93-TT

A Transputer T9000 Family Based Architecture for Parallel Database Machines

Qiang Li*
Dept. of Computer Engineering
Santa Clara University
Santa Clara, CA 95053
(qli@scuacc.scu.edu)

Naphtali Rishe
School of Computer Science
Florida International University
Miami, FL 33199
(rishen@fiu.edu)

Abstract

Parallel computing is a promising way to achieve high performance in a database system. The disk access speed has been a well known bottleneck for database machines, and the data intensive nature and the random communication patterns of databases make the interconnection network in a database machine difficult to design.

This article describes the design of a highly parallel, high throughput database machine based on the new T9000 transputer family and a large number of relatively inexpensive disks. The database machine is designed to achieve a throughput of more than 10,000 elementary transactions per second and a total size in the range of terabytes. The Semantic Binary Database Model is employed by the system. Besides its logical aspects, this model's implementation facilitates the utilization of the parallel processing power and improves the reliability of the system.

Keywords: Parallel database machine, Transputers, semantic model, interconnection network, disk array.

1 Introduction

The demand for high performance database systems increases rapidly. It is not surprise today to see the need for a system with a throughput of tens of thousands transactions per second and a size of terabytes. Such a system can be used by credit card companies, for telephone system control, satellite imagery, and in a massive public information retrieval center. Much effort has been made to build database machines [15, 7, 8, 5, 9, 14].

Our design aims at a large scale database machine with terabytes of storage space and a throughput exceeding 10,000 simple transactions per second (a good throughput when used at one hour before a $100 million lotto drawing.) The intention is to build a sound hardware

*This research has been supported in part by an IBM Research Award grant

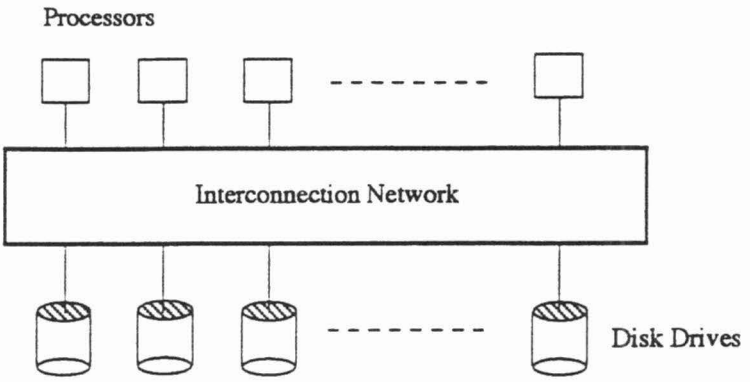

Figure 1: Processor-disk connection: option 1

platform for a large scale database system with a good blend of computing power, I/O power and interprocessor communication.

The speed of a single disk drive is limited by the available technology, and there is not much room for the system designers. To solve the I/O speed problem, one has to use a relatively large number of disk drives so that unrelated data can be accessed and transferred in parallel. This approach has been adopted by many researchers [4, 16, 10]. The computing power required by a database system, especially an intelligent database system, can be obtained by using a large number of processors. The problem is how to connect a processor farm with a disk farm so the resources can be efficiently utilized. One approach is to line up the processors at one end and the disk drives on the other end, and put an interconnection network in the middle (Figure 1).

This is based on a model where every processor has equal access to every disk drive. The data transferred in the network is the unfiltered raw data and every piece of data transferred from or to the disk drives must pass the network. The load of the network is therefore very heavy. Since the network is always the most difficult and most expensive component of any large scale parallel machine, this approach seems to exacerbate the burden.

The approach that we are using here is to move the processors close to the disks by attaching a processor to each disk drive to form a processor-disk unit. The processor should be powerful enough to deal with the basic operations on one disk drive. Many such processor-disk units are connected by an interconnection network as shown in Figure 2. Any request to data operations will be processed by the processor which is connected to the disk drive holding the relevant data. Only those requests which require data from several processor-disk units will cause data being transferred through the network, and the data is in most cases filtered. Therefore, only a small portion of the I/O data flows through the interconnection network.

Using powerful processors to deal with a disk drive can also be justified by the existence of high degree of medium to large grain parallelism in large scale database systems. For example, for a system requiring a high throughput of small transactions, the degree of parallelism at

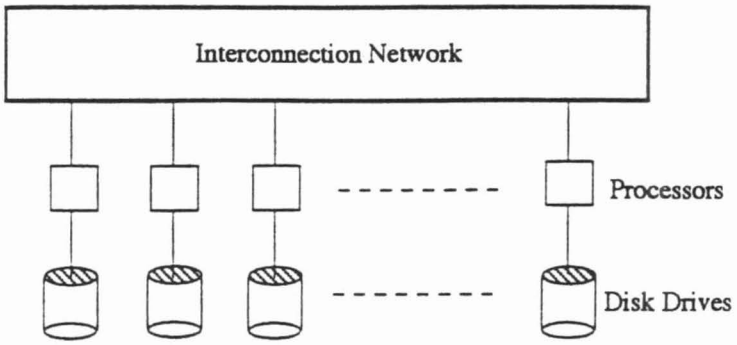

Figure 2: Processor-disk connection: option 2

transaction level can be a few hundreds to a few thousands. Or, when an exhaustive search is needed, the search can be done at all processor-disk units in parallel.

The network and the processor-disk units can be constructed in many different ways. Our design presented here utilized the latest development in Transputer systems [3] which is described briefly below.

2 The T9000 Transputer family

The T9000 transputer is the latest development by INMOS. Although its birth has been delayed, the principle behind the design is sound. The T9000 processor is based on the philosophy of balancing computing power and interprocessor communication power. The T9000 processor supports 32 bit integer operations and 64 bit floating point operations. The processor has a peak performance of 200 Mips and 25 Mflops, the fastest single chip processor ever been built. The price is about $2/Mips.

Each T9000 processor has 4 100 Mbits/second full-duplex serial links each with its own pair of DMA channels. Two T9000 processors can be directly connected by the links without external buffers and other supporting circuit. Collectively, the 4 links give a T9000 processor an interprocessor communication power of 400 Mbits/second. Hardware multiplexers are used to map virtual channels to physical channels to facilitate inter-process communication across processors.

To support more complicated interconnections, INMOS also provides communication chip, namely C104, which is a 32 x 32 crossbar switch. A pair of communication channels is fully compatible to the links of T9000. All the connections in C104 operate independently. The connection pattern of C104 is similar to its ancestor C004, but it has a very important improvement besides the communication speed. C104 supports *wormhole routing* [6, 3].

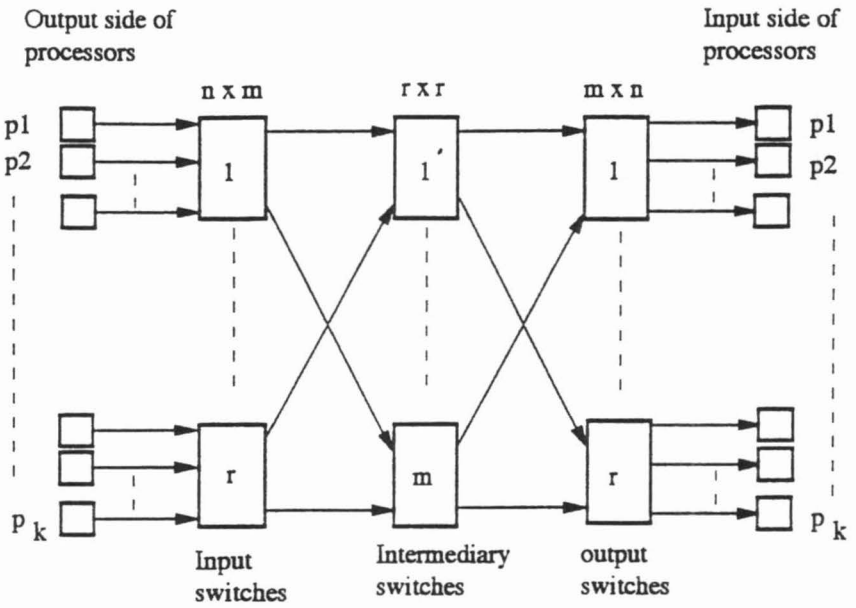

Figure 3: The three-stage Clos network

3 The Proposed Architecture

Our database machine design has the structure shown in Figure 2. Although different configurations are possible depending on the application, the model with 1024 processor-disk units will be used for this discussion.

3.1 The Processor-Disk Units

Each processor-disk unit consists of a T9000 processor, a disk drive and a large memory bank. The size and type of the disk drive depend on the application and the configuration of the system, such as the size of the database, the number of processor-disk units, and the type of data access. It can be, say, a 1 Gbyte hard drive, or an optical disk drive.

A large portion of the memory associated with a processor serves as the cache memory of the disk drive. Again, depending on the configuration, it can be a few Mbytes to a larger size, say, 20 Mbytes.

One of the T9000's communication links can be used to connect to the disk controller. The link speed is high enough to cope with the fastest data transfer rate of disk drives.

T9000 supports multiprogramming very well. Processes can be easily created and the context switch is extremely fast. T9000 has hardware support to map virtual channels onto physical channels. Thus, a process can communicate with another process running on a different processor. No software layer is used. This makes the interprocess communication across processors convenient and efficient. This feature facilitates the implementation of multiple databases on the same hardware platform, each being served by their own processes running on many processors.

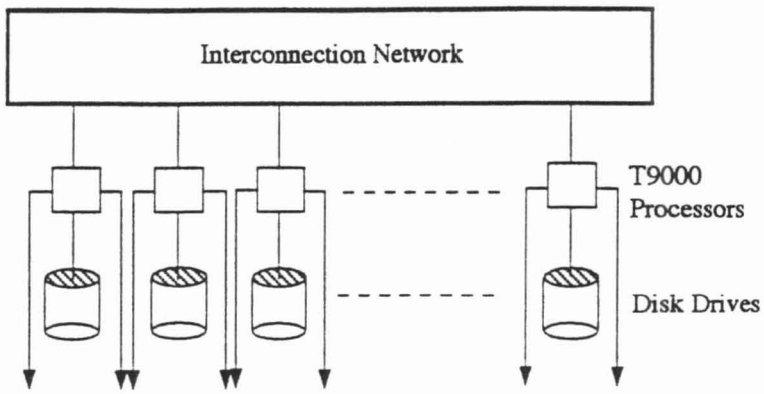

Figure 4: The system connections

3.2 The Interconnection Network

The interconnection network is a vital part of the proposed database machine. It must have high bandwidth and low latency. Many different types of networks have been proposed and studied, and a good collection can be found in [17]. The main issue is the trade-off between cost and performance. The C104 chip provides a sound building block for constructing a powerful network. In our design, 96 C104's are connected into a three-stage Clos network [2, 1]. Figure 3 shows the structure of such network. Each box in the figure is in general an $n \times m$ crossbar switching component (32×32 in our case.) There are three columns of switches: the input switches, the output switches, and the intermediary switches. The links between the switches are uni-directional. The leftmost column is the output side of the data processors; the rightmost column is the input side of the data processors. For convenience,

The three-stage Clos network has been selected because it can simultaneously connect many processors without too many layers of switching components which could result in a longer propagation time. A total of 1024 processor-disk units can be connected by the 96 switch chips. When a larger system is needed, the chips have to be cascaded to form a larger network.

The connections between the processors and the input switches are static, and so are the connections from the output switches to the processors. Thus, when processor A needs to send a packet to processor B, the input and output switches are predetermined, but any of the intermediary switches can be used to set up the circuit.

Since the data packets are always sent from left to right and there is no need for packet forwarding by processors, the network is an acyclic network, i.e. there cannot be a deadlock in the network, even though it allows every processor to communicate to any other processor.

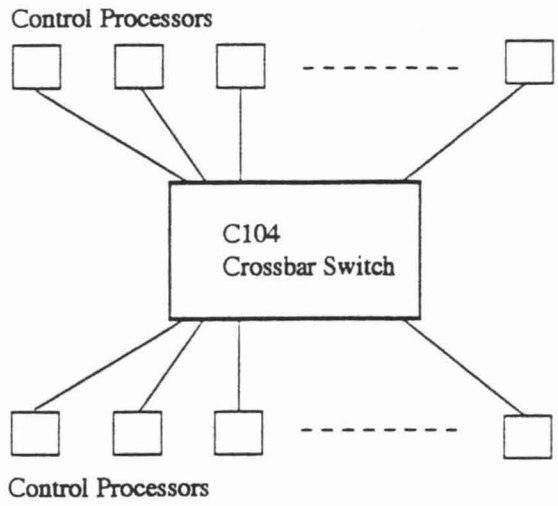

Figure 5: The control network

3.3 The System Connections

Figure 4 shows the system configuration. In general, each T9000 processor has one link connected to the network and one connected to the disk controller; two links remain free. The two free links can be used for purposes discussed later.

Any number of processors can be designated as the interface processors to the host machines utilizing one of the two free links. Since a large number of interface processors can be used, the system can have very convenient connections to the outside would. The model under our consideration has 32 interface processors. The interface nodes normally need to communicate with each other frequently since the they have to exchange control information such as some form of data directory, protection and security check. Therefore, an additional network is used to connect the interface processors. The network is simply one C104 chip as shown in Figure 5 which can dynamically set up a circuit between any two interface processors. To distinguish the additional network from the primary one, we call the former the control network and latter the global network. Thus, an interface processor has the choice of communicating with another interface processor through the control network or through the global network.

All the processors other than the interface processors are linked into a ring. This is important in three aspects.

First, in contrast to the global network where connection between any two nodes are treated equally, we now introduced local connections, i.e., dedicated links between processors. Each processor now have neighbors. The processes on each processor can be classified into two layers, the data layer and the service layer. The data layers of two adjacent processors are connected through a virtual channel supported by the T9000 hardware, and so are the service layers as shown in Figure 6. Since the the communication between the neighboring

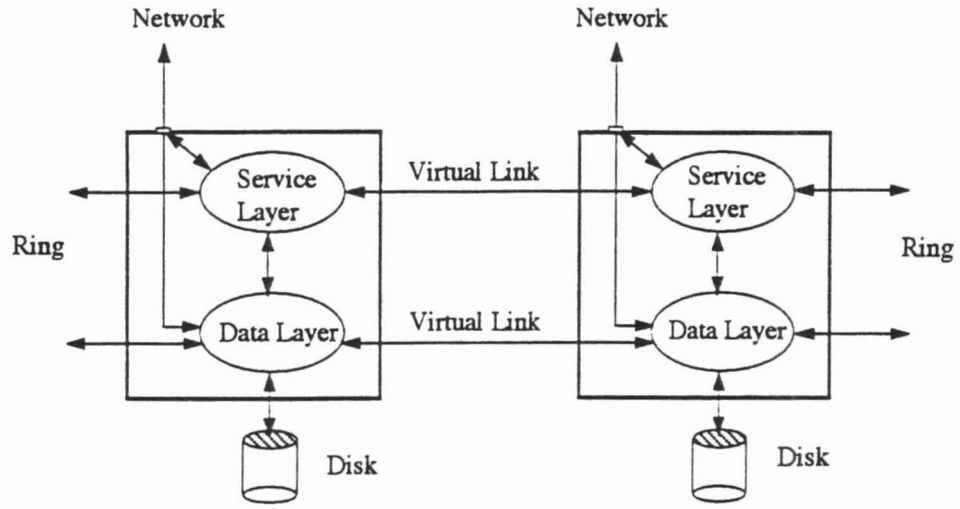

Figure 6: The layers and their connections

processors is not intensive, each virtual channel can have an effective full-duplex bandwidth of near 20 Mbyte/s.

The data layer is responsible for the data I/O, transfer, and basic data filtering. The service layer is responsible for sophisticated computing tasks. Data retrieving and storage requests coming from the network will be answered by the data layer while the transactions are handled by the service layer. The service layers of a few nearby processors in the ring can cooperate to solve complicated computing intensive requests utilizing the smaller grain parallelism. When a predefined parallelizable task arrives (dynamic parallelism recognition is still an very open problem), the receiving processor will partition the task and send service request and data to the neighbors for processing. The protocol between the service layers on the neighboring processors totally depends on the specific task and application. Further research on this topic is being undertaken.

Secondly, the local connection between processors can help reducing the impact of "hot spot". When a communication hot spot has developed, the data can be relayed by the neighbors of the "hot processor" utilizing the dedicated channels between them. Although a simulation has shown that the network in question is not very sensitive to the presence of hot spot under the designed load, the communication hot spot detection and avoidance is still an ongoing research in order to achieve higher resource utilization.

The third aspect is related to the reliability of the system. Let's study the Clos network closely. A path from input to the output can use any of the intermediary switches. Therefore, a failure of the intermediary switch is not that critical. However, when an input switch or an output switch fails, all the processors connected to the switch are isolated, and the entire system will most probably have to be stopped before the situation can be rectified. To increase the reliability of the system, the connection of the ring is designed carefully to ensure that two immediate neighbors in the ring are not connected to the same input switch or output

switch of the Clos network. In this way, if one of the input switches or the output switches fails, the data can be passed to a neighbor and be relayed to the intended destination. The system performance will degrade somewhat, but the system can be kept alive while the faulty part is being fixed or until a planned shut down time. If we number the processors connected to a switch of the Clos network from 0 to 31, and the switches in a column from 0 to 31, and we name each node on the input side of the Clos network by a pair of numbers: (node number, switch number), then the ring can be connected in the following sequence:
(0,1) (0,2) (0,3) .. (0,31) (1,1) (1,2) (1,3) .. (1,31) .. (31,1) (31.2) .. (31,31) (0,1)
It is not difficult to verify that all immediate neighbors are connected to different switches.

Although the addition of the ring seems to have clouded the clean direct access from processor to processor, its existence can be made transparent by the network software layer. When a message is sent, whether or not the ring network is used is decided by the routing software according to the availability of the channels and load of the channels.

4 The database model and its implementation

In choosing the database model we required two properties:

1. It should be a model that reflects the state-of-art of database.
2. Its implementation should facilitate parallel processing.

The database model chosen for the proposed database computer is the Semantic Binary Model [11]. The semantic binary database model represents information of an application's world as a collection of elementary facts of two types: unary facts categorizing objects of the real world and binary facts establishing relationships of various kinds between pairs of objects. The purpose of the model is to provide a simple natural data-independent flexible and non-redundant specification of information emphasizing its semantic aspects.

In our implementation [12, 13], the basic entity of the database is the binary relation between two data items in the form:

$$\text{Item1} \ \langle \text{rel} \rangle \ \text{Item2}$$

Its reversed form, which in some, but not all, cases is

$$\text{Item2} \ \langle \text{rel} \rangle^{-1} \ \text{Item1}$$

is also stored in the database as implicit indexing information. The entire database, including all of the implicit indexing information, is represented by one logical coherent file partitioned into segments of sufficiently small size, so that they may be stored on small disks. The number of processor-disk formations is sufficient to accommodate the totality of segments. Since a relation and its reversal are both stored in the database, the implementation duplicates each data item exactly once. The amount of duplication is not as serious as it sounds. Since very little indexing information is needed for the implementation, almost all the index trees existing in the traditional implementations are eliminated. Therefore, the relative overhead

is small [13] More importantly, the duplication of data is not simply making a mirror copy, rather, the duplication is in a different logical format which enables us to gain some significant advantages for a parallel system as discussed below.

4.1 Localization of information

An important property of the implementation is that for atomic operations, the related data can be found in a contiguous segment of the file, which localizes each simple query to one or two particular processor-disk units in most cases. The localization of the information reduces the interprocessor communication load, which is crucial in a large scale multiprocessor system.

4.2 Randomizing the data references

One more interesting result of the implementation is that it serves to randomize the data distribution to a certain extent. Objects are assigned random identifiers. The system sorts all the relationships globally, the logically coherent data items is not necessarily stored in a consecutive disk space. For example, what would be a relational table and stored in a consecutive space may now be scattered across a number of disks. When a logical entity in the user's view is accessed heavily creating a logical hot spot, it does not necessarily create a hot spot at the hardware level. This sounds contradictory to what we claimed earlier that related data is stored in consecutive disk locations. It can be best explained by an example. Suppose, in the relational model we have a table of student records, and the primary key is the social security number. Such a table is normally stored in relational database implementations in a consecutive space, and all of the references to the table have to access the same storage area (same disk.) In our implementation, the table is represented by many pairs of binary relationships, and stored in separated areas. For example, student A's record may be stored far away from student B's record, and references made to them respectively will not compete for the same disk area (disk drives). But a list of student names can still be found in a consecutive disk space owing to the duplicated data.

4.3 Improving system reliability

When a large array of relatively inexpensive disk drives are used, the reliability of the disk array becomes a serious problem. A good analysis of this problem is given by [4]. Since the disks can be divided into two partitions, one for the normal relations and another for the reversed relations, the duplicated data copied can be stored on different disks. Thus when a disk drive fails, the data can be reconstructed from the other disks. Hence the reliability of the database system is improved. Assuming the MTTF (mean time to failure) of a single disk is 30,000 hours which is typical for an inexpensive disk drive, the MTTF for 1000 disk drives is 30 hours, clearly not an acceptable rate. At such a failure frequency, no any other recovery can work very effectively.

With our implementation, the probability of the system failing at any given moment can be calculated as follows.

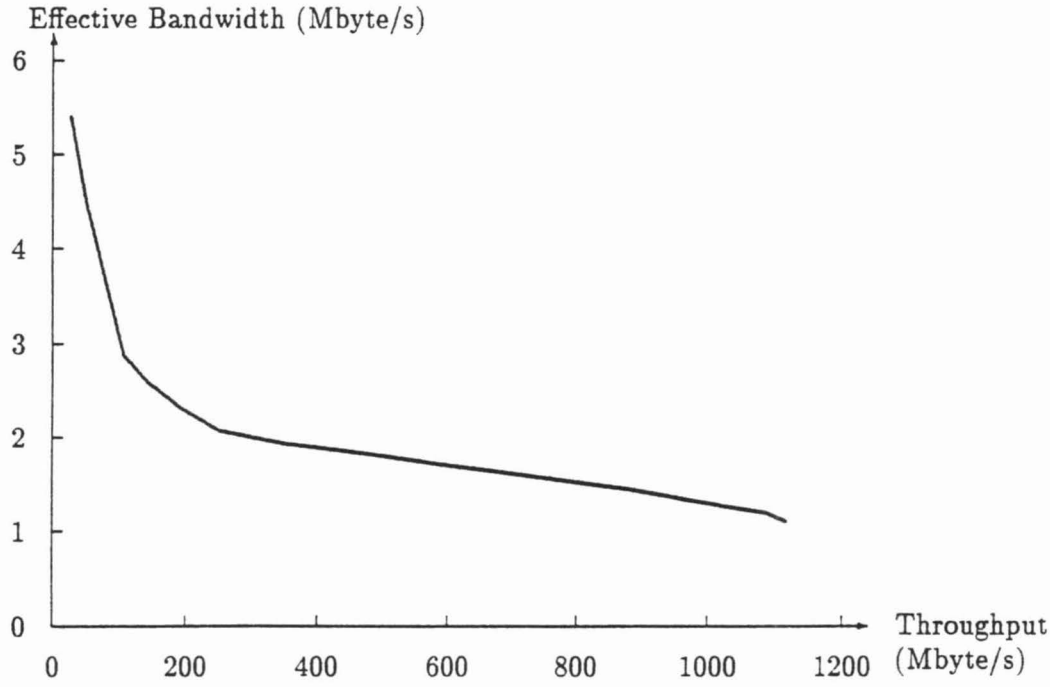

Figure 7: Effective bandwidth vs. communication throughput

We have two independent subsystems, one for the normal relations and another for the reversed relations. Let s be the MTTF of each subsystem. Let $L << s$ be the mean time to repair. Then the probability in any moment in time (in the stable state) that the entire system is down, i.e. both subsytems have failed within L before the current time is:

$$p = \frac{(L/s)^2}{1 + 2 * L/s + (L/s)^2}$$

A simulation has been run to predict the MTTF of the entire system. It is assumed that a pool of backup disks is reserved and one of them can be switched into the system electronically when a disk fails. With the assumption that MTTF of each individual disk is 30,000 hours, and the system has 1024 disks, 512 for each subsystem, and $L = 0.5$ hour, the MTTF for the entire system is about 3420 hours. MTTF calculated by the formula is 3434 hours, closely matching the simulation result.

5 Performance evaluation

A simulation program was implemented to evaluate the performance of the interconnection network. To simplify the simulation, the data size is always assumed to be a multiple of 32 bytes. Figure 7 depicts the average effective bandwidth between any two processors under different system communication throughput. When the system is lightly loaded, the effective bandwidth can be as high as 5 Mbyte/s. When the system throughput is between 200 Mbyte/s to 1 Gbyte/s, the effective bandwidth varies between 2 to 1 Mbyte/s.

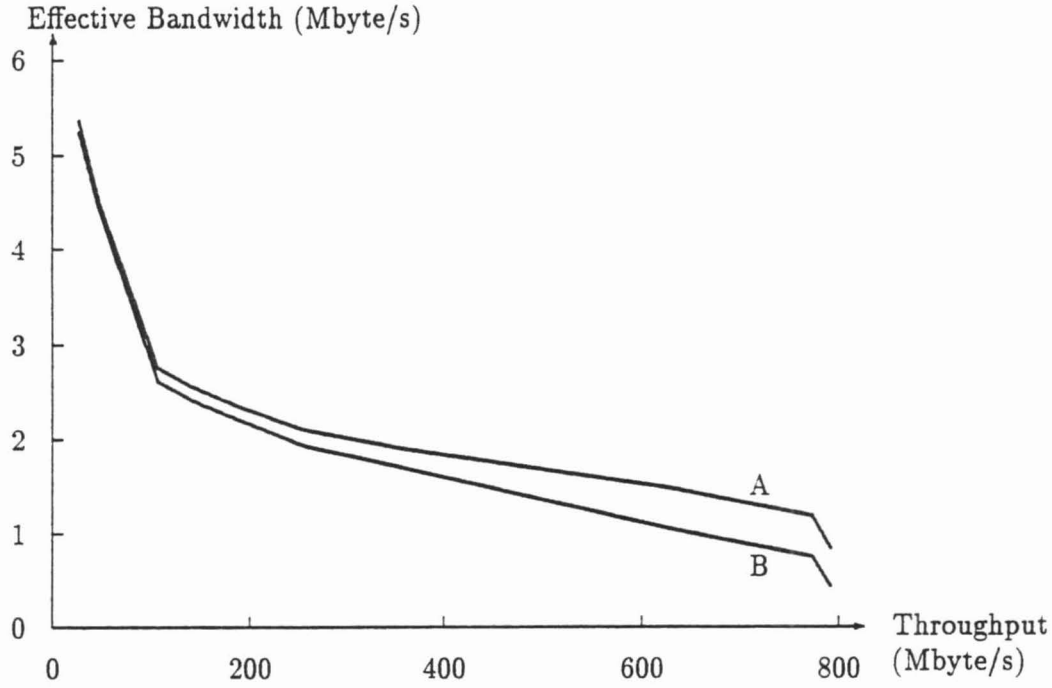

Figure 8: Effective bandwidth vs. communication throughput with hot spot

Assuming on the average that each simple query requires 10 Kbyte data transfer between processors (since most of the data can be processed locally), then the required communication throughput is 100 Mbyte/s with a system throughput of 10,000 simple queries per second. According to Figure 7, the effective bandwidth is about 2.5 Mbyte/s. Thus, assuming all the communication is done serially which is the worst case, the total latency caused by the interconnection network is $10000/(2.5 \times 10^6) = 4ms$, which is small comparing to other time factors involved in solving a query.

Figure 7 shows that the interconnection network can have a good effective bandwidth even when the throughput exceeds 1 Gbyte/s, i.e. there is a large communication reserve. This is desirable for reducing the impact of bursts of large data size and presence of hot spots.

Figure 8 shows the effective bandwidth when a hot spot exists in the system. The hot spot is simulated by letting one processor receive 10 times as many messages as any other processor receives. Curve A is the average bandwidth between any two processors, and Curve B is the average bandwidth between any processor and the "hot" processor. At the throughput range between 100 Mbyte/s and 200 Mbyte/s, the system performance is not sensitive to the presence of the hot spot.

Since the interconnection network has a large capacity reserve, the system throughput depends mainly on the throughput of each processor-disk unit.

Many performance aspects can not be realistically addressed until a relatively large size prototype is built, and specific applications are considered. It is important to note that the collectively 200,000 MIPS of peak raw computing power possessed by the system allows it to do a lot more than just answering queries. It has a great potential to handle the logical

reasoning, and other computation intensive jobs based on the stored data.

6 Other Implementation Issues

It is obvious that data distribution and implementation of software is more complicated on the proposed system. The design is still very sketch. A number of nodes can be designated as the interface to the outside world. Those nodes receive queries and initiate the process of solving queries. There are two steps involved in solving a query.

First, the interface nodes collectively maintain the directories of data allocation. The interface nodes analyze an incoming query and decide which node in the system is most qualified to process the query. The selected node is called the leading node the query. The decision is largely based on data distribution, and also the load of each individual node. Once selected, the interface node sends the query together with the information about data allocation to the leading node.

Based on the information it receives, the leading node of a query acts as a master, and decompose the query into subqueries or data requests. The subqueries and data requests are then sent to other slave nodes. The final result is assembled at the leading node and sent to an interface node.

The speed of the selection process by the interface nodes is yet to be evaluated. The throughput that can be handled by the interface nodes and the reliability of the interface nodes are critical issues and need to be investigated. As mentioned earlier, the interface nodes should be logically separated from other nodes and have additional connections between them and additional disk drives attached.

7 Conclusion

An architectural design aiming at a high performance database machine has been presented. The system can have a size of terabytes and a throughput of more than 10,000 simple transactions per second with available technology.

There are still a number of issues to be studied. The topics of reliability, such as data backup and system crash recovery, need further research. The performance related issues, such as load balancing, hot spot avoidance, and system behavior when hot spot is present, need further study. Many detailed algorithms need to be developed.

References

[1] V.E. Benes. On rearrangeable three-stage connecting networks. *The Bell System Technical Journal*, pages 1481–1492, Aug 1962.

[2] C. Clos. A study of nonblocking switching networks. *The Bell System Technical Journal*, pages 406–424, Mar 1953.

[3] INMOS Corporation. *The T9000 Transputer Product Overview Manual.* INMOS Corporation, 1991.

[4] G. Gibson & R.H. Katz D.A. Patterson. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of SIGMOD*, June 1988.

[5] D.J. DeWitt. DIRECT–a multiprocessor organization for supporting relational database management systems. *IEEE Transactions on Computers*, C-28:395–406, Jun 1979.

[6] C. Seitz *et al.* Wormhole chip project report. Technical report, Winter 1985.

[7] T. Fei, C.K. Baru, and S.Y.W. Su. SM3: A dynamically partitionable multicomputer system with switchable main memory modules. In *Proc. of the International Conference on Computer Data Engineering*, pages 42–49, Los Angeles. Apr 1984.

[8] J.R. Goodman and C.H. Sequin. Hypertree: A multiprocessor interconnection topology. *IEEE Transactions on Computers*, C-30(12):923–933, 1981.

[9] M. Kitsuregawa, H. Tanaka, and T. Moto-oka. Relational algebra machine GRACE. In *RIMS Symposia on Software Science and Engineering*, pages 191–212, 1983.

[10] Q. Li. *The Architecture and Related Control Problems of a Transputer Based Highly Parallel Database Machine.* PhD thesis, School of Computer Science, Florida International University, Miami, Florida 33199, December 1989.

[11] N. Rishe. *Database Design Fundamentals: A Structured Introduction to Databases and a Structured Database Design Methodology.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1988. 436 pages, hardbound. ISBN 0-13-196791-6.

[12] N. Rishe. Efficient organization of semantic database. In W. Litwin and H.-J. Schek, editors, *Foundation of Data Organization and Algorithms*, pages 114–127. Springer-Verlag Lecture Notes in Computer Science, 1989. vol. 367.

[13] N. Rishe. A file structure for semantic databases. *Information Systems*, 1991. accepted, to appear in Vol 16.

[14] J.A. Rudolph. A production implementation of an associative array processor STARAN. In *Proc. of the Fall Joint Computer Conference*, pages 229–241, Las Vegas, Nev., Nov 1972.

[15] S. Su. *Database Computers.* McGraw-Hill Book Company, New York, N.Y., 1988.

[16] J. Wilkes. The DataMesh research project. In *Proceedings of Transputing'91*, April 1991.

[17] C.l. Wu and T.y. Feng. *Tutorial: Interconnection Networks for Parallel and Distributed Processing.* IEEE Computer Society Press, 1984.