North-Holland
Microprocessing and Microprogramming 25 (1989) 33 – 38

Architecture for a Massively Parallel Database Machine

N. Rishe, D. Tal and Q. Li

School of Computer Science
Florida International University
The State University of Florida at Miami
University Park, Miami, FL 33199, USA

The Linear-throughput Semantic Database Machine (LSDM) is an attempt to bring a massively parallel database machine closer to realization. The primary approach undertaken is to alleviate the traditional processor/disk split which leads to inefficiency by building a more homogeneous database system in which processors and disks are combined together. The LSDM is based on a distributed system of many processors, most of them equipped with a local cache memory and a small disk drive. The individual processors are connected by means of a hypercube network topology. The database model employed for the proposed database computer is the Semantic Binary Model. **Keywords:** Parallel architecture, Database machine, Semantic binary database model, Transputers, Multitudinous secondary storage units.

1. INTRODUCTION

A recent trend in database management systems is to introduce parallelism into computation in order to achieve high throughput and fast response time. Many architecture strategies and hardware alternatives for database machines have been suggested or developed utilizing parallelism in one form or another. The approaches can be roughly classified into three categories:

1. The intelligent secondary storage devices called Cellular-Logic devices. The approach is to build more intelligence into the secondary storage by attaching processors to the read/write head of the storage device. In this way, data processing can be done entirely or partially on the device while reducing the limitations of conventional storage devices. A number of database computers such as CASSM [16], RAP [8], and RARES [7] which exploit these ideas have been produced.

2. Associative memory systems have been introduced into database machines. Using this approach, the contents of all memory cells can be examined and manipulated simultaneously, while attaining tremendous parallel processing power. An example of this category is STARAN [14].

3. The third category includes the multiprocessor database computers. The key idea is to achieve parallelism by distributing the data among several cooperating processors. The following are several examples of research and commercial computers utilizing this approach. GAMMA[4] is a multicomputer system consisting of a group of identical computers. MICRONET is a bus-structured multiprocessor DBM. It uses a custom-built MICRONET bus[16] to connect a number of processors with the host system. SM3[5] is the continuation and extension of MICRONET. In SM3, a hardware main memory switching technique is used. DIRECT[3] is a system using multiprocessor, multi-disk, and multi-memory modules.

In general, all these approaches employ a number of processing units and a few large disks in order to increase the parallel processing power and the parallel secondary storage accessing power. With the exception of Cellular-Logic devices, the emphasis has been put on the inter-processor communication and the communication between the processors and the memory modules.

However, in the overall structure of the contemporary systems, the processing units are one group, and the secondary storage units, possibly with some preprocessing power, are another. The two groups are connected by communication channels. The system's formation is still "Processor–I/O Channels–Secondary Storage". The traditional I/O channel bottleneck is still present. This problem remains no matter how many processors we add to the machine, because the throughput is dominated by the relatively poor performance of the disk and channel.

This research has been supported in part by a grant from the Florida High Technology and Industry Council.

One can see the analogy between the processor/disk split and the processor/memory split [6] that leads to the von Neumann bottleneck. The I/O bottleneck phenomenon is caused by the incompatibility between the granularity of the processing units and the granularity of the secondary storage units. For example, various systems have many small capacity processors (e.g. the Connection Machine [6]), but each processor is not powerful enough to efficiently deal with the secondary storage alone. Therefore, in a typical implementation of a database machine, many processors are grouped together to talk to the secondary storage units. Thus, the data has to be collected from the secondary storage devices and sent through I/O channels and then distributed among the processors.

The existence of the Processor–Channel–Disk formation prevents one from building high degree of parallel processing power into database machines. To dramatically increase the processing power of database machines, a new parallel architecture which can efficiently accommodate a large number of processors and secondary storage units is a necessity.

2. THE PROPOSED PARALLEL ARCHITECTURE

Our current research attempts to introduce highly parallel computing power into a database machine by altering the Processor–Channel–Disk formation and removing the I/O bottleneck.

We suggest the following design principles:

1. The database machine should consist of many fairly powerful processors, each capable of handling, independently, the data retrieved by a group of related secondary storage operations. (Each processor can be further replaced by a group of processors. But, since they act like a single processor with respect to the secondary storage units and other processors, we prefer to treat them virtually as a single processor in this context.)

2. In order to utilize the locality, the secondary storage should consist of many small units, each of them associated with a processor in a physical proximity. Thus, accesses to different secondary storage areas will most likely be handled concurrently by different secondary storage units.

3. Suitable database model and implementation strategies have to be selected to benefit the most from the architecture pattern – the data has to be distributed among the secondary storage devices in such a way that the majority of the data stored on a unit will be processed by the local associated processor.

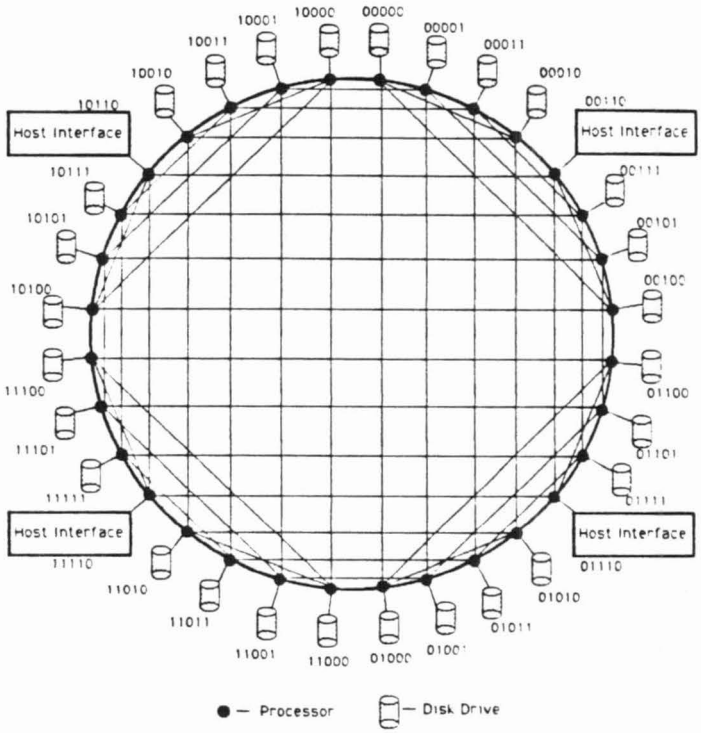

Figure 1 LSDM Hypercube Network 32 nodes are depicted

We propose here a new architecture concept, called the Linear-throughput Semantic Database Machine (LSDM), which follows the above design principles and attempts to alleviate the typical bottlenecks of contemporary database management systems. The LSDM consists of many, thousand[s], fairly powerful processors, each equipped with a local memory and a small capacity, non-volatile storage (e.g., a 20 megabyte disk). Figure 1 shows a reduced version of the database machine. As mentioned before, each processor can actually be composed of a group of several processors at a lower granularity level. However, this implementation detail is transparent at the system level which we are currently dealing with (we chose to treat the entire group as a single unit for simplicity reasons.) In order to support load balancing and sharing of information, the processors-disk units are mapped onto a tightly-coupled communication network.

Each processor-memory-disk unit is similar to a small independent computer and has its own basic operating system. There is no memory and secondary storage sharing between the processors. A processor can store and retrieve data to/from a remote disk unit (which is not directly connected to it) by exchanging messages with the processor physically attached to the desired unit through the inter-processor communication network. Since there are no global memories (main or secondary), the bus contention or I/O channel contention existing in many systems are eliminated. At this point, one may argue that the contention is merely moved to the inter-processor network. But, only a small percentage of data needs to be moved through the network. Based on the locality characterizing a subtransaction performed by a processor, we foresee that only a small portion of the storage accesses will actually take place between a processor and remote storage units. Therefore, the system configuration can grow to include many processor-disk units and attain a high parallel computing power. Moreover, since typically most of the processor-disk units process different queries simultaneously, the throughput will be high even when there exists some contention in the network.

The usage of a large number of small secondary storage units brings several advantages. First, it provides massively parallel I/O operations and relaxes the I/O bottleneck which exists in all database machines. Secondly, since the number of units is large, each individual unit can be small, thus providing a potential to utilize various secondary storage techniques, such as "bubble memory". When hardware cost allows, it is possible to replace the entire secondary storage by main memory without down-grading the size of the database system. In other words, the architecture provides a smooth transition from a secondary-storage-based system to a main-memory-based system. Finally, due to the small size of the secondary storage units, it is more convenient to backup data, replace unit and recover from unit failure without significantly disturbing the operation of the system.

The database model and its implementation strategies have to ensure the locality of references to the secondary storage. It is the efficient inter-processor communication network which must be developed to maximize the communication capacity. The following two sections discuss these issues.

3. THE SEMANTIC BINARY DATABASE MODEL

The choice of the database model and its implementation should satisfy the following requirements:

1. It should be a model which reflects the state-of-the-art in databases.
2. Its implementation should facilitate parallel processing.
3. The model has to offer a great deal of logical independency between logical items and physical structure. This will allow a natural distribution of data among the many physical storage units and will benefit the most from the architecture.
4. The nature and the granularity of items stored or retrieved have to be compatible with the architecture granularity.

The database model chosen for the proposed database computer is the Semantic Binary Model [10]

Since 1974[1], many semantic data models have been studied in the Computer Science literature. Although somewhat different in their terminology and their selection of tools used to describe the semantics of the real world, they have several common principles. The advantages of the semantic models versus the Relational and older models with respect to database design, database maintenance, data integrity, conciseness of languages, and ease of DML programming are known [10]. The semantic models have potential for much more efficient implementation than the conventional data models. This is due to two facts:

- All the physical aspects of representation of information by data are user-transparent in the semantic models. This creates greater potential for optimization: more things may be changed for efficiency considerations, without affecting the user programs. The Relational Model has more data independence than the older models, for example the order of rows in the tables (relations) is transparent to the user. The semantic models have yet

more user-transparency. For example, the representation of real-world entities by printable values is transparent to the user.

- In the semantic models, the system knows more about the meaning of the user's data and about the meaningful connections between such data. This knowledge can be utilized to organize the data so that meaningful operations can be performed faster at the expense of meaningless operations.

In LSDM we use the Semantic Binary Model (SBM) [9], a descendant of the model proposed in [1]. This model does not have as rich an arsenal of tools for semantic description as can be found in some other semantic models. Nevertheless, the SBM has a small set of sufficient simple tools by which all the semantic descriptors of the other models can be constructed. This makes SBM easier to use for the novice, easier to implement, and usable for delineation of the common properties of the semantic models.

The semantic binary model represents the 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 graphical database schema and the integrity constraints determine what sets of facts are meaningful, i.e. can comprise an instantaneous database (the database as may be seen at some instance of time.)

The formal semantics of the semantic binary model is defined in [10] using the methodology proposed in [11]. The syntax and informal semantics of the model and its languages (data definition, 4-th generation data manipulation, non-procedural languages for queries, updates, specification of constraints, userviews, etc.) are given in [9]. A non-procedural semantic database language of maximal theoretically-possible expressive power is given in [12]. In this language, one can specify every computable query, transaction, constraint, etc.

In our implementation, the entire database, including all indexing information, is represented by one logical coherent file partitioned into segments of sufficiently small size, so that they may be stored in small disks. The number of processor-disk formations is sufficient to accommodate the totality of segments. An important property of the implementation is that for most simple queries, the data needed to answer a query is stored in a sequence of consecutive records, which localizes each simple query to a particular processor-disk unit in most cases. This helps to reduce, significantly, the traffic in the inter-processor communication network and the number of I/O operations on each unit.

4. THE INTER-PROCESSOR COMMUNICATION NETWORK

Due to its wide bandwidth and highly regular connections, a modification of the n-cube topology has been chosen to connect the network. The n-cube [17] is defined as follows: Each of the 2^n nodes is labeled from 0 to 2^{n-1} by a unique binary string of length n. Two nodes are connected iff they differ in exactly one bit position. Each processor-disk unit is a node in the hypercube.

Since this is a tightly coupled inter-processor network, the communication protocol of the network will be kept as simple as possible in order not to introduce heavy overhead into the system. A three-layer protocol is used, namely, the processor layer, the intermediate layer, and the communication layer. In the processor layer, the data appears in its original form. In the intermediate layer, the data is partitioned into packets of certain size. In the communication layer, each packet is attached with a header which contains the destination, packet length and some other information, which is the form of data actually being sent through the network.

The protocol can be bypassed under some special circumstances. For instance, when a large chunk of data needs to be transferred between adjacent nodes in the network, which happens during the dynamic load balancing between the processor disk-units, the data is sent in its original form directly from the originator to the destination.

Practical considerations, such as heavy traffic between consecutive file fragments, dictate that nodes associated with those fragments should be neighbors in the network. Therefore, we need a linear ordering of hypercube connections between the processors. This facilitates sequential operations between nodes, as well as random connections. Moreover, in order to support data backup, trouble shooting and module replacement, sub-hypercubes should be represented in each sequence by consecutive sub-sequences. For these reasons, we have developed a hypercube sequencing algorithm[13].

5. LOGICAL PROCESSING OF TRANSACTIONS

5.1. Efficiency of update transactions

A transaction is a set of interrelated update requests to be performed as one unit. Transactions are generated by programs and by interactive users. A transaction can be generated by a program fragment containing numerous update commands, interleaved with other computations. However, until the last command within a transaction is completed, the updates are not physically performed, but rather accumulated by the DBMS. Upon completion of the transaction the DBMS checks

its integrity and then physically performs the update. The partial effects of the transaction may be inconsistent. Every program and user sees the database in a consistent state: until the transaction is committed, its effects are invisible.

A completed transaction is composed of a set of facts to be deleted from the database, a set of facts to be inserted into the database, and additional information needed to verify that there is no interference between transactions of concurrent programs. If the verification produces a positive result, then the new instantaneous database is: ((the-old-instantaneous-database) - (the-set-of-facts-to-be-deleted)) ∪ (the-set-of-facts-to-be-inserted)

Efficient performance of update transactions is required, although more than one disk access per transaction is allowed.

We employ no locking, but rather an optimistic concurrency control: just before the transaction is physically performed in the database, the DBMS will check whether all the information retrieved by the user program during the generation of a transaction is still the same, i.e. no other user's transaction has changed any relevant data in the meanwhile.

The stream of completed transactions will proceed in the DBMS through a set of filters: translation of user-views and subschemas, integrity verification, non-interference verification, logging, data encoding, etc. Many of the filters can be implemented by separate processors. After that, the transaction will be split into subtransactions which will be performed on several disks by their processors. After performing the subtransactions, the processors will respond to the transaction scheduler. If any processor fails to execute its part of the transaction, the whole transaction is undone.

5.2. Queries

Queries are requests for retrieval of information. They can be generated by interactive users or user programs. Normally, when a transaction is generated by a user program, several queries are made in order to decide the contents of the transaction.

Queries will come to the physical part of the DBMS in several streams, through filters. These streams will be independent of the stream of completed transactions. After initial translation and filtering, the queries will go through a multiplexer-integrator processor. A multiplexer-integrator will split a query into a set of parallel and/or sequential sub-queries to be performed by different processors with different disks. When the answers arrive, the results are integrated and sent to outer layers of the DBMS. While waiting for the answers, the multiplexer/integrator continues to work on other queries and send them to other disks.

6. THE LOW LEVEL SYSTEM ASPECTS

The proposed system can either lend itself to a stand alone database computer or works as a backend computer connected to hosts. A number of processors in the system will act as the hosts or interfaces to user hosts. In particular, nodes in certain symmetric points in the hypercube are chosen as the interface nodes.

The processor-disk units comprising the machine work simultaneously on different segments of the same query and on concurrent queries as well, offering two levels of parallelism. The hosts receive streams of user's requests. Each request is dispatched to the processor having the best control of the relevant fragments of the database. The processor then decomposes the request into smaller operations and communicates them to subcontracting processors. The processors related to a request can communicate with each other to get the data necessary to carry out the operations concurrently. A query can be processed in the system in different ways depending on the characteristics of a particular query, amount of data required to be transferred through the inter-processor network, system load, etc. In some sense it is analogous to systolic processing. A query is initially sent to one or more relevant nodes which hold the data needed for the first step processing. Then, the intermediate results are sent to some other nodes, and so on, until the final result is derived. Semantically, it is crucially different, since one node usually acts as the master processor for a particular query. Subqueries are derived and sent from the master processor to other processors and the results are sent back to the master processor and the final result is derived. These two basic methods can be used in a combined fashion. With the completion of a request, the results are sent to the host nodes and eventually back to the users.

A small number of processor-disk units in the network may be used as contingency units. They are idle in a normal operation situation. When an extremely heavy load on a unit caused by a burst of references to the unit is detected, one of those contingent units will be used as a duplicate of the overloaded unit. Queries that would be sent to the overloaded unit will be sent to one of the two units. In other words, each one of them has approximately half of the load. Since the units are small, the duplication of an arbitrary unit consumes a reasonable amount of time. Some mechanism is still to be developed in order to guarantee the consistency between the duplicated units.

The processors that we are currently using are the INMOS transputers [2].

7. CONCLUSION

The LSDM has been designed to alleviate several problems in contemporary database management systems and to offer high performance and reliability. The large amount of coherent disk-processor units create fertile ground for data backup and failure recovery without significant disturbance to the normal operation of the whole system. The architecture remains feasible as the number of disk-processor units comprising the machine rise. The LSDM attempts to exploit the emerging advantages of VLSI and is built from a few modular components. User requests can be divided into fairly small granules achieving two levels of parallelism: One is at the request level. When the system is under a heavy load, many requests are processed in parallel by different processors, and disks related to different requests are referenced in parallel also. Another is at a lower level where each request is processed in parallel by several processors using, in parallel, several disks. Finally, the I/O bottleneck and its by-products are eliminated since only several processors and disks are normally engaged in any given query; reducing (on the average) the I/O interference between the requests.

One of the primary goals of the proposed architecture is to achieve a high throughput and approximate linearity of the throughput in the degree of parallelism. The throughput is measured in the number of transactions per day. The degree of parallelism is the number of processor–secondary storage units employed by the system. The linearity is to be achieved for a typical transaction load comprised of a large number of relatively small, localizable queries and transactions.

ACKNOWLEDGEMENT

The authors are grateful to David Barton and Nagarajan Prabhakaran for a very fruitful exchange of ideas.

References

[1] J.R. Abrial. Data semantics. In J.W. Kimbie and K.L. Koffeman, editors, *Data Base Management*, Noth Holland, 1974.

[2] INMOS Corporation. *Transputer Architecture Reference Manual.* INMOS Corporation, Bristol, U.K., 1986.

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

[4] D.J. DeWitt, R.H. Gerber, G. Graefe, M.L. Heytens, K.B. Kumar, and M. Muralikrishna. GAMMA: a performance dataflow database machine. In *Proc. of the 12th International Conference on Very Large Data Bases*, pages 228–237, Kyoto, Japan, Aug 1986.

[5] 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.

[6] W.D. Hillis. *The Connection Machine.* The MIT Press in Artificial Intelligence, U.S.A., 1985.

[7] Q. Li, W.B. Feild, and D. Klein. Channel design primitives in occam. In *Proc. of the 3rd US Occam User Group Meeting*, Chicago, IL, September 1987.

[8] C.S. Lin, D.C.P. Smith, and J.M. Smith. The design if a rotating associative memory for relational data base applications. *ACM Transactions on Database Systems*, 1(1):53–65, 1976.

[9] E.A. Ozkarahan, S.A. Schuster, and K.C. Smith. RAP - an associative processor for data base management. In *Proc. AFIPS Conference*, pages 370–387, Anaheim, Cal., 1975.

[10] N. Rishe. *Database Design Fundamentals: A Structured Introduction to Databases and a Structured Database Design Methodology*, chapter 1, pages 1–40. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1988.

[11] N. Rishe. *Database Semantics.* Technical Report 87-002, Computer Science Department, University of California, Santa Barbara, 1987.

[12] N. Rishe. On denotational semantics of data bases. In *Lecture Notes in Computer Science, vol. 239*, pages 249–274, Springer-Verlag, 1986.

[13] N. Rishe. Postconditional semantics of data base queries. In *Lecture Notes in Computer Science, vol. 239*, pages 275–295, Springer-Verlag, 1986.

[14] N. Rishe, D. Tal, and Q. Li. A sequenced hypercube topology for a massively-parallel database computer. To appear in Proceedings of Second Symposium on the Frontiers of Massively Parallel Computation, Oct. 1988. Fairfax, Va.

[15] 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.

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

[17] S.Y.W. Su, L. Nguyen, E. Ahmed, and G. Lipovski. The architectural features and implementation techniques of multicell cassm. *IEEE Transactions on Computers*, C-28(6):430–445, Jun 1979.

[18] L.G. Valiant and G.J. Brebner. Universal schemes for parallel communication. In *In STOC, ACM Conference Proceedings*, Milwaukee, 1981.