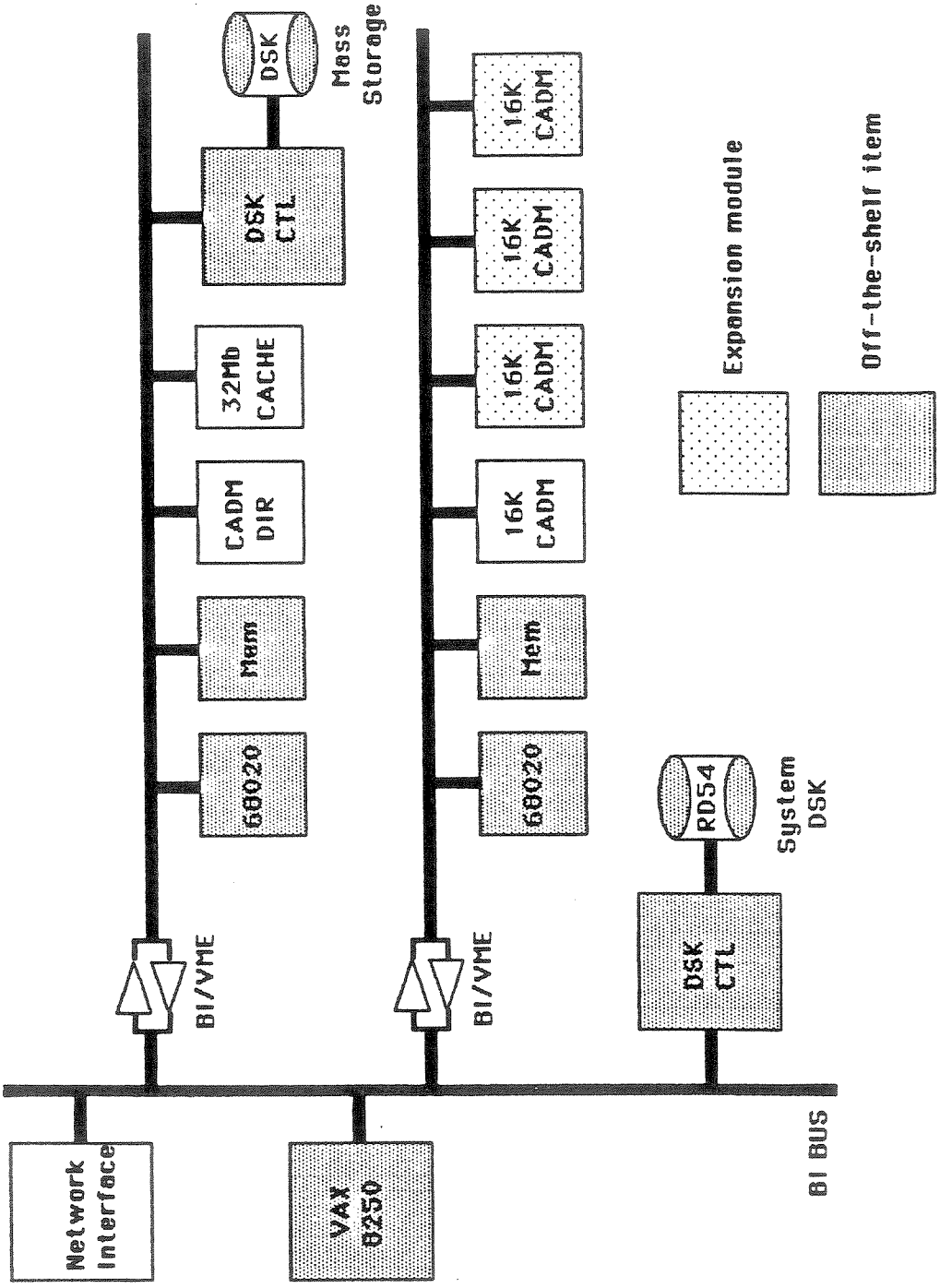

Single I/O Node
Network Interface
VAX 8250
BI BUS
BI/VME
68020
Mem
CADM DIR
32Mb CACHE
DSK CTL
DSK
Mass Storage
BI/VME
68020
Mem
16K CADM
16K CADM
16K CADM
16K CADM
DSK CTL
RD54
System DSK
Expansion module
Off-the-shelf item

Figure 4

(d) Network Interface Unit (NIU)

The VAX8250 (CP) with 4MByte of local memory is being used to control the I/O node system. The NIU multiplexes the log N incoming channels and routes data through the appropriate link(s). Block transfers between CP and NIU/S^{3}E/DS are through a high-speed BI bus which operates at 13.3 MByte DMA transfer rate. The BI bus was chosen because it uses distributed arbitration thus eliminating the need for a dedicated arbiter.

The S^{3}E is the heart of the I/O node design, so called because it performs sorting, search and set operations (union, intersection,etc.). It is being designed utilizing an array of Content Addressable Data Manager (CADM) chips designed by Advanced Micro Devices Corporation (AMD). Each CADM chip has 1 KByte of memory. We are currently building a 16K S^{3}E using an array of 16 CADM chips. We have made provision for S^{3}E expansion of upto 64K bytes (Fig. 4). The Motorola MC68020 is being used to provide the control for the CADM array's sort/search/set operations as well as for interfacing the S^{3}E to the CP. Our choice of the CADM chips was based on results of a simulation which indicated an expected 15-50 speedup in sorting over the VAX 11/780 and a speedup of 1.5 over software sort on the CRAY X/MP-24.

The DS will use a Disk Controller and nominally between one and four 500 MByte Disk Drives. The cache capacity has been targeted at 8% of mass storage. We are currently simulating the performance of the cache system. The actual design of this block will be driven by results of this simulation. Our tentative design envisages a track-organized design so that the minimum granularity for disk transfer is a track. The data filter can resolve the location of an object within a track residing in cache in 6 μsec. The data filter is currently being implemented in software on the MC68020.

The BI/VME converters are controllable bus interfaces between the BI and VME busses. This means that transfer through these converters occurs only by command of the CP. This bus structure will allow the cache subsystem, S^{3}E subsystem and CPU/NIU subsystems to function concurrently thus providing speedup due to the overlap of their respective operations.

5 Conclusion

We have shown that implementing queries on a Hypercube with horizontally fragmented data causes a serious traffic imbalance when the results are sent to the host, particularly if they are required to be sorted. The architecture proposed remedies this imbalance by having a tree structure for the collection phase. This architecture also has some other nice properties as demonstrated in [MENE85].

We strongly believe that the performance of the architecture proposed will be superior to that of a multiprocessor based purely on the Hypercube.

REFERENCES

[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.

[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.

[DeWI87] DeWitt, D.J, M.Smith, H.Boral, "A Single-User Performance Utilization of the Teradata Database Machine" *MCC Technical Report # DB-081-87*, March 5, 1987.

[MENE85] 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.

[VALD84] Valduriez, P. and G.Gardarin, "Join and Semi-Join Algorithms for a Multiprocessor Database Machine" *ACM TODS*, vol 9, March 1984.