Parallel Algorithms and Their Implementation in MICRONET*

Stanley Y. W. Su
Krishna P. Mikkilineni
Database Systems Research and Development Center
Department of Computer and Information Sciences
University of Florida

Abstract

This paper describes a simple microcomputer network system and its architectural support for four categories of database operations. The design and implementation of hardware and software and the parallel algorithms for the database operations are described and illustrated. Three new algorithms, one for finding maximum/minimum, and two for sorting distributed files, are presented together with their implementations in MICRONET. The results of the analyses of the new sorting algorithms and a comparison with other sorting algorithms are also given. The system is characterized by its simplicity in network connection and communication, flexibility in expanding or contracting the size of the network, reliability achieved by interchangeable hardware and software, and high performance achieved by one-to-all broadcasting, hardware scheduling, and special control lines for inter-processor communication and synchronization.

1. Introduction

The continuous decrease in hardware cost and the idea of a high-performance computer system tailored for database applications have motivated many researchers to investigate many types of "database machines", which are surveyed in [SMI79, SU79, HS180, EPS80, SON81]. Many systems take advantage of the availability and low cost of microcomputers to interconnect these computers into networks which provide the distributed and parallel processing capabilities needed for handling large databases [MAD75, SU78, LIP77, DEW79, BAN79, WAH80, GAR80, HS181]. Due to the difference in architectural designs and special hardware facilities available in the existing systems, the same software algorithms may be implemented quite differently from system to system. The design of various parallel algorithms for database operations in some of these systems has been presented in [SU79, BOR80, HS180a, VAL82, MAW81], but algorithms proposed in one system may not be optimal to implement in other systems. Thus, new algorithms may have to be specially designed to suit a particular architecture and the hardware may have to be tailored to support a specific algorithm. The interaction and integration of hardware and algorithm designs are of paramount importance to achieve the needed efficiency for handling database problems.

This paper deals with the use of a simple and flexible microcomputer network (MICRONET) for the implementation of four categories of algorithms useful for database management. It describes the architectural supports for the implementation of the relational algebraic operators and the hardware and software algorithms for handling aggregate functions such as maximum/minimum, sum and count, and the sorting of distributed files. The present system, whose hardware implementation has been completed, differs from the earlier version of MICRONET [SU78] in the following ways: 1) There is no single dedicated control computer in the system; all microcomputers in the network can become the control computer to oversee the execution of a database command, 2) the system now operates in two modes (global and local) which allow the network to perform as a MIMD machine rather than a SIMD machine, and 3) the hardware facilities such as control lines for system inter-processor communication and synchronization and hardware scheduler for the control of the network has have been designed and implemented to aid the design of software algorithms.

The intended contributions of this paper are as follows: 1) It demonstrates that very simple hardware facilities can be added to a common-bus network system to greatly reduce the amount of message passing and hand-shaking among the processors and simplify the task of synchronizing the concurrent operations of database operations; 2) it presents the techniques for implementing some familiar algorithms, such as those for implementing Selection, Projection, Join, etc., using the hardware facilities; 3) it presents an algorithm for finding the maximum/minimum value of an attribute of a distributed file without having to transfer the values to a specific processor for comparison; and 4) it presents two sorting algorithms (one software and one hardware) for sorting distributed files in

*This research is supported by Department of Energy contract #DE-AS05-81ER10977 and the Center of Excellence, College of Engineering, University of Florida. The computer systems used in this project were provided by the National Science Foundation under grant #NCS 76-21029.
both of which the global sorting of the locally sorted file segments can be accomplished in time close to the time needed to transmit all records to a designated processor. The result of an analysis of five alternative sorting algorithms that can be implemented in bus networks in general and MICRONET in particular is also given.

We present in section 2 the hardware design and implementation of MICRONET and in section 3, the software architecture and data organization. The algorithms for various categories of database operations and their implementation techniques are presented in section 4, which is followed by a conclusion summarizing the features of the system.

2. Architecture and Hardware Design and Implementation

MICRONET consists of a set of microcomputer systems interconnected by a system bus which has 16 data lines and 16 control lines designed to facilitate interprocessor control, communication and synchronization. The "multidrop" bus configuration allows one-to-many communication among the processors (Figure 1). Data, commands, or messages placed on the bus by any processor can be simultaneously received by all the processors. The use of the system bus by different processors to broadcast data, commands, or messages is controlled by a distributed ring register which implements the round-robin scheduling algorithm.

This is implemented by the sender granting circuit shown in Figure 2 which ensures that only one computer is in control of the bus and, consequently, the entire network for the duration of one global operation.

The processor which obtains the bus to broadcast a dataprocessing command (relational operation) becomes the "control computer" (CC) which oversees the execution of the command. All other processors become the "data processors" (DPs) which, together with the control computer, execute the command against their respective local databases. The results of the processing can be either stored distributively in the data processors for further processing or transferred to the control computer for output to the user.

Since the communication is one-to-all in MICRONET, the control computer interrupts all the processors and sends a code word. Each computer in the network reads this code word and will then return to local processing if it is not addressed by the control computer. The decoder shown in Figure 2 selects one of the I/O buffers. The interrupt circuit (Figure 2) is responsible for interrupting the other computers in the network or for acknowledging the interrupt.

The processing of distributed databases in a network system often requires an excessive amount of interprocessor communication and synchronization. In MICRONET, interprocessor communication and synchronization are aided by the control lines. The communication synchronization circuit shown in Figure 2 handles the synchronization be-

tween sending and receiving the data. Two control lines called "sender ready" (SR) and "receiver ready" (RR) are used for this purpose. After sensing that the SR is set, the computer which is ready to receive the data that has been put on the bus receives the data into its buffer and sets its local receiver ready line. The sender computer, after sensing by means of the global RR line (logical-AND of all the local RR's) that all the computers have received the previous word, sends another word on the bus and sets the SR line. Five of the control lines (global lines) are the logical-AND of the local lines. Individual processors set the local lines to report their local conditions. When all the local lines are set, the corresponding global line will be set automatically. The global line can be sensed by all the processors to determine a global condition, which could be the completion of a command, the receipt of a message, etc. Much of the needed hand-shaking and communication protocol found in conventional network systems are, therefore, eliminated.

In MICRONET, the microcomputer systems are connected to the network bus through a set of identical interfaces. The size of the system can be expanded or contracted simply by plugging or unplugging microcomputers to or from the interfaces. Additional hardware flexibility and reliability is achieved by its interchangeable processors and I/O devices. A prototype MICRONET which consists of three PDP 11/03 microcomputers has been implemented. The prototype system provides a proper environment for conducting research in distributed processing and distributed database management. Here, we have presented only a broad outline of MICRONET. A detailed description of this system can be found in [SU78, LEE78, NIC80, NIC81].

3. Software Architecture and Data Organization

The data model used in this system is the relational model. The advantages in adapting the relational data model in single processor systems can be realized in distributed processing systems also. Moreover, the simplicity of data representation in the relational model matches the simplicity of MICRONET. The physical data representation using a modified inverted file structure had been originally considered and presented in [SU78]. Relations are established in the local database based on the natural distribution of data among the microcomputer systems. The locality of data is, therefore, preserved on each network node. However, from a network perspective, these local relations are segments of global relations which are horizontally partitioned and stored in a distributed fashion. In the local mode, the commands of a database query are executed against the local database. In the global mode, each command is broadcast to all the processors. The ones that contain the relevant data files will be operated under global control to carry out the command.
The ones that do not contain the data files will continue their local processing after a short interrupt by the global command.

The users of MICRONET submit queries through the microcomputers. These queries are translated into sequences of commands for execution. In the global mode, the execution of these sequences is interleaved. Thus, a command of a sequence can be executed before the other sequence is completed. However, an interrupt to a sequence can only be recognized at the end of a command execution. In this system, relational operations (commands) are carried out by the data processors in parallel against the local databases. A relational operation may or may not involve interprocessor communication during its execution. For example, a selection operation can be carried out independently by the data processors, where a Join operation would involve transferring relational tuples from one system to another. The processing of the tuples has to be synchronized in the latter case. The local and global control lines described in the preceding section are used for this purpose.

In this system, all microcomputers use identical software. Thus, the software of one system can be reloaded from another system in case of failures. This increases the speed of recovery from system failure and reduces the overall system development cost. All relations in the network are addressed by their names, rather than by the specific processors in which they are stored. Therefore, data are not tied to the processors. The secondary storage devices (e.g., disks) of the microcomputers can be freely interchanged without affecting the computation results. Furthermore, the disks of a failed processor can be mounted on another system (for example, a spare system) and the network would continue to function. The high availability and flexibility of the network system is, therefore, achieved.


In this section, we describe the algorithms for several categories of database operations and the hardware facilities for supporting these algorithms. Performance evaluation and analysis of these algorithms can be found in [GEN81, BRO80, LEE78a, SU82b].

Many parallel algorithms have been developed and analyzed after the advent of the parallel processors [KUN76, VAL75, PRE77], but most of them are tailored for a specific architecture [BOR80, SU79, BAN78, HSI80a]. After a close evaluation of some of these algorithms [KNU73], we decided that some of them are not suitable for adaptation in MICRONET. Thus, new algorithms for finding maximum/minimum and for record sorting are formulated. We shall classify the algorithms we developed into four categories based on the type of data transfer among the processors in MICRONET. They are:

Type 1: Algorithms which can be carried out by the processors independently without data transmission among the processors (e.g., Select, Project (without elimination of duplicates), Delete, Update, and Insert).

Type 2: Algorithms which require the transmission of data values among the processors (e.g., the aggregate functions such as Sum, Maximum/Minimum, Average, and Count).

Type 3: Algorithms for sorting files which require transfer of a large number of tuples from the data processors to the control computer.

Type 4: Algorithms which need the broadcasting of tuples (or records) among the processors (e.g., Intersect, Union, Join, and elimination of duplicates).

4.1 Type 1 Algorithms

The control computer broadcasts the macrocommand to all the data processors including itself. All processors then execute the command concurrently. Each processor will set a specific local acknowledge line after completing the specified operation. When all the processors finish the processing of the macrocommand, the global line which is the logical-AND of all these local lines will be set automatically. After sensing that the global line is set, the control computer releases the control status. All processors then compete to gain control of the bus to process their macrocommands. The setting of local lines and the sensing of the global line in this system achieve the needed interprocessor communication and synchronization and avoid the time-consuming message transfer among processors through the network bus. The operations in this category utilize the parallelism and the property of data distribution in the network to improve the performance of these operations over large files. Once the command is broadcast, all the data processors execute that operation both independently and concurrently. Thus, the execution time complexity of these algorithms is equal to the time for processing the largest of the distributed segments at one processor. The common algorithms used for performing these operations in single processor systems also determine the efficiency of these algorithms in MICRONET. If we assume that the sizes of the individual segments of a file decrease with the increase in the number of processors in the network, the total execution time of these operations also reduces with more processors in the network.

4.2 Type 2 Algorithms

Aggregate functions such as Sum, Maximum, Minimum, Count, and Average are important functions in statistical database applications. They are generally used in conjunction with retrieval operations where a set of records is
MICRONET, we utilize two of the five control
ments of a large file. Thus, the performance
When both Gl and G2 are 0, the processors which
wire-AND of these lines Gl will be set. When all
all the processors are set and consequently the
global operation starts. All processors
simultaneously check the contents of their Rls
and set the two local lines L1 and L2 accord-
bit-by-bit, starting from the most significant
position. The LRV is then compared against the
next less significant bit in their Rls. When both
G1 and G2 are 0, the processors which
had 0 in that bit position will stop participat-
ing in the comparison operations, since some
other processor's Rl obviously has a larger value.
These processors can then return to their local
processing chores. One processor remains com-
paring towards the end signifying that it has the
global maximum value in that processor. The
tuple having the maximum value is then trans-
ferred to the control computer from that proces-
sor. The minimum value can be found likewise by
placing the complements of the actual values in
Rls of the processors. This algorithm thus
avoids the unnecessary communication among pro-
cessors and achieves the maximum amount of paral-
lelism. The above algorithm is similar to that
proposed by Foster [FOS81] which uses a number
of logical wire-OR gates.

The execution time of these algorithms is
the sum of the time for performing the local
maximum/minimum, the time for finding the global
maximum out of these local maximal values, and
the time for transferring the tuple having the
global maximal attribute value. The efficiency
of the above algorithm is quite apparent when we
see that the worst case execution time depends
only on the maximum number of bits used to repre-
sent the attribute and not on the number of pro-
cessors connected to MICRONET. This is because
the global wire-AND control lines in MICRONET
are realized by using open collector-AND gates.

4.3 Type 3 Algorithms

Sorting is one of the important operations
performed frequently in data processing appli-
cations. Other database operations, such as Join, Intersect, Elimination of Duplicates, etc.,
also benefit from having relations sorted in
order. The efficiency of a sorting algorithm is,
therefore, very important.

In our design of the sorting algorithm for
MICRONET, we feel that it is important to avoid
the transfer of tuples (records) over the net-
work as much as possible during the sorting pro-
cess. This is because the data transfer ties up
the most important network resource, i.e., the
network bus, and the transfer of large records
can be very time-consuming. For this reason, we
rejected several existing sorting algorithms
which require that tuples be transferred to some
specific processor(s) for sorting by merging.
For MICRONET, we have investigated a software
sorting algorithm and a hardware algorithm using
a special function processor to perform the sort-
ing operation. They are described below.

4.3.1 Key Broadcasting Algorithm

All processors first perform the local sort-
ing of their local segments of a relation and
obtain an array of the sort key values (Sort
Array) before the start of the global sorting
operation. Figure 4.1 shows some locally sorted
arrays. The processors will then broadcast their
first key values to the other processors as shown
in Figure 4.2. Each processor will compare the
received key value to find the lowest received
value, LRV. The LRV is then compared against the
first key value in the sort array to see if it has
the lowest value in the global sort order.
Here, we assume that the sorting is in ascending
order. The computer which has the lowest value
will compare the subsequent key values in its
array until it finds the one greater than the
LRV. Then that processor will send the key value
(the one that is greater) to all the other pro-
cessors and the block of tuples which corre-
spond to the smaller values in the sort array
(which are in global sort order) to the control
computer (see Figure 4.3). While the tuples are
being transferred, the other computers will have
completed the comparison process and one of those computers will be ready to send another block of tuples. This process continues until the control computer has received all the tuples in global sort order. The three main features of this algorithm are: (1) blocks of tuples which are in global sort order are transferred on the bus instead of one tuple at a time, thus reducing the time required for the transfer and the frequency of interrupts to the processors; (2) the comparison process for sorting is overlapped to a large extent with the secondary storage I/O and the tuple transfer and, thereby, reducing the effective sorting time considerably; and (3) during the sorting process, only the key values are transferred among the processors; the tuples are not transferred unnecessarily, but are transferred to the control computer only if they are already in the global sort order.

4.3.2 Hardware Algorithm

The hardware algorithm for sorting makes use of special hardware we designed for this purpose. Figure 1 illustrates the way this special processor is connected to MICRONET. All the processors first perform the local sort operation like in the software algorithm and then transfer the first key values in their sort arrays (Figure 4.1) to the special processor. The special processor contains a set of shift registers, each one corresponding to a computer in the network. The registers are used to hold the key values transferred to the functional processor (Special Processor). By shifting the higher order bits to the left and testing the higher order bit values, the maximum of all the key values in all the shift registers can be found.

A number of control lines are used to connect the special processor with each computer (Figure 1). They are used to synchronize the starting of the sorting operation and to enable the loading of the key values into shift registers.

When the special processor finds the maximum, it will notify the particular computer which has sent that value. That computer then sends the next value in its sort array to its shift register in the functional processor and then sends the selected tuple to the control computer. Since the comparison for the maximal value can be done by the special processor during the time when the tuple is being transferred, the special processor will be ready to notify this or another computer which will transfer the next tuple in global sort order.

Although the functional processor is designed to find the maximal value for sorting in descending order, sorting in ascending order can be done by transferring the complements of the field values to the processor. Also, the aggregate functions maximum and minimum can be handled by this processor. In this algorithm, sorting time is completely overlapped with the tuple transfer time. The additional advantage of this algorithm is that the same piece of hardware can be used to calculate the global maximum (also minimum) efficiently. Furthermore, we have eliminated the unnecessary transfer of tuples as in the software algorithm by transferring to the control computer only the tuples that are known to be in the global sort order.

We have conducted a thorough analysis of both the software and hardware sorting algorithms. We have compared the performance of these algorithms against that of some other algorithms that one can adapt to perform sorting on common-bus networks. Figure 5 illustrates the behavior of these different algorithms with increasing file sizes. $T_1$ is the total execution time of the sorting algorithm, where all the segments of the file from all the data computers are transferred to the control computer, which then sorts the combined file segments using an external sorting method. $T_2$ is the execution time of the algorithm, where all the segments are initially sorted locally at the corresponding data computers. The data computers then transfer, one after the other, their logical segments to the control computer to be merged with the resultant segment of the previous merging operations. The control computer will have the final sorted file when the segment from the last data computer is merged completely. $T_3$ is the execution time of the algorithm which also initially sorts locally all the distributed segments. Each data computer then transfers a smaller block of its sorted segment of the file to the control computer which merges these blocks into a single list. The final globally sorted file is formed when all these blocks from all the computers are merged completely. $T_4$ and $T_5$ are the execution times of the key broadcasting and hardware sorting methods, respectively.

Our results show that both the above algorithms perform much better than the others under different conditions created by varying the number of processors, the bus speeds, the interrupt times, the file sizes, and the I/O times. The key broadcasting algorithm is very close in its performance to that of the hardware algorithm presented in section 4.3.2, leading us to conclude that the extra cost and time involved in designing hardware are not worthwhile, when we have a simple software algorithm designed to suit a simple architecture. The execution times of both the above algorithms decrease with the increase in the number of processors. The other observation we made from our results is that the slower network bus bandwidths deteriorate the performance very quickly, whereas the higher bandwidths do not add to the performance benefits very much. Therefore, high-performance networks do not seem to provide the expected improvements in the efficiency of the algorithms for parallel-sorting, in proportion to their added cost and complexity. A complete study of these algorithms and their performances can be found in [SU82b].

4.4 Type 4 Algorithms

Statistical analysis of data often involves correlating large quantities of data across sever-
al files. The efficiency of relating files by operations such as Join, Intersect, Union, etc., can often determine the overall efficiency of a system for database applications. In MICRONET, the control computer initializes all the control lines and sends the command to all the data processors specifying what are the two relations to be Joined, Intersected, or Union-ed and which one of the two relations (i.e., the relation with fewer tuples) to be broadcast over the network. All the data processors including the control computer which have the segments of the smaller of the two relations will compete for the network bus and interrupt the other computers to broadcast a block of tuples to all the processors. All the processors will then simultaneously process the local segments of the large relation against the received block of tuples. This process continues until all the blocks of the smaller relation have been broadcast and processed in the network. Figure 6 illustrates the global Join operation with three computers in the network as an example of this type of algorithms. This method of transferring tuples in blocks for processing this type of operations against distributed relational segments is not new. It has been used for the Join operation in the early version of MICRONET [SU78] and in DIRECT [DE79]. However, in our system, the hardware is tailored to support the software algorithms. For this category of operations, a number of control lines, i.e., the five global lines (GA0-GA4) and their associated local lines (LO-L4), are used to reduce the amount of message transfers among processors and to speed up the synchronization of sub-processes required in the operations.

To illustrate the use of these control lines, we shall use the global Join as an example. For the Join operation, all five global lines (GA0-GA4) and their associated local lines (LO-L4) are used. LO, when set, indicates the control status of the processor. L1 indicates that the received block of tuples of the smaller relation (say relation A) has been joined with the local segment of the large relation (relation B). L2 signals that either the processor does not have a segment of the A relation in its local memory or the entire segment has already been transferred. L3 signals that the processor does not have a segment of the B relation. L4 indicates that the last block of relation A tuples has been broadcast and processed by all the processors. It is worthwhile here to state again that the setting of all the local lines will automatically cause the setting of the corresponding global lines. Since only one local line is set to indicate that some processor is in control of the bus and is processing the macrocommand, GA0 is always zero. GA1, when set, signals that the Join of the received tuple block of relation A with relation B has been completed by all the processors. GA2 indicates that all processors have transferred their segments of relation A. (At this point, the last segment transferred still needs to be merged.) When GA2 is set and the processor has completed the Join of the last tuple received, its L4 will be set. GA3 is not used in this operation since L3s are used by the processors for checking local conditions about relation B. GA4, when set, indicates that the last processor has processed the last block of tuples received and therefore the global Join operation is completed.

The flowchart shown in Figure 7 illustrates the subprocesses of the global Join operation. It also shows how the local and global control lines are used during the Join operation. By setting the local lines and sensing the global lines, operations in various processors can easily be synchronized. The intensive message broadcasts and acknowledgements which are commonly seen in implementing these algorithms in the existing distributed systems are eliminated. This is important because the performance analysis of this Join algorithm [BR80] shows that the execution time of these kinds of algorithms is primarily I/O bound when the relations are small and becomes network transmission bound when their relation sizes and the network sizes increase.

An alternate approach for performing the Join operation in MICRONET is to sort both the A and B relations locally on all the processors concurrently. Each processor then sends the lowest and highest key values and the number of elements in its sort array to the control computer. The control computer determines the range of the key values whose corresponding A and B tuples need to be collected in a particular processor and broadcasts that data to that processor. Then, while performing the global sorting operation on B (the larger relation) using the key broadcasting algorithm described in section 4.3.1, the computer which has the lowest key value sends the block of B tuples in the global sorting order to the particular processor which has been determined earlier to receive that range of tuples. At the end of the global sorting operation, each processor has the tuples that are in the global sorting order and that fall in the assigned range of tuples to that processor. Each processor then sends its A tuples to the processor which has the corresponding range of B tuples. After all the processors complete transferring the A tuples, all the processors merge their segments of A and B relations simultaneously.

The advantages of this approach are 1) local sorting of both relations are performed concurrently on all the processors, 2) global sorting of the larger (B) relation is performed by modifying the sorting algorithm proposed in this paper which proves to be faster than the other methods for global sorting, and 3) the result of the Join operation is distributed on all the processors in the global sorting order which is important for subsequent processing of the resulting relation. However, when either both the relations are small or very different in size,
the extra time for sorting may be large compared to the simple transfer and search operations in the nested loop method, and thus the first method may be more advantageous in such situations.

We can utilize the key broadcasting algorithm (section 4.3.1) for eliminating the duplicate tuples in a global relation also. Elimination of duplicates results as a by-product of the sorting algorithm because the situation where a set of duplicate records is distributed over the network can be easily identified by all the processors. Therefore, all except one processor refrain from transferring the corresponding duplicate tuples to the control computer, thus eliminating the duplicates automatically. Therefore, the execution time for the elimination of duplicates in MICRONET is almost the same as that of global sorting.

5. Conclusion

In this paper, we have presented the hardware and software architectures of MICRONET and the distributed algorithms for four categories of database operations. We described the hardware facilities available in MICRONET for supporting these software algorithms. We have also given the performance improvements we can achieve by using these new approaches (both in hardware and software) in solving the common problems in database management. Several features of MICRONET which are suitable for database applications should be stressed. First, the system is highly flexible. As a database grows in size, additional microcomputers can be easily added to the network by plugging the interface cables to the backplanes of the microcomputers. An organization can use any number (22) of microcomputers to form a network system to suit its needs. Second, the system is very reliable because of its simple network structure and its identical hardware (network interfaces and microcomputers) and software. A failed interface or microcomputer can be easily replaced by a spare, thus increasing the speed of system recovery. Also, since data files are addressed by their names rather than by the processors in which they reside, the disk packs of a failed system can be moved to another system without affecting the computation results. Third, the system is very efficient due to its one-to-all broadcasting, its interprocessor communication and synchronization using special-purpose control lines and its parallel processing capabilities. Lastly, perhaps the most important feature, the system is extremely simple and inexpensive to implement, in contrast to many other multiprocessor database machines.

The entire network interface for the present three-node system was built at the cost of approximately $300. Since the network interfaces for the computer are identical, a mass production of these interfaces will drive the cost very low. The simple network also provides an excellent environment for conducting research in distributed processing and distributed database management areas.

References


Figure 1. System Block Diagram
Figure 2. Block Diagram of the Interface
If BIT=1; LO is set
L1 is reset
If BIT=0; LO is reset
L1 is set

LOCAL MAXIMUM
OF

\begin{tabular}{c|c|c|c}
\hline
PROCESSOR & BIT & LO & L1 \\
\hline
1 & 1 & 1 & 0 \\
2 & 1 & 1 & 0 \\
3 & 1 & 1 & 0 \\
\hline
\end{tabular}

Result: All Processors (P1, P2, and P3) proceed to test next bit.

COMPARISON 2

\begin{tabular}{c|c|c|c}
\hline
BIT & VALUE & LO & L1 \\
\hline
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\hline
\end{tabular}

Result: P2 returns to local processing; P1 and P3 proceed to test Bit 5.

COMPARISON 3

\begin{tabular}{c|c|c|c}
\hline
BIT & VALUE & LO & L1 \\
\hline
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 \\
\hline
\end{tabular}

Result: P1 and P3 proceed to test Bit 4.

COMPARISON 4

\begin{tabular}{c|c|c|c}
\hline
BIT & VALUE & LO & L1 \\
\hline
1 & 1 & 1 & 0 \\
X & 1 & 1 & 1 \\
0 & 0 & 0 & 1 \\
\hline
\end{tabular}

Result: P3 returns to local processing; P1 has Global Maximum.

Figure 3 A Method for Finding Global Maximum/Minimum.
Figure 4.1 Initial State of Global Sorting Operation.

SEND TUPLES CORRESPONDING TO 10, 13, 15, 18, 20, 23 TO CC AND SEND 30 TO ALL NODES.

Figure 4.2 PASS 1 of Global Sorting Operation.

Figure 4.3 PASS 2 of Global Sorting Operation.
Number of Records in the Global File with Number of Processors = 50, Interrupt Time = 100 us, Number of Records in a Page = 100, Record Size = 250 bytes, Page Write Time = 15 ms, Network Bus Speed = 2.5 Mb/sec

Figure 5. Execution Time vs. Size of File.
Global Join

A * B

IAI < IBI

1. DP_1 Transmits First Block of its A RELATION to all Other Machines for External Join.
   \[ A_{DP1} * B_{CC} \]

2. DP_2 Transmits its A RELATION to all Other Machines
   \[ A_{DP2} * B_{CC} \]

3. DP_1 Transmits its Last Block of A RELATION
   \[ A_{DP1} * B_{DP1} \]

4. CC Transmits its A RELATION and Waits for End of Global Operation
   \[ A_{CC} * B_{CC} \]

5. End of Global Operation

Figure 6. Global Join Operation
Global Join

Broadcast "Prepare to Join" Command

Set/Reset L(0)...L(4) accordingly

1 (No A)
0 (Move A)

L(2)

Perform Join on Local Segment B with Received Block of A Tuples

Wait for G(2) Set

Perform Join on Local Segment B with Transmitted Block of A Tuples

Set L(4)

Return to Local Processing

Release Control Status

End

Interrupt Service Routine

"Data Processor"

Get Control of Bus

Yes

No

Broadcast "Join" Command with Block of Tuples to Others

If Final Block of A, Set L(2)

Perform Join on Local Segment B with Received Block of A Tuples

L(0)=1: Control Computer Status
L(1)=1: Received Tuples Have Been Joined
L(2)=1: The Local Segment of A has been Completely Transferred
L(3)=1: There is No Local Segment of B
L(4)=1: The Last Block of Global A Relation has been Processed

Figure 7. The Global Join Operation