Query Scheduling and Site Selection Algorithms for a Cube-Connected Multicomputer System

Ophir Frieder
Bell Communications Research,
435 South Street, Morristown, NJ 07960

Chaitanya K. Baru
Advanced Computer Architecture Lab
Dept. of EECS, The University of Michigan
Ann Arbor, MI 48109

Abstract

Query scheduling and site selection algorithms for read-only queries on a cube-connected multicomputer are presented. The assumed global system architecture was initially presented in [Fri87]. An architecture model for the system is provided, and via this model, a site selection algorithm which determines where to execute the upcoming operation sequence is developed. The query trees of queries entering the system are converted into operation sequence trees. Operation sequences belonging to a query are queued until the query is initiated. Two query selection policies are presented. A simulation comparing the two policies is described, and the simulation results are discussed. The scheduling restrictions which guarantee the avoidance of deadlock in both algorithms are presented.

1. Introduction

There is much current interest, in both the research and industrial communities, in studying hypercube multiprocessors. A hypercube is an n-dimensional boolean cube, Q_n, defined as a cross product of the complete graph K_2 and the (n-l)-dimensional boolean cube Q_{n-1}, with $Q_1 = K_2$. Each node is connected (or adjacent) to each of its $n = \log_2 N$ neighbors, where N is the number of nodes. For example, in a 4 dimensional cube, Q_4, node 0000 is adjacent to nodes 0001, 0010, 0100, and 1000. Several existing hypercube based machines include INTEL's IPSC [Int85], NCUBE's NCUBE [Hay86] and Floating Point Systems' (FPS) T/1000 series [Fre86].

Despite their common interconnection topology, these machine differ greatly in their hardware implementation. For example, INTEL's IPSC can be assembled in a Q_5 (32 node), a Q6 (64 node), or a Q_7 (128 node) configuration; the NCUBE machine can contain up to 1024 nodes and the FPS configuration is designed to contain over 16,000 nodes. The disk configuration and the I/O bandwidth also differs greatly among these machines. In the NCUBE, a disk controller is connected to a *subcube* of processors with the associated processors sharing access to the common disk. On the other hand, the FPS T-Series supports a disk at each node. Despite their architectural differences, however, most applications targeted for these systems are similar, concentrating on large scale, computationally intensive, numerical applications.

Recently, some algorithms for relational database operations on a hypercube connected *multicomputer* system were suggested in [Bar87a, Bar87b, Bar87c]. A multicomputer is defined as a system consisting of an interconnection of multiple independent computers each with its own CPU, independent operating system, main memory, secondary storage, software utilities, etc. A 3-dimensional cube multicomputer system (i.e., number of nodes, N = 8) with a disk attached to each node is shown in Figure 1.

The database algorithms presented are unique in that they *dynamically* or *on-the$y* account for poor intermediate data distribution by balancing relation tuples roughly evenly across all nodes. The time involved in performing some common database operations were calculated in [Bar87c] and compared favorably with other database multicomputers such as the SM3[Bar86, Su84] and the NONVON [Hil86,Sha82].

This paper presents an approach to query processing in a cube system. Query scheduling strategies that support intra- and inter-query concurrency for read-only queries are discussed. The scheduling assumes a global system architecture and an initial data distribution as described in [Fri87]. The remainder of this paper is organized as follows. Sections 2 and 3 provide overviews of the cube-based, relational database algorithms and the initial data distribution and proposed global system architecture, respectively. Section 4 describes in detail the query scheduling algorithm. A brief summary concludes the paper as section 5.

2. Relational Database Algorithms on a Hypercube

The database operations introduced in [Bar87a, Bar87c] are divided into those that implement the relational operations,

CH2541-1/88/0000/0094$01.00

such as Select, Join, etc., and others that are used to support dynamic data redistribution, i.e., the "on-the-fly" reorganization of data to promote a more balanced workload. Brief descriptions of the data redistribution and database operations are provided here.

Tuple Balancing: redistributes relation tuples to a roughly even distribution across all the nodes of a cube or subcube.

Relation Compaction & Replication (RCR): replicates a relation stored in a cube of dimension **n,** such that after the operation each of the two, equal-sized. dimension **n-l.** subcubes of the original **cube** contain a copy of the relation.

Relation Compaction (RC): same as RCR except that only one of the two logical subcubes finally contains the data.

Cycle: embeds a ring within each subcube formed as a result of the RCR or RC operations, and pipelines data packets around the ring.

Selection: Each node performs selection on a local segment of a relation in parallel. If the results are to be collected, then an output collection step is incorporated, otherwise no global operation is necessary. As with all the operations, operation termination is signaled via the use of global ***control lines*** [Pet85].

Scalar Aggregation: First, all the nodes compute their local aggregate value. Next, the global aggregation phase commences. In the **kth** step, **k** = 1 to log*N, nodes whose rightmost **k** address bits are equal to the rightmost **k** bits of the host address, obtain the aggregate value from the nodes which differ in address from the host in the kth bit and compute the next local aggregate.

Project: Initially, a local projection (removing non-relevant columns from each tuple and eliminating local duplicates, if neccesary) is performed. The tuple distribution across nodes may become skewed as a result of this step. In this case, tuple balancing is performed as the first step. The optional tuple balancing step is followed by one or more RC steps in each of which, nodes eliminate duplicates between the local and incoming tuples. If an RC step cannot be performed, then tuple balancing is performed and the RC is retried. The algorithm enters the cycle step if RC is not possible even after tuple balancing. Global duplicates are eliminated in this last step.

Join: The relational join operation is always preceded by tuple balancing and the RCR operation. This ensures that the data are evenly distributed and the subcube size used in the cycle step is reduced as much as possible. Finally, the cycling operation is used to send the tuples of the smaller relation around in a ring and perform local joins at each node.

It is appropriate to comment on the use of semi-joins, at this point. Semi-joins are useful in situations where the communication time clearly dominates over the computation time. The semi-join reduces the amount of remote data transfer (communication) at the expense of additional computational burden (performing the join computation several times). However, preliminary results show that semi-join algorithms perform very badly in hypercubes [Men87]. For additional details concerning the data redistribution and relational database operations see [Bar87c].

3. System Architecture and Initial Data Distribution

The system architecture consists of a single directory node called the control node, CN, (a O-dimensional cube, Qo), connected via a bus to a larger cube (Qm) of Output Collection Nodes, OCN's. The cube of OCN's is called the Control Cube, CC. Each OCN in Q, is in turn connected via a bus to one of m disjoint subcubes, called Initial Storage Subcubes, IS%, of size Qn-, where **n** is the cubical dimension of the base cube. Note that each node in the base.cube is connected via a bus to exactly one OCN. Finally, the nearest neighbor links which connect the ISS's to one another am called the data routing links. Figure 2 shows a 64-node base cube controlled by an 8-node control cube, which in turn, is controlled by the control node. The lines connecting the circles (ISS's) signify the data routing links. A relation is contained entirely within an ISS, with the tuples evenly distributed. Each relation is uniquely identified with one ISS. A global relation directory is maintained for the entire system at the CN. For each entry there are four fixed size fields-relation name, tuple size in bytes, number of tuples, and the ISS number.

4. Scheduling and Site Selection

Query processing is initiated by the arrival of an "optimized" query tree at the Control Node (CN). The query tree is transformed into an operation sequence tree (OS-tree) (operation sequences are defined in section 4.1). The CN directs the necessary disk I/O commands to the relevant OCN which queues disk requests and broadcasts them on a first-come-first-served basis to the ISS nodes. The disk I/O requests are accompanied by the qualifiers required for any selection to be performed on the base relations. Since the base relations are evenly partitioned across all the nodes of the ISS, the disks in the ISS nodes complete their service at about the same time. Further operations in the query are scheduled at the neatest idle ISS. An idle ISS is one in which the CPU's of all the nodes are not busy (see 4.3). If **d** represents the

number of data routing links to be traversed from the current ISS to a given ISS, then the nearest ISS is the one for which **d** is a minimum. Note that when **d** = 0, the current ISS is itself the nearest ISS.

4.1 Operation Sequences

An operation sequence, OS, is a list of consecutive, relational operations in a query tree with all the operations, except possibly the first, being unary operations. The OS is the smallest schedulable unit in a query The fiit operation in an OS can be a unary or binary operation. The length of an OS, L, is defined to be the number of unary operations in the sequence. Three types of sequences are defined, selection, simple, and join. A single selection operation performed on a base relation is called a selection sequence (sel-seq). Therefore, the length of a sel-seq, Lsel = 1, always. All sel-seqs are executed at the ISS storing the base relation. A simple sequence, sim-seq, is defined as a sequence of unary database operations immediately following a selection. The length of a simple sequence, Lsim 2 1. A join sequence, joingeq, is any sequence in which the first operation is a join. By definition, a join sequence which is not followed by a unary operation (other than output collection), has a length, Ljein = 0. Additionally, a base join sequence, bjoin_seq, is defined as a join sequence with at least one of its inputs being a simple or selection sequence. The particular significance of the base join sequence is discussed in section 4.6.

Figure 3 shows a query with three sel-seqs consisting of the selections Sl, S2, and S3; two join-seqs, one consisting of Jl followed by Pl and the other of only a single join, J2; and one sim-seq consisting of P2. Both the join sequences in this query are base join sequences. In general, a query consists of at least one selection sequence and zero or more join or simple sequences. By scheduling operation sequences and not individual operations, the number of scheduling messages and the data movement during query execution is reduced. Furthermore, the evaluation of the individual operations within a sequence can proceed in a pipelined manner, further reducing the query execution time.

4.2 Site Model

For the purposes of scheduling and site selection, the system can be viewed as a set of independent nodes consisting of a CPU and I/O processor pair, i.e. NODE = (CPU, I/O). The main memory of a NODE is sufficient to accommodate at most two input relations, partial results of an output relation, and possibly, a single output block per node from the selection sequence executing in the subcube. The NODE's are assumed to be connected via a point-to-point ring, with all scheduling decisions determined by a central node, CN, adjacent to all the NODE's as shown in figure 4. In the case of the cube system, each NODE is actually a subcube; the point-to-point ring is simulated via the data routing rings; and the CN is replaced by the 2-level hierarchy of the CN and CC. The assumption on the NODE memory size is justifiable since hypercubes with nodes containing up to 8 Mbytes of user memory have been implemented pet851. Thus, a 16-node subcube would have a total memory of roughly 128 Mbytes.

4.3 The Sequence Directory

The CN maintains a sequence directory which stores all the information contained in an OS tree for each active and inactive query in the system. Figure 5 illustrates a sequence directory. Each entry in this directory has a distinct sequence number (Seq #), sequence type (Seq Type), query submission/ initiation time (Init Time), input relation(s) required (Input 1, Input 2), sequence number of the successor sequence or number of the host if this is the last sequence in the query (Succ), the ISS on which it is/was scheduled (ISS run) in the case of an active query, and an indication whether the query is active or inactive (Active Query). A zero in the Active Query column indicates that the query has not yet been scheduled, while a 1 indicates that the query is currently in execution.

A Subcube Activity Table (SAT) is also maintained which provides information about which subcubes are busy/free. Whenever a subcube is freed the CN scans the sequence directory and schedules the first "ready" sequence. A sel-seq is ready if the ISS containing the required base relation is free. A sim-seq or join-seq is ready when the required input/s is/are available. If there is no ready sequence and there are sufficient resources (i.e. subcubes) to execute the next query, then a new query is activated. Once a query terminates, the corresponding row entries are logged and deleted from the Sequence Directory.

4.4 Intra- and Inter-query Concurrency

Operation sequence scheduling to support intra-query concurrency is explained here assuming only a single active query present in the system. Fist, all sel-seqs are scheduled at the appropriate lSS's containing the base relations. All sequence scheduling decisions, except for sel-seqs, are

dynamic. That is, site selection decisions are made during runtime, as and when required by the query and not in advance. Figure 6 shows an OS tree for a query that will be used in explaining the scheduling algorithm. There are three sel-seqs, Sel, Se2, and Se3, one sim-seq, Sil, and two join-seqs, Jsl and Js2, in this query. The sel-seqs operate on inputs Rl, R2, and R3, respectively. Both, Rl and R3 reside on ISS 1, and R2 resides on ISS 3.

All sel-seqs are scheduled first. The output of a se&q is the input for either a sim-seq or a join-seq. In the case of a sim_seq, the sequence is scheduled on an idle ISS which is nearest to the ISS executing the sel-seq. The join-seq, on the other hand, has two inputs, say X and Y. If input X becomes available frost, then the join-seq is scheduled on an ISS which is closer to Y and vice-versa. The moment a site is selected for a join-seq, both inputs X and Y (in partial or full form) are routed to that site and tuple balancing of both begin. In figure 4, if Se2 completes first, then the join-seq, Jsl, is scheduled near Sel. The details are as follows. Assuming that Jsl is scheduled on ISS 1, the output of Se2, R2', is routed to ISS 1. Since ISS 1 has been chosen for Jsl, the two inputs R1' and R2' are balanced at ISS 1. Thus, the balancing involves all the R2' tuples and the R1' tuples that are currently available. Upon the completion of Sel, the balancing taking place at ISS 1 is terminated and Jsl is initiated. Since both input relations of Jsl were being balanced, the tuple distribution is expected to be roughly even across all nodes. Thus, the tuple balancing step of Jsl will take less time.

Similarly, when sel-seq Se3 terminates, the sim-seq Sil is scheduled for execution on ISS 5 (ISS 1 is assumed to be busy). Finally, join-seq Js2 is scheduled as follows. The first input sequence to terminate (say, Sil), notifies the CN of its completion and initiates tuple balancing on its output. This continues until the second sequence (Jsl) terminates. When the output from Sil and Jsl is available at ISS 1, the join-seq Js2 is initiated. Output collection is overlapped with result computation via the OCNs.

4.5 Site Selection for Join-seqs

Further details on site selection for join-seqs are considered here. Assume that the two input sequences to a join sequence are SEQ1 and SEQ2. Furthermore, assume that SEQ1 is scheduled at ISS X and SEQ2 is scheduled at ISS Y. Since each of the two inputs to the join can either be selection or simple/join sequences, there are four possibilities to be considered. In each of these cases SEQ 1 is assumed to block/terminate first. A sequence blocks if one or more nodes in its ISS produces more output than the memory of that node can hold. It is assumed that is at least one ISS is available to execute the upcoming operation sequence. This is ensured by the query selection policies described in section 4.5 below. The four possible cases are as follows.

1. Both SEQ 1 and SEQ 2 are executing selection sequences.
2. SEQ 1 is executing a selection sequence, and SEQ 2 is executing either a simple or a join sequence.
3. SEQ 1 is executing either a simple or a join sequence, and SEQ 2 is executing a selection sequence.
4. Both SEQ 1 and SEQ 2 are executing either a simple or a join sequence.

The site selection decisions are explained below, in semi-formal language, for the four cases. In each case, the ISS at which the join-seq is scheduled is called, Z, and the join-seq is initiated only after both its input sequences have terminated and all the data has arrived at Z. However, the balancing of input data can be begun in advance as indicated in each of the cases below. Note that the node CPU's are involved in the tuple balancing operations. Thus, balancing can be done only by those ISS's whose CPU's are idle. If an ISS is executing a sel-seq and its CPU's are busy, then the output of the sel-seq is routed directly, without being balanced, to the ISS where the next sequence is executed.

Case 1.
SEQ1 is a sel_seq executing on ISS X,
SEQ2 is a sel_seq executing on ISS Y,
if SEQ1 *blocks* first **then**
begin
Estimate the completion times of SEQ1 and SEQ2 based on their respective start times and relation sixes;
Choose ISS Z to be closer to the slower sequence
end
else if SEQ1 *terminates* first **then**
Choose ISS Z to be closer to ISS Y,
Route the current outputs from X and Y to Z and start balancing at Z;

Case 2
SEQ1 is a sel-seq executing on ISS X;
SEQ2 is a sim-seq or a join-seq executing on ISS Y;
if SEQ1 *blocks* first **then**

Since it is not possible to easily estimate the completion time of SEQ2, choose ISS Z to be closer to X;

else if SEQ1 *terminates* first **then**

Choose ISS Z to be closer to ISS Y,

Route the current outputs from X and Y to Z and start balancing at Z;

Case 3.

SEQ1 is a sim-seq or join-seq executing on ISS X;

SEQ2 is a sel_seq executing on ISS Y,

if SEQ1 *blocks* first **then**

Balance data in ISS X until block is cleared and continue with SEQ1

else **if** SEQ1 *terminates* first **then**

begin

Choose ISS X itself to be the site for the upcoming join_seq;

Route the current output from Y to X and start balancing the data at X;

end;

Case 4.

SEQ1 is a sim-seq or join-seq executing on ISS X;

SEQ2 is a sim-seq or join-seq executing on ISS Y;

if SEQ1 *blocks* first **then**

Balance data in ISS X until block is cleared and continue with SEQ1

else if SEQ1 *terminates* first **then**

begin

Choose ISS X itself to be the site for the upcoming join-seq

Start balancing data at X until SEQ2 terminates;

When SEQ2 terminates, tranfer data from X to Y (since the data at X is already partially (or fully) balanced, this will be more efficient than transferring Y's unbalanced data);

end;

There is one exception to the above site selection algorithm in the case when both inputs to a join-seq are sim_seqs. It is stated above that a sim-seq should be scheduled at an ISS which is nearest to its input. However, in this case, the slower sim-seq should be scheduled at the same site where the faster sim-seq is executing. This scheme reduces the number of ISS's needed to execute a query (i.e. reduces the ASC of a query, see 4.6) and the expected data routing time.

4.6 Query Selection Policies

When several queries are waiting for execution, the order in which they are selected can influence performance. This section describes and compares two query selection policies for the cube-connected multicomputer. Queries are selected for execution based on their resource demands. A resource is considered to be an ISS used to execute a simple or join sequence (note that the loading caused by selection sequences is assumed to be zero) The maximum number of concurrent operations at any level of a query tree decreases as one proceeds from the leaves to the root of the tree. As one proceeds up the levels of the tree, the number of join and/or simple sequences at each level decreases. In fact, the number of base join sequences in a query provides a measure of the maximum number of concurrent resources (ISS's) required to execute the query. Assigning this number of ISS's to a query permits maximum concurrency at the lower levels of the query tree. Thus, the resource demands of a query are modelled in terms of the number of base joins in a query. In addition, queries consisting only of a simple sequence also require a single resource (ISS). As the operation sequences at the lower levels terminate, ISS's are relinquished by a query. Finally, in keeping with the above assumptions, queries with only selection sequences make no demands on the resources of interest and are selected accordingly in each of the four selection policies discussed below.

Each query is assigned a count indicating its maximum resource demands. This count is called the Active Sequence Count (ASC). The sum of all the ASC's of all currently active queries is called the Total Active Sequence Count (TASC). The number of available ISS's at any time is = the total number of ISS's in the system - TASC. A query is selected for execution only if its ASC does not exceed the number of available ISS's in the system. Two policies which select the next query to execute from among the set of waiting queries are investigated

1. Maximum Active Sequence Count (MASC) policy, i.e. select the query with the greatest ASC first.
2. Least Active Sequence Count (LASC) policy, i.e. select the query with the least ASC first.

The results obtained by simulating the system using these query and site selection policies are provided in section 4.5.

4.6.1 Indefinite Wait and Deadlock Prevention Mechanisms

The query selection policies must insure that both system deadlock and indefinite wait conditions are avoided. Indefinite waits are prevented by timestamping queries with arrival times. Once a query has "sufficiently aged", it is placed at the head of the query selection queue, and no additional queries are selected ahead of it. When the resources required to execute the query become available, the aged query is initiated.

System deadlock is avoided by limiting the number of active queries in the system. Since only a limited number of relations can simultaneously reside within a subcube's local memory, too many operations/relations simultaneously present in the system (too many queries initiated) may lead to deadlock. A deadlock state may arise if an ISS X blocks as a result of sel_seq computation, but all the ISS's are currently busy/active awaiting the arrival of the slower sequence required to execute the join-seq already scheduled at ISS X. This deadlock is prevented by maintaining a running TASC count and ensuring that the TASC never exceeds the number of subcubes in the system. The following TASC update algorithm is used in the CN.

TASC Incrementation :

The TASC is incremented by the ASC count whenever a new query is scheduled, i.e. TASC := TASC + ASC of new (active) query.

TASC Decrementation :

The TASC is decremented by 1, TASC := TASC-1, whenever (1) a query containing at least one simple or join sequence terminates or (2) a join sequence with at least one join sequence as input terminates.

4.7 Simulation Study

A simulation was developed to evaluate the proposed query selection policies. As input, the simulation is provided with a set of base relations and queries and returns as output the global and the average query completion times, as well as the average initiation delay for all the queries in each query *mix*. The *global query-completion time* is the time from the arrival of the first query set until the completion of the last query; the *average query-completion time* is the average of the individual completion times of all the queries in the set; and the *average initiation delay* is the average of the time each query in the set must wait from its arrival time to the time it is actually scheduled.

For this simulation it was assumed that the system consists of a 1024-node base cube, divided into 64 ISS's of 16-nodes each. Each node was assumed to contain a 2 MIPS CPU with internode communication handled by the communication processors using 20 Mbit/sec nearest neighbor links. A maximum limitation of 65,536 bytes was imposed on the packet size. A total of 100 different base relations were generated, with each relation consisting of between 2000 and 10,000 tuples. All tuples were of 128 bytes, and the exact number of tuples in each of the 100 relations was randomly generated. The relations were distributed across the 64 (16 node) subcubes, such that each of the 36 lower numbered (0 - 35) subcubes housed two relations, and the remaining 28 subcubes stored only one relation each. Within each subcube, the relations were evenly partitioned across all the subcube nodes.

To obtain reasonable results, one should generate a query mix that is representative of a typical work-world. However, as reported in [Bor84], [Dew85], [Eic85], and [Haw85], characterizing a "typical" query mix is quite difficult. The query mix chosen here included eight query types of varying complexity. The base relations accessed by a query were determined by generating random numbers that linked queries to relations. The query mix consisted of 200 queries distributed as follows :

Type 1. 60 queries, each consisting of only a single selection operation.

Type 2. 50 queries, each consisting of only a single join operation.

Type 3. 35 queries, each consisting of a 2-join chain. At least one of the inputs of each join is a selection operation.

Type 4. 15 queries, each consisting of a 3-join chain. At least one of the inputs of each join is a selection operation.

Type 5. 12 queries, each consisting of a 4-join chain. At least one of the inputs of each join is a selection operation.

Type 6. 8 queries, each consisting of a 6-join chain. At least one of the inputs of each join is a selection operation.

Type 7. 12 queries, each consisting of 3 joins in the shape of a full binary tree.

Type 8. 8 queries, each consisting of 7 joins in the shape of a full binary tree.

Results from running the simulation on six different data sets (the same set of queries, with the base relations involved in each query differing from set to set) are presented in figure 7. For all six data sets, and for both query scheduling policies, the global and the average individual query completion times, as well as the average query initiation delay, are computed (the first, second, and third row entries per data set, respectively). The six data sets should not be compared against one another since they represent different base relation access requirements. However, it is always the case that the MASC policy results in lower global completion times but higher average processing times and average initiation delays. The lower global completion times of the MASC policy is directly related to the availability of the system resources. At startup, or under low resource utilization, all system resources am available. Thus. it is possible to initiate many complex queries without incurring long initiation delays. Since the simple queries can be readily scheduled without introducing long query initiation delays (few resources are required to execute them), scheduling them later reduces the overall processing time. On the other hand, if the simple queries are scheduled ahead of the complex queries (utilizing LASC), then when the complex queries are actually scheduled, relatively long initiation delays result

The longer average completion time and initiation delays introduced by the MASC query selection policy is the result of the long time duration required to execute the complex queries which are executed ahead of the simple queries. Similar to the proofs concerning the CPU job scheduling algorithms presented in [Pet83], it is possible to prove that if the shortest jobs (simple queries) are executed first, the average execution rate and initiation delays will be optimally low. The simulation results obtained agree with this claim,

5. Summary

This paper presented an approach for scheduling read-only queries in a cube-connected multicomputer. Similar policies can be used, in general, in a highly parallel, distributed memory machine. However, the particular architecture considered in this work is based on the hypercube. In this system, an input query tree is transformed into an operation sequence tree, and the operation sequences are queued until the query is initiated The operation sequences of an active query are scheduled dynamically, at runtime. Algorithms for dynamic site selection were provided. Also presented were two policies which select the query to be initiated from among the waiting queries. Both query selection policies incorporate restrictions which insure that system deadlock and query indefinite wait conditions do not occur.

References

[Bar86] Baru,C.K. and Su,S.Y.W., "The Architecture of SM3: A Dynamically Partitionable Multicomputer with Switchable Memory," *IEEE TC,* pp.790-802, Sept. 1986.

[Bar87a] Baru,C.K. and Frieder,O., "Implementing Relational Database Operations in a Cube-Connected Multicomputer", *Procs. IEEE 3rd Intl. Conf. on Data Engineering,* February 1987, Los Angeles, CA.

[Bar87b] Baru,C.K., Frieder,O., Kandlur,D., and Segal, M., "Join on a Cube: Analysis, Simulation, and Implementation", *Procs. of the 5th Intl. Workshop on Database Machines,* Japan, 1987.

[Bar87c] Baru, C. K. and Frieder, O., "Database and data redistribution operations on a cube-connected multicomputer," under review, Dec. 1987.

[Bor84] Boral, H. and Dewitt, D. J., "A Methodology for Database System Performance Evaluation", *Procs. ACM SIGMOD,* pp 176-185, June, 1984.

[Dew85] Dewitt, D. J., "Benchmarking Database Systems: Past Efforts and Future Directions", *IEEE Database Engineering,* Vol. 8, No. 1, pp 2-9, March, 1985.

[Eic85] Eich, M. H., "Transaction Oriented Performance Analysis of Database Machines", *IEEE Database Engineering,* Vol. 8, No. 1, pp 53-60, March, 1985.

[Fre86] Frenkel, K. A., "Evaluating Two Massively Parallel Machines," *CACM,* 29, 8, August 1986.

[Fri87] Frieder,O. and Baru,C.K., "Data Distribution and Query Scheduling Policies for a Cube-Connected Multicomputer System", *Procs. 2nd Intl. Conf. on Supercomputing Systems,* May 1987. San Francisco, CA.

[Haw85] Hawthom,P.B., "Variations on a Benchmark", *IEEE Database Engineering, 8.1,* pp 19-28, March, 1985.

[Hay86] Hayes, J. P. et al "Architecture of a Hypercube Supercomputer", *IEEE MICRO,* Oct. 1986.

[Hil86] Hillyer, B. and Shaw, D. E., "NON-VON's Performance on Certain Database Benchmarks," *IEEE TOSE,* SE-12, 4, April 1986.

[Int85] Intel iPSC Data Sheet, Order No. 280101-001, 1985.

[Men87] Menezes,B.L., "Design of a HyperKYKLOS-based Multiprocessor Architecture for High-Performance Join Operations," *5th Intl. Workshop on Database Machines,* Oct. 1987, Japan.

[Pet83] Peterson, J. L., Silberschatz, A., *Operating System Concepts,* Addison-Wesley Publishing Co.. 1983, Reading, Mass.

[Pet85] Peterson,J.C. et al, "The MARK III Hypercube-Ensemble Concurrent Computer", *Procs. Intl. Conf. on Parallel Proc.,* August 1985.

[Sha82] Shaw, D. E., "The NON-VON Supercomputer," Dept. Comp. Science, Columbia Univ., Tech. Rep., August, 1982.

[Su84] Su, S. Y. W. and Baru, C. K., " Dynamically Partitionable Multicomputers with Switchable Memory", Journal of Parallel and Distributed Computing 1, pp 152-184, Nov, 1984.

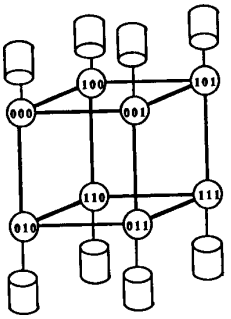

Figure 1. 3-cube with disk attached to each node

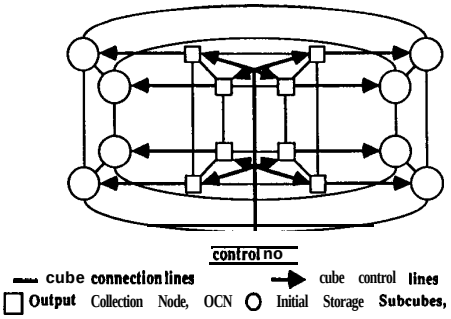

Figure 2. A 64-node system with corresponding control structure

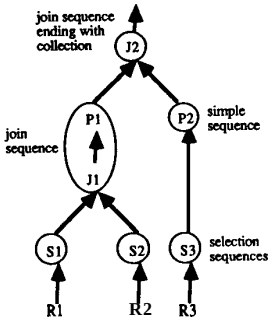

Figure 3. Query tree with operation sequences

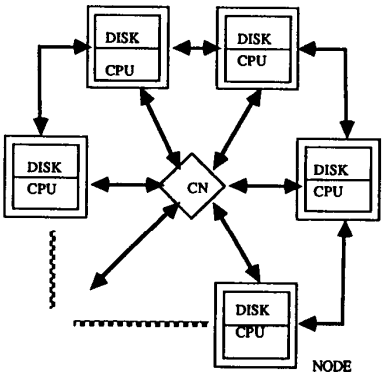

Figure 4. System Architecture Model

Seq#	Seq. Type	Init Time	Input 1	Input 2	Succ	ISS run	Active Query
Q1-1	simple	102	rel 3		Q1-3	1	1
Q1-2	simple	102	rel 9		Q1-3	3	1
Q1-3	join	102	Q1-1	Q1-2	host 1	3	1
Q2-1	simple	125	rel 7		host 6	2	1
Q3-1	simple	913	rel 4		host 5	None	0

Figure 5. A Sequence Directory

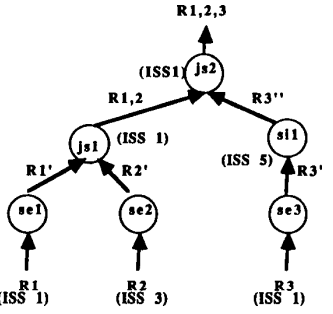

Figure 6. An Operation Sequence Tree

DATA SET		MASC	LASC
Data Set 1	Total time	12571	14075
	avr query	2803	1157
	avr wait	1705	156
Data Set 2	Total time	6346	7561
	avr query	2551	1070
	avr wait	1596	160
Data Set 3	Total time	14840	16501
	avr query	3241	1289
	avr wait	1961	154
Data Set 4	Total time	6520	8930
	avr query	2330	1000
	avr wait	1442	148
Data Set 5	Total time	4957	7031
	avr query	2298	995
	avr wait	1524	149
Data Set 6	Total time	6956	7909
	avr query	2497	1010
	avr wait	1638	138

All times are in milliseconds

Figure 7. Query scheduling policy simulation results