DECEMBER 1981 Vol.4 NO. 2

a quarterly bulletin of the IEEE computer society technical committee on

Database Engineering

Contents

Database Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Database Engineering. Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Database Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in Database Engineering Technical Committee is open to IEEE Computer Society members, student members, and associate members. (Application form in this issue.)

Letter from the Editor

This special issue of **Database Engineering** is intended to report on the status of various on-going projects in the area of database machines and to serve as a forum for presenting some initial results from recently started research projects. S.W. Song surveys existing database machine designs under an interesting taxonomy he proposes. David Hsiao describes research programs in multi-mini database management and database machine architecture at the Ohio State University. The results obtained from the DIRECT prototyping efforts at the University of Wisconsin at Madison are summarized in a paper by Haran Boral and David DeWitt. They also indicate proposed research activities on database machines at Wisconsin.

Michael Ubell shows the hardware/software architecture of the IDM-500 currently being marketed by Britton-Lee, Inc. He also indicates design decisions that went into architecting the system, and gives concrete cost/performance figures for the system. Paula Hawthorn describes a proposed methodology for determining the cost/performance requirements for a database machine architecture to support applications that access a large volume of data. David Shaw, et al., presents an overview of the NON-VON machine currently being constructed at the Columbia University. The architecture of a backend database sysem being studied at the University of Maryland is described in a working paper by S. Bing Yao, et al. The system is being prototyped to study feasibility of VLSI implementation. S.K. Arora and S.R. Dumpala introduce us to the Well-Connected Relation Computer, which has been designed to simultaneously support the relational, hierarchical and network models of data on the same physical data. They also propose two different storage structures for the physical data.

This issue takes up nearly twice the number of pages that I initially estimated it would. The reason is simply that I vastly underestimated the number of authors who have affirmatively responded to my invitation for papers. To my pleasant surprise, I now realize that I also seriously underestimated the quality of the papers that these authors eventually contributed to this issue. I would like to thank them again for the enthusiasm and cooperation they have shown. Due to constraints to our budget and the editors' time, however, we will restrict ourselves to publishing short papers (4 to 10 double-spaced pages) in the future. We will accept (and invite) papers that describe the status of on-going research projects, that explain design decisions that have gone into constructing commercial systems, and that motivate and summarize (with no formulas or theorems) important new ideas being developed.

I would like to take this opportunity to say that it is my privilege to be assisted by four of my most outstanding colleagues. The pleasure of interacting with the associate editors to reach editorial decisions, solicit papers from prospective authors and chart the course of this publication has been more than enough of a reward for my time. In fact, each of them has volunteered to publish one issue for 1982. Our plans for 1982 are as follows. Don Batory is preparing for a special issue on Directions in Physical Database Research for the March issue. He is soliciting short papers (2 to 4 pages) that describe on-going projects on physical database design. Deadline for submitting papers for the March issue is December 1, 1981.

The June issue will be managed by Randy Katz. He plans a special issue on Database Applications for VLSI Designs. He is interested in papers that describe the

status of and/or initial results from on-going research projects in the area. Deadline for the June issue is March 15.

David Reiner will manage a special issue on Database Query Processing for September. He is soliciting papers that describe relatively new ideas on query processing for both the centralized and distributed database systems. Deadline for the issue will be June 15.

Alan Hevner will be in charge of a special issue on Research in Distributed Database Systems for December 1982. He would like to solicit papers that emphasize the current status of research in progress, express opinions about the current state of the art and future researh directions, or papers that describe interesting new ideas that have not been widely publicized. Deadline will be September 15, 1982.

All papers that fall into any of these categories should be submitted to the associate editors in charge of the special issues. Although our plans for 1982 place emphasis on the four topics mentioned above, short papers on other topics related to database engineering will be welcomed. Papers that do not deal with topics that the special issues are planned for and papers that are submitted too late for publication consideration in one of the special issues should be sent to me. We will consider publishing a special issue (?) on general topics during the summer. Also please send any comments on the contents and direction of our publication to me. We will publish selected comments in the **feedback** section.

A Survey and Taxonomy of Database Machines*

S. W. Song**

Department of Computer Science

Carnegie-Mellon University

Pittsburgh, Pa. 15213

The purpose of this paper is twofold. We survey existing database machine designs, and propose a taxonomy. Some recent designs that exploit the rapidly advancing VLSI technology are included in the survey. At the risk of oversimplification, the proposed taxonomy attempts to group many seemingly different designs into a few categories by concentrating on their similarities. First we characterize the problem by identifying two bottlenecks. Then we describe the several dimensions the taxonomy is based on. The remainder of the paper can then be viewed as a detailed presentation of these categories, with a brief survey of previous database machine designs that fall into each category.

1. Problem Characterization

We can identify two potential bottlenecks, namely the I/O bottleneck and the so-called von Neumann bottleneck. Latency and bandwidth constitute the main problems of I/O. Careful design of access paths and maintenance of appropriate indices help in reducing the number of disk accesses. To eliminate the need for access paths and indices, database machines with fast retrieval times have been proposed. Most database machine designs use the *logic-per-track* approach [29] which can provide fast on-the-fly retrieval.

In a compute-bound task a data element participates in many operations. The best place to carry out such a task is inside the primary memory, because of its faster access speed. It was observed in [18] that in a conventional von Neumann machine, each operation typically fetches one or more operands from memory. Hence the amount of I/O (between the memory and the central processing unit) is proportional to the number of operations to be performed rather than the number of inputs required for the computation. The von Neumann bottleneck is in fact an I/O bottleneck in lesser scale. To reduce traffic through this bottleneck, many works have been done in the context of optimal register allocation or usage of cache memory. Studies of such solutions abound in the literature and fall outside the scope of the present work.

---

*This research was supported in part by the Office of Naval Research under Contracts N00014-76-C-0370, NR 044-422, and N00014-80-C-0236, NR 048-659, in part by the National Science Foundation under Grant MCS 78-236-76, and in part by the Defense Advanced Research Projects Agency under Contract F33615-78-C-1551 (monitored by the Air Force Office of Scientific Research).

**Present address: University of Sao Paulo, Institute of Mathematics and Statistics, Department of Applied Mathematics, C. P. 20570, CEP 01451, Sao Paulo, SP, Brazil. At the time this work was done, the author was supported in part by CNPq, Conselho Nacional de Desenvolvimento Cientifico e Tecnologico, Brazil, under Contract 200.402-79-CC, and was on leave from University of Sao Paulo.

2. Dimensions of the Space of Database Machine Designs

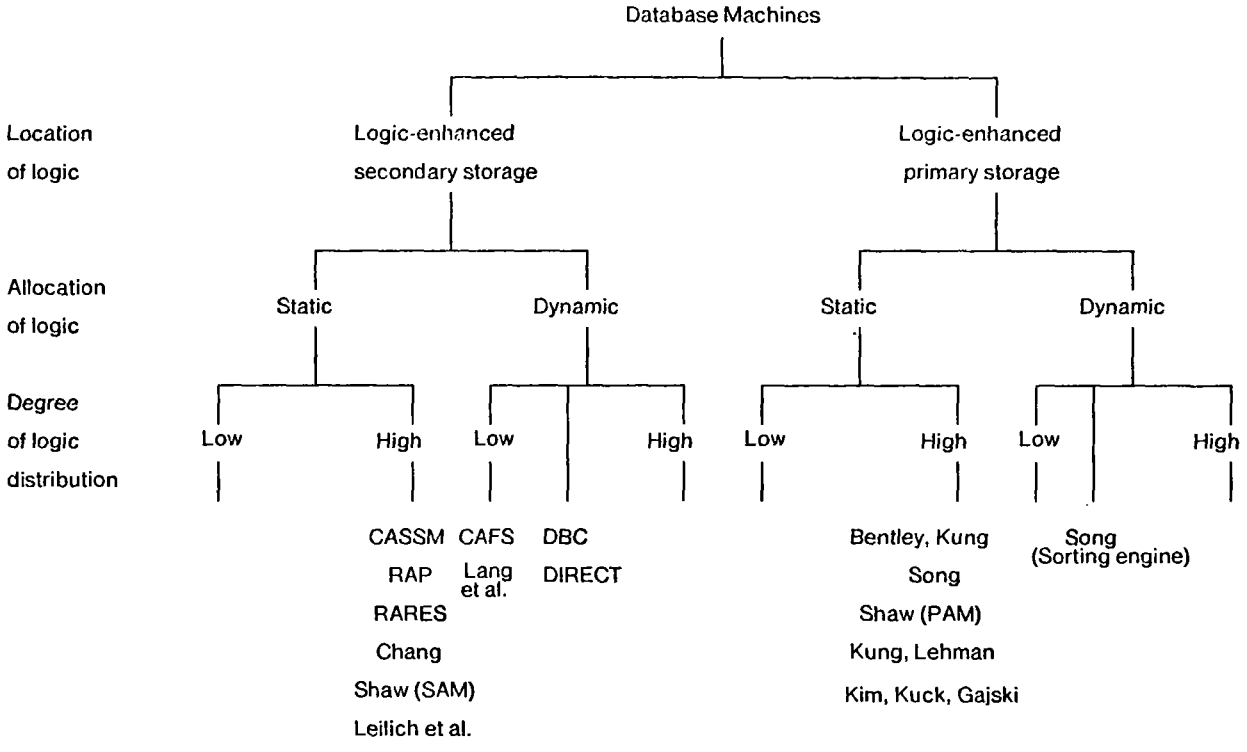


Figure 2-1: An overall framework.

The taxonomy is based on several dimensions. Depending on where special-purpose logic is applied, we have *logic-enhanced secondary* or *primary storage* designs. Most previous designs, as we shall see, are variations of the first type. Another dimension along which designs can be classified is the way logic is allocated to storage units, whether *statically* or *dynamically*. A third dimension is the degree of distribution of logic among memory elements, defined in [27] as the number of storage elements associated with each processing unit. Along this dimension, we may have a wide spectrum of designs. High degree of logic distribution signifies faster computation rate, and this should be such that a balance between computation and data access rate is achieved. Classification of designs along this dimension is also important since it is related to the cost of physical implementation. As an example, we can view a conventional von Neumann machine as a logic-enhanced primary storage device occupying the lowest end of the logic distribution spectrum. The allocation of logic is dynamic since one processing unit serves the entire memory. Figure 2-1 shows an overall framework, illustrated with a few particular designs. The reader can refer to this figure when reading the rest of this paper. Its purpose is to illustrate the several categories and not to show the exact positions of the various designs. No attempt should therefore be made to derive quantitative conclusions. Only with a more specific definition of degree of logic distribution and a more detailed analysis of the implementations involved can such positions be made more precise.

3. Logic-enhanced Secondary Storage Designs

3.1. Uni-Search-Processor Scheme

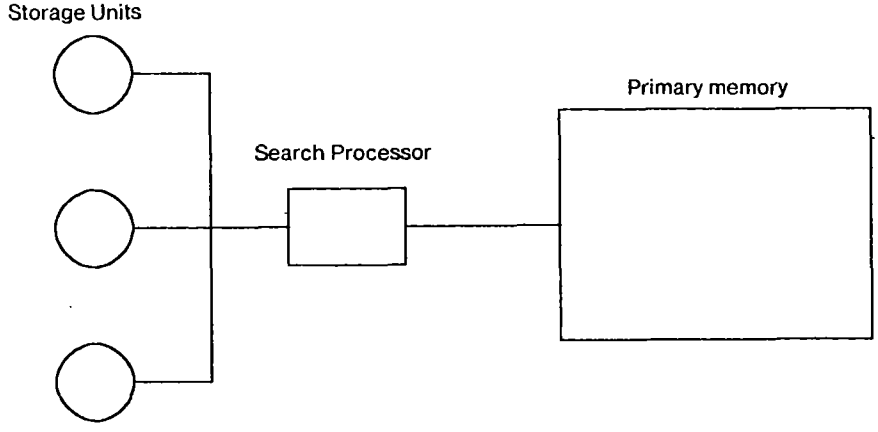


Figure 3-1: Uni-search-processor model.

One search processor is attached between the secondary storage devices and the primary memory (Figure 3-1). Irrelevant data can be filtered out before they reach the primary memory, and thereby reducing I/O traffic. Some early examples of database machines are of this kind. This scheme occupies the low end of the logic distribution spectrum. The allocation of logic is dynamic since one search processor serves the entire secondary memory. Examples include the Content Addressable File Store, or CAFS [1, 9], and the designs by Bancilhon and Scholl [2], and by Lang et al. [19].

3.2. Multi-Search-Processor Scheme - Static Allocation

A storage unit considered in the following is usually a disk track, bubble memory, or charge-coupled device. The static allocation scheme pre-allocates one search processor to each storage unit, requiring as many search processors as there are storage units (see Figure 3-2). With such designs, the search time is typically tens of milliseconds. They occupy the high end of the logic distribution spectrum.

Among the designs which fall into this category, we cite the following. (Some of these designs, conceived with static allocation in mind, are now shifting to the dynamic allocation scheme, to be discussed shortly afterwards.) CASSM [32], or Content Addressed Segment Sequential Memory, designed and with a prototype built at the University of Florida at Gainesville, is one of the earliest design efforts and thus has exercised considerable influence over other designs. RAP [25, 26], or Relational Associative Processor, was designed and implemented at the University of Toronto. It supports the relational data model and uses CCD memories as storage units. Chang [8] proposes slightly modified major/minor loop bubble chips to accommodate storage and access for relational databases. RARES [23], also designed to support the relational model, differs from others mainly in that tuples are stored across the tracks of the head-per-track disk storage. A system of up to 14 search processors has been designed and implemented by Leilich et al. [21].

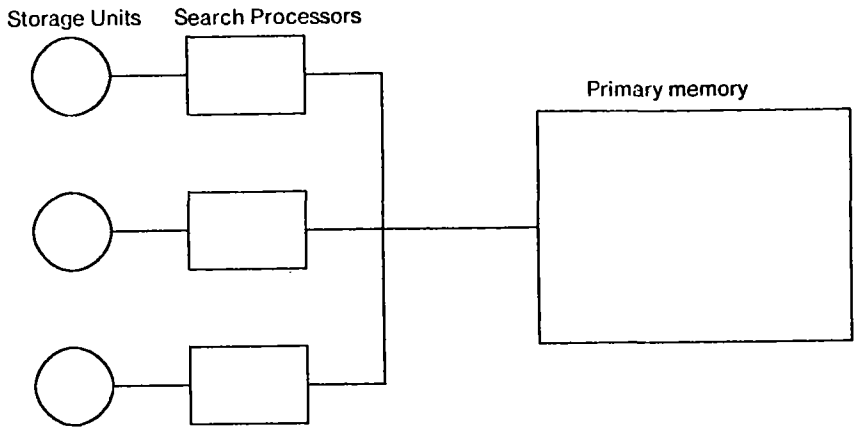


Figure 3-2: Multi-search-processor (static allocation).

In the static allocation scheme, each storage unit has its own private search processor and it does not matter in which storage units a file should reside. Therefore storage organization is quite simple. The main problem, of course, is the waste of too much potential resource. The global database may contain information for many different users and applications. In processing one specific query, however, the actual load (i.e., the amount of data needed for the processing) may well constitute a tiny fraction of the whole database. Providing logic to all disk tracks of the entire database is analogous to a memory management system in which enough physical memory is provided to hold all programs ever written by all the users of a given installation.

3.3. Multi-Search-Processor Scheme - Dynamic Allocation

Still using our analogy with a memory management system, a virtual memory system allocates physical memory dynamically to segments of programs only when their presence in the main store is required for execution. The amount of physical memory can thus be significantly less than the total program space. Similarly, in the dynamic allocation scheme, a number of search processors are allocated dynamically to those storage units containing information to be processed. The search logic is therefore distributed to the entire database and hence will occupy the lesser end of the logic distribution spectrum as compared to those with static allocation. Depending on how search processors are connected to storage units, we have many variations of this scheme. Two of these, which correspond to real database machine designs, will be examined.

3.3.1. Complete-Bipartite-Graph Connection

In this scheme each search processor is connected to every storage unit of the database, as depicted in Figure 3-3. It works in the following manner. Each storage unit keeps broadcasting its contents to all the search processors. An individual search processor can choose to listen to one of the storage units and ignore the others. We have thus a very flexible connection. Any search processor can operate on any storage unit. Furthermore, several search processors can operate independently on the same storage unit, as long as they do not contend (for instance, they should not all try to update at the same time). This connection also allows a multi-user system in which some search processors may be operating to answer one user's queries, while some other search processors are working on another user's queries. Flexibility is of course obtained at the cost of the number of connections and the more complex control mechanism of the search processors. DeWitt

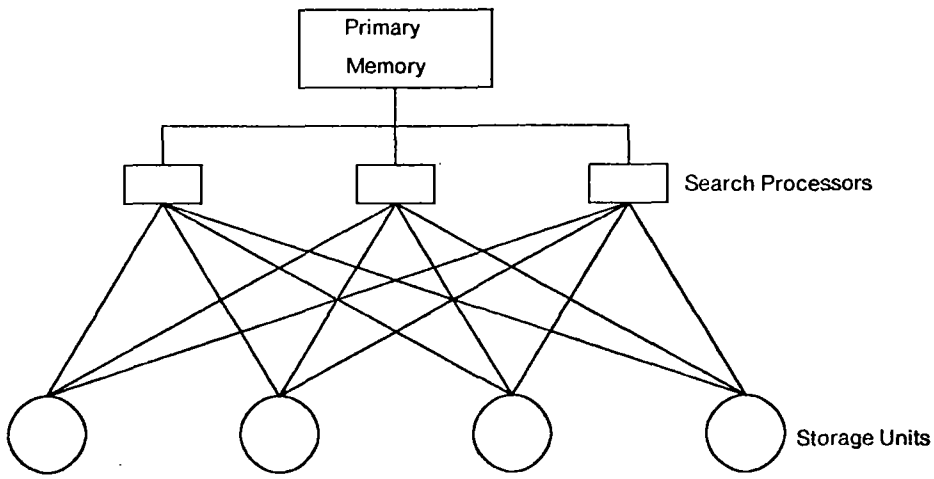


Figure 3-3: Complete-bipartite-graph connection.

[10], at the University of Wisconsin, Madison, proposes a design of a system called DIRECT which uses essentially this connection. A prototype has been built.

3.3.2. Partitioned-Storage-Units Connection

In this connection scheme, if we have t search processors and n storage units (typically, $t \ll n$), then all the storage units will be divided into n/t partitions, each with t storage units. The t search processors can be connected to the storage units of one partition, but not to storage units of different partitions. Data residing in one partition can be examined by the search processors in essentially one disk revolution time (assuming a storage unit to be a disk track). Therefore, if related data are clustered into the same partition, they can be searched very quickly. We thus see that this scheme can provide the same performance as the static allocation scheme, given that we have enough search processors (t is sufficiently large) and related data are properly clustered.

The Data Base Computer [3], or DBC, designed by a group headed by D. Hsiao at the Ohio State University, fits into this model. The DBC design uses moving-head disks as storage devices, the only requirement being the parallel read-out capability of the t tracks of one disk cylinder. Data read out from one cylinder are then fed into t search processors. The DBC design was conceived with dynamic allocation approach in mind, and has provided extensive literature on such issues as data clustering and security checks [4].

3.4. Appraisal

Logic-enhanced secondary storage designs are based on the *logic-per-track* philosophy and have one common goal: that of providing efficient on-the-fly search of massive amounts of data in one or a few disk rotations. They constitute promising approaches to the important selection operation. Some other frequently used database operations, however, require not only knowledge of the values of individual data items, but depend on some kind of interaction among data items. The relational join between two relations of size n each, for example, requires $O(n^2)$ comparisons in its straightforward implementation. With a secondary associative storage device, it can be implemented as follows. For each tuple of one relation, we extract the specific field over which the join is being performed. Then

in one revolution time we compare it with the corresponding field of all the tuples of the second relation. Therefore the join operation can in principle be obtained in approximately O(n) revolutions, where n is the number of tuples of the first relation. While this linear performance result might seem quite acceptable at a first sight, we have to keep in mind that one revolution time is on the order of tens of milliseconds. Therefore this mechanism is acceptable as long as we have a small number of argument tuples.

Some recent designs combine the secondary associative storage devices with a logic-enhanced primary memory. In such designs, the secondary associative memory plays an important role in the case when the problem size is too large to be handled entirely in the primary memory. Appropriate partitions can be retrieved by the logic-enhanced secondary storage devices and delivered to the primary store. For example, Lin [24] discusses the usage of associative secondary storage to aid in external sorting. Sorting is also discussed in the article on RARES [23]. The method is based on the knowledge of a histogram concerning the key values. By using content addressability of the secondary store, the appropriate partition is brought into the main memory which is assumed to be fast enough to produce sorted sequences in a pipelined fashion as a new partition is being retrieved. Discussions of several partitioning schemes can be found in [27, 31].

4. Logic-enhanced Primary Storage Designs

Depending on the degree of logic distribution, several kinds of logic-enhanced primary storage designs can be considered. At the low end of the spectrum is the attachment of special-purpose hardware of limited size to a conventional passive memory. In such designs, logic is allocated dynamically to the entire memory. At the other end of the spectrum are the designs using the so-called smart memory (which we will refer to as *logic-per-datum* designs), in which there is a commingling of logic and memory elements in a fine grain. Such designs are of very high performance and constitute a departure from the von Neumann architecture.

4.1. The Post-Processors of The DBC Design

In the DBC design mentioned earlier, functions such as sorting of retrieved records, relational join operations on two sets of records retrieved from secondary memory, and such set functions as maxima and average, are all handled by what is known as the post-processors. Recall that search logic is allocated dynamically to the secondary memory so that a cylinder of data can be content searched in essentially one disk rotation. The retrieved data are fed, in a pipelined fashion, to the post-processors. The post-processing functions are presented in a number of reports describing the sort operation [13] and the join operation [12, 14]. In particular the last report also contains a comparison of the DBC join method with other proposed methods. The post-processing functions are performed by a multiprocessor system consisting of a number of linearly connected processors each with private memory. In an earlier description [12] they share an associative memory for storage and fast retrieval of join attribute values. The associative memory maps each unique join attribute value to an integer index. To join relations A and B, for example, the tuples of relation A is first stored in appropriate memory locations according to the integer index provided by the associative memory. Then for each tuple of relation B whose join attribute value is in the associative memory, the corresponding tuples of A can be located. The result tuples can thus be obtained by concatenation. This first design has the drawback of not being easily hardware-extensible. All processors share the same associative memory which will become a bottleneck when the number of processors increases. A new design described in [14] distributes the associative memory among all the processors, such that each will contain a fraction of the original associative memory.

4.2. The Hierarchical Associative Architecture

A hierarchical associative architecture has been proposed by Shaw [28] for the efficient evaluation of relational primitives such as join, project, and select. It consists of a hierarchy of associative storage devices under the control of a general-purpose processor. At the bottom of the hierarchy is a secondary associative memory (SAM), which may be implemented using parallel logic-per-track disks, as in CASSM, RAP or RARES. At the top of the hierarchy is a primary associative memory (PAM), capable of fast content-based searches. Complex relational primitives such as the join operation on two relations are evaluated in the primary associative memory, with the assistance of the secondary associative memory. Shaw considers the important case of handling large problems whose size exceeds that of the primary associative memory. He distinguishes two kinds of evaluations, namely, internal evaluation where the argument relations can be fit entirely into the primary associative memory, and external evaluation where the relations exceed its capacity. Shaw shows that when argument relations are large, the time required for the evaluation of complex primitives with the hierarchical associative architecture represents a substantial improvement over the results attainable using only secondary associative storage devices.

4.3. Systolic Priority Queues

The systolic array approach has been proposed as a solution to offload costly computations: for a list of systolic algorithms, see [17]; for a discussion of the philosophy of the systolic architecture, see [18]. One systolic design especially useful in database applications is the priority queue proposed by Kung and Leiserson [22]. A linear array of cells is used to store a collection of elements with the possible operations of insertion, deletion and minimum extraction. (Fisher [11] presents designs of systolic arrays for computing running order statistics where ranks other than the minimum and input spaces of higher dimensions are considered.) In addition to storage, some comparison logic is provided at each cell. A sequence of the above operations can be executed concurrently in a pipelined fashion, in such a way that the response time is a constant, independent of the length of the array. Notice that a systolic priority queue is a logic-per-datum device and, as such, occupies the high end of the logic distribution spectrum. However, if such a device of limited size is used to aid in the internal sorting of a much larger collection of numbers, then the degree of logic distribution will be considerably less. (Song [31] examines internal sorting with the aid of systolic devices.) Hence, depending on the size of a systolic device and the problem size it is able to handle, its usage may be economically infeasible, or perfectly viable and justifiable.

4.4. The Systolic Arrays for Relational Operators

Kung and Lehman [16] consider the use of a large number of simple processors connected in a linear array for the handling of relational operators. They describe, among others, arrays for performing intersection (which can also be used for projection with duplicate removal) and join of two relations. A single database transaction may consist of a number of relational operations. Therefore to process all the operations required in one or more transactions, an integrated system containing several systolic arrays is needed. A crossbar scheme connecting the memories holding required data and the special-purpose systolic arrays is proposed.

In a recent work by Kim, Kuck, and Gajski [15], a bit-serial/tuple-parallel relational query processor is proposed. The scope of the study is limited to designing a query processor that will efficiently process data already loaded into the primary memory. As in the case of the systolic arrays of Kung and Lehman, the proposed query processor is designed with the view toward LSI (VLSI) implementation.

4.5. The Tree Machine

A logic-per-datum design consisting of a binary tree of cells has been proposed by Bentley and Kung [5] (and independently by Browning [6, 7]). The internal cells of the binary tree can propagate information to, as well as combine the information of the descendant cells (such as taking the logical and, or select the minimum, etc.). Data elements reside in the leaf cells which are provided with logic to carry out a limited repertoire of instructions. Such a structure is especially suitable for different kinds of searching problems, because of the logarithmic path between the root cell and any leaf cell. It has been extended by Song [30, 31] to handle the sort operation, and relational operations such as project, join, and union. Such a design is of high performance and occupies the high end of the logic distribution spectrum.

4.6. Appraisal

Logic-enhanced primary memory designs are useful for compute-bound tasks where a same datum participates in many operations. On the other hand, I/O-bound tasks such as selection are better handled by logic-enhanced secondary storage devices, before the data even get to the primary memory. The best architecture is perhaps a hierarchy containing both kinds of devices. Logic-enhanced secondary devices may be used to filter out the irrelevant data, and more complex operations on the selected ones are processed in the logic-enhanced primary memory. Designs where logic is allocated dynamically to the entire memory is usually economical to implement but require careful study of the issue of problem partitioning, that is, how to decompose a large problem such that it can be handled by a special-purpose device of smaller size. The logic-per-datum designs can provide very high performance and constitute a departure from the von Neumann architecture. Their implementation cost may however limit their usage to very specialized applications, where fast response time and throughput are required, e.g. on-line bank-teller systems that support a huge number of simultaneous transactions [20].

Acknowledgment

The author wishes to thank Dr. Won Kim for his valuable comments and suggestions.

References

[1] E. Babb.
Implementing a Relational Database by means of Specialized Hardware.
ACM Transactions on Database Systems 4(1):1-29, March, 1979.

[2] F. Bancilhon and M. Scholl.
Design of a Backend Processor for a Data Base Machine.
In *Proceedings of the ACM SIGMOD 1980 International Conference on Management of Data*, pages 93-93g. May, 1980.

[3] J. Banerjee, D. K. Hsiao, and K. Kannan.
DBC - A Database Computer for Very Large Databases.
IEEE Transactions on Computers 28(6):414-429, June, 1979.

[4] J. Banerjee, D. K. Hsiao, and J. Menon.
The Clustering and Security Mechanisms of a Database Computer.
Technical Report OSU-CISRC-TR-79-2, The Ohio State University, Computer and Information Science Research Center, April, 1979.

[5] J. L. Bentley and H. T. Kung.
A Tree Machine for Searching Problems.
In *Proceedings of 1979 International Conference on Parallel Processing*, pages 257-266. IEEE, August, 1979.
Also available as a CMU Computer Science Department technical report CMU-CS-79-142, September, 1979.

[6] S. A. Browning.
Computations on a Tree of Processors.
In *Proc. Conference on Very Large Scale Integration: Architecture, Design, Fabrication*, pages 453-478. January, 1979.
Conference held at Caltech in Pasadena, California.

[7] S. A. Browning.
The Tree Machine: A Highly Concurrent Computing Environment.
PhD thesis, Computer Science Department, California Institute of Technology, January, 1980.

[8] H. Chang.
On Bubble Memories and Relational Data Base.
In *Proceedings 4th International Conference on Very Large Data Bases*, pages 207-229. 1978.

[9] G. F. Coulouris, J. M. Evans, and R. W. Mitchell.
Towards Content Addressing in Data Bases.
Computer Journal 15(2):95-98, May, 1972.

[10] D. J. DeWitt.
DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems.
IEEE Transactions on Computers C-28(6):395-406, June, 1979.

[11] A. L. Fisher.
Systolic Algorithms for Running Order Statistics in Signal and Image Processing.
Technical Report, Carnegie-Mellon University, Computer Science Department, 1981.
In preparation.

[12] D. K. Hsiao and M. J. Menon.
The Post Processing Functions of a Database Computer.
Technical Report OSU-CISRC-TR-79-6, Computer and Information Science Research Center, The Ohio State University, July, 1979.

[13] D. K. Hsiao and M. J. Menon.
Parallel Record Sorting Methods for Hardware Realization.
Technical Report OSU-CISRC-TR-80-7, Computer and Information Science Research Center, The Ohio State University, July, 1980.

[14] D. K. Hsiao and M. J. Menon.
Design and Analysis of Relational Join Operations of a Database Computer (DBC).
Technical Report OSU-CISRC-TR-80-8, Computer and Information Science Research Center, The Ohio State University, September, 1980.

[15] Won Kim, D. J. Kuck, and D. Gajski.
A Bit-Serial/Tuple-Parallel Relational Query Processor.
Research Report RJ 3194, IBM Research Laboratory, San Jose, California, July, 1981.

[16] H. T. Kung and P. L. Lehman.
Systolic (VLSI) Arrays for Relational Database Operations.
In *Proceedings of the ACM SIGMOD 1980 International Conference on Management of Data*, pages 105-116. ACM, May, 1980.
Conference held in Santa Monica, California. Also available as a Carnegie-Mellon University Computer Science Department technical report CMU-CS-80-114, March, 1980.

[17] H. T. Kung.
Let's Design Algorithms for VLSI Systems.
In *Proc. Conference on Very Large Scale Integration: Architecture, Design, Fabrication*, pages 65-90. January, 1979.
Conference held at Caltech in Pasadena, California. Invited paper.

[18] H. T. Kung.
Why Systolic Architecture.
To appear in *Computer Magazine*, 1981.

[19] T. Lang, E. Nahouraii, K. Kasuga, and E. B. Fernandez.
An Architectural Extension for a Large Database System Incorporating a Processor for Disk Search.
In *Proceedings of the Third International Conference on Very Large Data Bases*, pages 204-210. 1977.

[20] P. L. Lehman.
The Theory and Design of Systolic Database Machines.
Thesis Proposal. Carnegie-Mellon University, Computer Science Department, December, 1980.

[21] H. O. Leilich, G. Stiege, and H. C. Zeidler.
A Search Processor for Data Base Management Systems.
In *Proceeding 4th Conference on Very Large Data Bases*, pages 280-287. September, 1978.

[22] C. E. Leiserson.
Systolic Priority Queues.
In *Proc. Conference on Very Large Scale Integration: Architecture, Design, Fabrication*, pages 199-214. Caltech, January, 1979.
Also available as a CMU Computer Science Department technical report CMU-CS-79-115, April, 1979.

[23] C. S. Lin, D. C. P. Smith, and J. M. Smith.
The Design of a Rotating Associative Memory for Relational Database Applications.
ACM Transactions on Database Systems 1(1):53--65, March, 1976.

[24] C. S. Lin.
Sorting with Associative Secondary Storage Devices.
In *Proceedings of the National Computer Conference*, pages 691-695. 1977.

[25] E. A. Ozkarahan, S. A. Schuster, and K. C. Sevcik.
Performance Evaluation of a Relational Associative Processor.
ACM Transactions on Database Systems 2(2):175-195, June, 1977.

[26] S. A. Schuster, H. B. Nguyen, E. A. Ozkarahan, and K. C. Smith.
RAP 2 - An Associative Processor for Databases and its Applications.
IEEE Transactions on Computers C-28(6):446-458, June, 1979.

[27] D. E. Shaw.
A Hierarchical Associative Architecture for the Parallel Evaluation of Relational Algebraic Database Primitives.
Technical Report STAN-CS-79-778, Department of Computer Science, Stanford University, October, 1979.

[28] D. E. Shaw.
A Relational Database Machine Architecture.
In *Fifth Workshop on Computer Architecture for Non-Numeric Processing*, pages 84-95. ACM, March, 1980.

[29] D. L. Slotnick.
Logic per Track Devices.
In Tou, J. (editor), *Advances in Computers, Vol. 10*, pages 291-296. Academic Press, New York, 1970.

[30] S. W. Song.
A Highly Concurrent Tree Machine for Database Applications.
In *Proceedings of the 1980 International Conference on Parallel Processing*, pages 259-268. IEEE, August, 1980.
Also available as a CMU technical report, VLSI Document V055, June, 1980.

[31] S. W. Song.
On a High-Performance VLSI Solution to Database Problems.
PhD thesis, Carnegie-Mellon University, Computer Science Department, 1981.

[32] S. Y. W. Su, L. H. Nguyen, A. Emam, and G. L. Lipovski.
The Architectural Features and Implementation Techniques of the Multicell CASSM.
IEEE Transactions on Computers C-28(6):430-445, June, 1979.

The Laboratory for Database Systems Research at the Ohio State University

David K. Hsiao
Dept. of Computer and Information Sciences
The Ohio State University
Columbus, Ohio

The Laboratory for Database Systems Research was established in August 1980 for the purpose of conducting experimental research in the area of database software and hardware architectures. With substantial equipment grants from the Digital Equipment Corporation, Office of Naval Research and The Ohio State University, the laboratory is endowed with considerable computer equipment for supporting experimental work. The equipment configuration is depicted in the Figure. Presently the research on datbase computer architecture has both short-term and long-term goals. For the short term, we are concentrating on a research program in multi-mini database management systems. For the long term, we are striving for an ideal database machine architecture. Let us describe briefly these two programs.

Research Program in Multi-Mini Database Management Systems

It is generally known that the use of a single general-purpose digital computer with dedicated software for database management to offload the mainframe host computer from database management tasks yields no appreciable gains in performance and functionality. Research is therefore being pursued to replace this software backend approach to database management with an approach which will yield good performance and new functionality.

The proposed research utilizes a system of six PDP 11-44s and one VAX 11/780, known as the test vehicle. Each of the PDP 11-44 computer systems consists of a primary memory box of 256K bytes, at lease one disk of 67M bytes and a memory-to-memory bus connected to the VAX 11/780 which has 1.5M bytes of primary memory and assorted peripheral devices. The configuration is intended to achieve concurrent operations among the six PDP 11-44s and their respective disks. The VAX 11/780 schedules the concurrent operations on the PDP 11-44s, communicates with other computer systems (known as front-ends) and serves as the control of the entire configuration (i.e., the database computer backend). For our study, the VAX 11/780 also interfaces with the systems programmers (See Figure).

The aim of the proposed research is to investigate whether, for the management of large databases, the use of multiple minicomputer systems in a parallel configuration is feasible and desirable. By feasible we mean that it is possible to configure a number of (slave) minicomputers, each of which is driven by identical database management software and controlled by a (master) minicomputer for concurrent operations on the database spread over the disk storage local to the slave computers. This approach to large databases may be desirable because only off-the-shelf equipment of the same kind is utilized to achieve high performance without requiring specially built hardware and because identical database management software is replicated on the slave computers. The approach makes the expansion of the capacity and concurrency of the database management system easy.

To study the feasibility, we intend to investigate the software architecture issues and hardware limitations of the master and slaves. We also intend to investigate the replicable software for the slaves. Since these slaves are to be operated concurrently with corresponding single-channel disks, we can investigate the effects of either single-query, multiple-database stream or multiple-query, multiple-database stream operations for performance improvement. To study the desirability, we intend to consider factors relating to the problem of capacity growth and cost effectiveness. The central issue may be whether we can realize a high-performance and great-capacity database management backend with the cheapest possible minis, large number of single-channel disks and replicable software cost-effectively.

The above program is termed short term, because it assumes only available hardware and present technology. Furthermore, both feasibility and desirability issues can be resolved in the time frame of two to three years.

Research Program on Database Machine Architecture

In the long run, the solution to large-capacity and high-performance database management may lie in special-purpose machines, known as the database computers. With the advent of LSI and VLSI, block access memories (such as magnetic bubbles and charge-coupled devices), and modified and improved online disk technology, it is perhaps time to replace complex and inefficient database management system software (DBMS) with innovative and cost/performance effective hardware solutions to very large database management. This search for an ultimate database machine architecture to provide large-capacity, high-performance and low-cost database management is the goal of this research undertaking.

Initially, the test vehicle is used to emulate the architecture of the database computer DBC (see References). As specialized hardware, DBC is designed to handle very large databases (say, beyond 10 billion characters), to perform database management operations effectively, and to yield high throughput unattainable by general-purpose computers with conventional database management software. Due to the complexity of database applications and the diversity of database management, analytical evaluation of database computer performance has been mostly based on broad assumptions and simplified settings. Although these evaluation results have been published (see References), they are too gross to be useful for the identification of performance bottlenecks caused by the components of the database computer. Without a closer examination and refined evaluation of potential performance bottlenecks of various hardware components, it is not possible to locate the strong and weak points of the database computer architecture. Consequently, improvements to the database computer design cannot be readily carried out.

By emulating the database computer on the test vehicle, experimental and realistic performance evaluation of DBC in supporting various database applications can be conducted. The evaluation of DBC in supporting various database applications can be conducted. The evaluations can focus on database management in terms of modes of operations (e.g., retrieval-intensive vs. update-intensive), models of databases (say, relational vs. CODASYL) and time and space constraints of the man/database interaction (e.g., real-time requirements with redundant data entries and integrity requirements with serial updates).

The information gained from the performance evaluation can then be extrapolated to reflect the performance of DBC. Furthermore, the information can be used to verify the analytical results found earlier.

The objectives of the research are: (1) To identify the performance evaluation techniques and methodology that are unique to database computer architecture. (2) To validate the analytical study of the DBC design against the experimental results conducted on the test vehicle. (3) To relate the findings on the emulation to the design details of DBC in particular and of database computers in general. And (4) To recommend modifications and improvements of the DBC architecture in order to support very large databases, attain high throughput and perform effective database management.

More specifically, the test vehicle is being configured to reflect the design of the three major components of DBC. The three components are (1) the database command and control processor DBCCP (i.e., the central processing and control unit of DBC) which executes the DBC commands, clusters the records for input and updates, schedules various subtasks and activities of DBC, and communicates with the host computer; (2) the online mass memory MM (i.e., the repository of the database) which provides high-volume and high-performance database management with track-in-parallel readout and write-in and content-addressability features; and (3) the structure memory SM (i.e., the index storage and processing unit) which enables access control information to be stored and processed readily so that accesses to the database can be restricted to relevant and authorized data, thereby narrowing the content-addressable space of the mass memory MM. Physically, the PDP 11-44s will be used to emulate the DBCCP.

Ultimately, this program will lead to the study of other database machine architectures and their relative cost and performance gain and loss compared to the performance and cost of the DBC design.

The test vehicle of 6 PDP 11-44s and one VAX 11/780 and assorted disks, terminals, tapes and printer will be funded in two stages. In the first stage, two PDP 11-44s with three disk devices, one tape station, four terminals and one printer have been funded for the 1980-1982 period. In addition, parallel transfer buses connecting the PDP 11-44s and VAX 11/780 are included. The equipment and its maintenance are funded jointly by the Digital Equipment Corporation, the Office of Naval Research and the Ohio State University.

Director of the laboratory is Douglas S. Kerr; and graduate students presently working in the laboratory are Jaishankar Menon, Ali Orooji, Tamer M. Ozsu and Paula Strawser

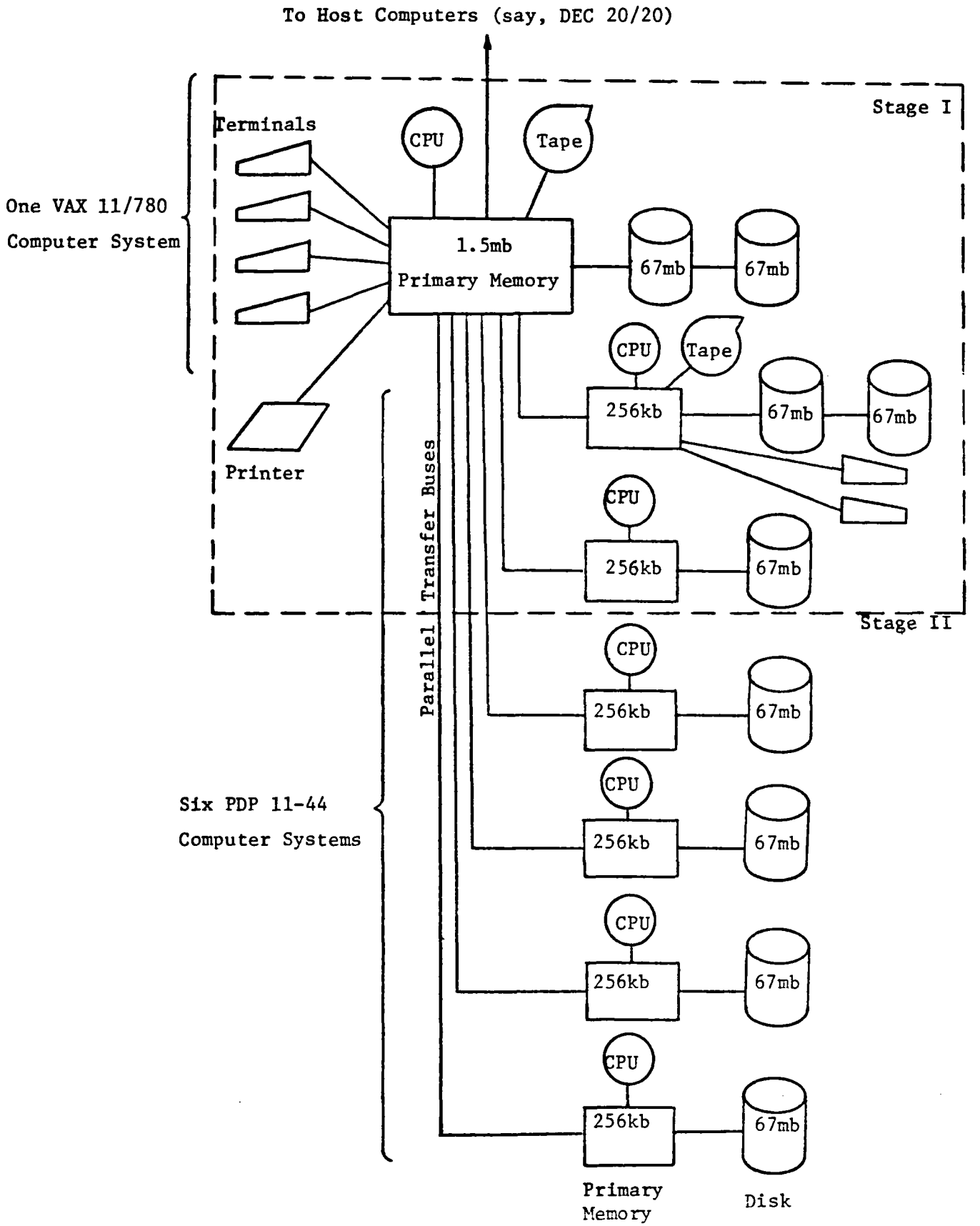


Figure: A Test Vehicle for Database Computer System Research

REFERENCES

On the Design of a Database Computer, known as DBC:

[1] Banerjee, J., R. Baum, and D.K. Hsiao. "Concepts and Capabilities of a Database Computer", ACM Transactions on Database Systems (TODS), 3, 4, December 1978, pp. 347-384.

[2] Banerjee, J. and D.K. Hsiao. "DBC - A Database Computer for Very Large Databases", IEEE Transactions on Computers, C-28, 6, 1979, pp. 414-429.

[3] Kannan, K., D.K. Hsiao, and D. Kerr. "A Microprogramming Keyword Transformation Unit for a Database Computer", Proceedings of the 10th Annual Workshop on Microprogramming, New York, October 1977, pp. 71-79.

[4] Hsiao, D.K., K. Kannan, and D. Kerr. "Structure Memory Designs for a Data Base Computer", Proceedings of ACM Conference '77 Seattle, October 1977, pp. 343-350.

[5] Kannan, K. "The Design of a Mass Memory for a Database Computer", Proceedings of the Fifth Annual Symposium on Computer Architecture, April 1978, pp. 44-51.

[6] Banerjee, J. and D.K. Hsiao "Parallel Bitonic Record Sort - An Effective Algorithm of the Realization of a Post Processor", Technical Report, The Ohio state University, (OSU-CISRC-TR-79-1), March 1979.

[7] Banerjee, J., D.K. Hsiao, and J. Menon. "The Cluster and Security Mechanisms of a Database Computer (DBC)", Technical Report, The Ohio State University, (OSU-CISRC-TR-79-2), April 1979.

[8] Hsiao, D.K. and J. Menon. "The Post Processing Functions of a Database Computer", Technical Report, The Ohio State University, (OSU-CISRC-TR-79-6), July 1979.

[9] Hsiao, D.K. and J. Menon. "Design and Analysis of Update Mechanisms of a database Computer (DBC)", Technical Report, The Ohio State Universtiy, (OSU-CISRC-TR-80-3), June 1980.

[10] Hsiao, D.K. and J. Menon. "Parallel Record-Sorting Methods for Hardware Realization", Technical Report, The Ohio State University, (OSU-CISRC-TR-80-7), July 1980.

On DBC's Capability in Supporting Existing Database Applications with Improved Throughput:

[11] Banerjee, J., D.K. Hsiao, and F. Ng. "Database Transformation, Query Translation and Performance Analysis of a New Database Computer in Supporting Hierarchical Database Management", IEEE Transactions on Software Engineering, SE-6,1 January 1980, pp. 91-109.

[12] Banerjee, J., and D.K. Hsiao. "A Methodology for Supporting Existing CODASYL Databases with New Database Machines", Proceedings of ACM Conference '78, Washington, D.C., December 1978.

[13] Banerjee, J. and D.K. Hsiao. "The Use of a Database Machine for Supporting Relational Databases", Proceedings of the 5th Annual Workshop on Computer Architecture for Non-numeric Processing, Syracuse, N.Y., August 1978.

[14] Banerjee, J. and D.K. Hsiao. "Performance Study of a Database Machine in Supporting Relational Databases", Proceedings of the 4th International Conference on Very Large Data Bases. Berlin, Germany, September 1978.

On General Treatment on Database Computers:

[15] Baum, R.I. and D.K. Hsiao. "Database Computers - A Step Towards Data Utilities", IEEE Transactions on Computers, C-25, 12, december 1976, pp. 1254-1259.

[16] Hsiao, D.K., "Database Computers", Advances in Computers, Academic Press, v. 19, June 1980, pp. 1-64.

[17] Hsiao, D.K., "The Role of Database Computer Prototypes", to appear in the Proceedings of the 13th International Hawaii Conference on System Science, Honolulu, Hawaii, January 1980.

[18] Hsiao, D.K. and S.E. Madnick. "Data Base Machine Architecture in the Context of Information Technology Evolution", Proceedings of the 3rd International Conference on Very Large Data Bases, Japan, October 1977, pp. 63-84.

[19] Hsiao, D.K., "Future Database Machines", Future Systems, Infotech State of the Art Report, November 1977, (U.S. distributors: Auerback Publisher, Ltd.) pp. 307-330.

[20] Banerjee, J. and D.K. Hsiao. "Data Network - A Computer Network of General-purpose Front-end Computers and Special-purpose Back-end Data Base Machines", Proceedings of the International Symposium on Computer Network Protocols, Leige, Belgium, February 1978.

[21] Hsiao, D.K., "Database Machines are Coming: Database Machines are Coming! - A Guest Editor's Introduction", Computer Magazine, 11,3, 1979, pp. 7-9.

[22] Kerr, D. "Database Machines with large Content - Addressable Blocks and Structural Information Processors", Computer Magazine, 11,3, 1979, pp. 64-79.

Database Machine Activities at The University of Wisconsin

Haran Boral
David J. DeWitt

Computer Sciences Department
University of Wisconsin
Madison, Wisconsin

Abstract

In this paper we summarize recent database machine research activities at the University of Wisconsin, outline current research projects, and describe our future research plans.

1. Introduction

Rather than using this forum to discuss one current database machine research project at the University of Wisconsin (which the reader might or might not be interested in), we decided to present brief descriptions of several completed and ongoing projects. We hope that this format will serve the casual reader by permitting him/her to quickly find out what we have been doing, while, at the same time, providing the database machine researcher with references to those results that have not yet found (or never will find) their way into the literature.

The research projects described in this document largely grew out of the DIRECT project. Several people have worked on DIRECT and participated in some of the projects. Although this document is authored by only two of us this should not be understood by the reader to mean that we did all the research work. Rather, it was a group effort. Jim Goodman and Randy Katz joined our department after most of the work reported here began. We included a description of their recent database machine activities.

The results of some recently completed research projects are presented in Section 2. Section 3 describes some of our ongoing database machine activities. In Section 4, we descibe some activities that we have planned for the future.

2. Recent Research Results

In this section we present the results of some recently completed database machine research projects.

2.1. Parallel Algorithms for the Execution of Relational Database Operations

In [BORA80] we present and analyze algorithms for parrallel processing of rela-

---

This research was partially supported by the National Science Foundation under grant MCS78-01721, the United States Army under contracts #DAAG29-79-C-0165 and #DAAG29-80-C-0041, and the Department of Energy under contract #DE-AC02-81ER10920.

tional database operations in a DIRECT-like multiprocessor framework. To analyze the alternative algorithms, we developed an analysis methodology which incorporates I/O, CPU, and message costs. The paper presents parallel algorithms for sorting, projection, join, update, and aggregate (both scalar aggregates and aggregate functions) operations. Although some of these algorithms have been suggested previously in the literature, we generalized each in order to handle the case where the total amount of memory in the processors is not sufficient to hold all the data operated on.

One (rather obvious) goal of this research was to compare alternative parallel algorithms for relational database operations. However, the primary goal of this research was to evaluate these alternative algorithms so that we could begin the design of a new database machine using an algorithmic approach (top down) rather than an architecture-directed approach [DEWI81].

This research was conducted by H. Boral, D. DeWitt, D. Friedland, and W.K. Wilkinson.

2.2. Performance Evaluation of Four Associative Disk Designs

While our research on parallel algorithms evaluated alternative techniques for processing complex relational operations, it did not address the problem of efficiently processing selection operations. In [BORA82] we develop models of alternative associative disk architectures and present the results of an event-driven simulation based on this model. The designs analyzed are Processor-per-Track (PPT), Processor-per-Bubble-cell (PPB), Processor-per-Head (PPH), and Processor-per-Disk (PPD). We considered the effects of a number of factors, including output channel contention, availability of index information, impact of mark bits, and channel allocation policy on the performance of these machines. Our results indicate that while in the general case the PPT organization is best, the performance of the PPB design is remarkably good considering the slow speed of bubble memory chips. Also index information can be used by both the PPH and PPD organizations to improve their performance to a level almost comparable to PPT.

This research was conducted by H. Boral, D. DeWitt, and W.K. Wilkinson.

2.3. Using Data-Flow Techniques in Database Machines

After the design of DIRECT was completed (1977) we began to examine different strategies for allocating processors to tasks in order to maximize performance. The reults of this research are presented in [BORA81a].[1] The main result of this work is that dynamic scheduling of processors using a data-flow strategy can provide significant improvements in performance. However, this increase in performance is achieved only through a significant increase in control costs. A database machine such as DIRECT with its centralized controller is not appropriate. In fact, it was shown that the cost of controlling the processors in DIRECT dominated the execution time even for the static strategies. We therefore began to examine new ways of organizing database machines that could efficiently support data-flow query processing.

Using the results of our research on parallel algorithms ([BORA80]) and our

1. This research was completed in 1979.

results on associative disk design ([BORA82]) we have finished an initial design of a database machine, named FLASH, that uses data-flow techniques. This machine uses broadcasting as the basic building block for all relational algebra algorithms, and combines "on-the-disk" processing of selection operations with "off-the-disk" processing of complex relational operations.

We have recently been critical of other researchers for designing "yet another" paper database machine. For that reason we have not prepared a document describing FLASH.[2] We are beginning to study different ways of comparing the performance of database machines. We intend to demonstrate (to ourselves) that FLASH significantly outperforms other proposed designs and that its performance justifies its complexity before we trumpet its design.

This work was performed by H. Boral and D. DeWitt.

2.4 Performance Evaluation of Database Machine Architectures

While many database machine designs have been proposed, each proposal is usually vague about its performance with respect to a given data management application or other database machine. Furthermore, no one has attempted a comprehensive performance evaluation of different database machine architectures. In [DEWI81][3] we develop a simple analytical model of the performance of a conventional database management system and four generic database machine architectures. This model is then used to compare the performance of each type of machine with a conventional DBMS for selection, join, and aggregate function queries. We demonstrate that 1) no one type of database machine is best for executing all types of queries, 2) for several classes of queries, certain database machine designs which have been proposed are actually slower than a DBMS on a conventional processor, and 3) increases in complexity of hardware generally do not result in corresponding increases in performance.

This research was conducted by D. DeWitt and P. Hawthorn of Lawrence Berkeley Laboratory.

2.5. Implementation of the Database Machine DIRECT

DIRECT is a multiprocessor database machine designed and implemented at the University of Wisconsin. The initial design was completed in 1977. In the spring of 1980 we had a first working version of DIRECT running on 5 LSI 11/03 microcomputers. In [BORA81c] we describe our experiences with the implementation of DIRECT.

The paper begins with a brief overview of the original machine proposal and how it differs from what was actually implemented. We then describe the structure of the DIRECT software. This includes software on host computers that interfaces with the database machine; software on the back-end controller of DIRECT; and sofware executed by the query processors. In addition to describing the structure of the software we attempt to motivate and justify its design and implementation. The paper also discusses a number of implementation issues (e.g., debugging of the code across several machines) and concludes with a list of the

---

[2] An outline of FLASH is provided in [BORA81b].

[3] See [HAWT82] for an earlier attempt at comparing the performance of specific database machines.

"lessons" we have learned from this experience.

The implementors of DIRECT are H. Boral, D. DeWitt, D. Friedland, N. Jarrell, and W. K. Wilkinson.

2.6 Duplicate Record Elimination in Large Data Files

This paper [FRIE81] addresses the issue of duplicate elimination in large data files in which many occurrences of the same record may appear. The paper begins by presenting a comprehensive cost analysis of the duplicate elimination operation. This analysis is based on a combinatorial model developed for estimating the size of intermediate runs produced by a modified merge-sort procedure (in which duplicates are eliminated as encountered). We demonstrate that the performance of this modified merge-sort procedure is significantly superior to the standard duplicate elimination technique of sorting followed by a sequential pass to locate duplicate records. These results can also be used to provide critical input to a query optimizer in a relational database system.

This research was conducted by D. Friedland and D. DeWitt.

2.7 Concurrency Control and Recovery in Transaction Processing Systems

While most of our database machine efforts have concentrated on the design of database machines for processing "large" queries (those that touch large amounts of data and take a long time to execute), we have recently begun to explore database machines for processing large volumes of "small" transactions (e.g. "debit-credit" transactions). In [WILK81] we examine issues of concurrency control and recovery in a database machine that consists of user nodes (intelligent terminals) connected to database server nodes via an ethernet communications device.

The central idea in this work is a "passive" concurrency control mechanism which makes use of the broadcast nature of the communications medium. By eavesdropping on requests that correspond to database accesses by the user nodes to the database server nodes, a single concurrency control node can perform conflict analysis for the entire system without explicit lock messages.

We present two algorithms: a passive locking and a passive non-locking algorithm, and show that they are robust to communications and processor failures. Simulation results indicate that the passive schemes have very low overhead and perform better than corresponding distributed algorithms (both locking and non-locking) in this environment. Also, we show that the cost of the recovery protocol necessary to ensure atomic commit at all sites (i.e., the distribtued two-phase commit protocol) is high and, in many cases, overshadows the cost of concurrency control.

2.8. VLSI Implications of Database Systems

Our research has also been directed at exploiting the vast capabilities available with the arrival of VLSI technology. By the middle of the decade, it will be possible to build single-chip processors, including memory, with performance capabilities achievable only by main-frame computers today. While this enormous computational power will be available at a modest cost, the processor requirements resulting from the support of a large modern database system are also very substantial. And though it may be possible to build a single, supercomputer from

many VLSI components, a much more cost-effective approach will probably be to construct a system using large numbers of single-chip computers.

We have postulated a processor model, incorporating VLSI constraints. We have concluded that a single integrated circuit type, containing a more-or-less conventional von Neumann processor with a significant amount of memory, can be employed to construct a machine with great generality in capability but which will perform database operations in a highly efficient manner. We have analyzed a number of possible topologies for a collection of these processors, assuming processors approximately in proportion to the number of disk heads available. We have studied the performance of a number of topologies for the most important relational database operations.

In [GOOD80a] we examined the duplicate elimination problem resulting from the relational projection operation. We concluded that the topology strongly influences the effectiveness of various algorithms, and that for each topology there is an optimal algorithm. Comparing these pairs, we concluded that the best topology we could think of was a variation of the X-Tree Hypertree interconnection [GOOD81a], containing a perfect shuffle interconnection at the leaves of the tree.

In [GOOD81b] we studied parallelized version of traditional join algorithms in a similar manner, finding that performance was deficient. In [GOOD80b] we considered hashing techniques, inspired by [BABB79], to implement the equi-join algorithm, and showed that these can be used with the Hypertree interconnection to implement extremely efficient join algorithms.

This section was written by J. Goodman.

2.9. VLSI Implementation of a Join Chip

Randy Katz designed and implemented a custom VLSI chip for processing equi-joins. The chip accepts three serial bit streams: one each for the inner and outer tuples of the nested loops join method, and a bit mask. A flag is set if the tuples are equal under the mask. The concatenation of the tuples is formed on-board the chip, and removed at twice the input bit rate. The total design was laid out in under three weeks using the Mead-Conway design methodology, and consists of approximately 1500 transistors. The chip is in fabrication, and should be available for testing soon. Clearly we can make no claim as to the utility of the chip for processing joins, but the application of VLSI technology to database processing is certainly an intriguing area of research.

3. Current Activities

3.1. DIRECT Implementation

The implementation of DIRECT is proceeding in several directions. In July 1981, we finished implementing the INGRES update operations utilizing the parallel update algorithms presented in [BORA80]. We have recently begun implementing scalar aggregates and aggregate functions. This will complete an initial version of DIRECT except for implementation of concurrency control and recovery (something we have not yet figured out how to do).

The host and back-end software for DIRECT is presently running on a PDP 11/40 that runs Version 6 UNIX and INGRES. This machine is on its very last legs and

will be retired in early November 1981. It will be replaced with a VAX 11/750 which has been ordered. Adquisition of this new machine to run the DIRECT back-end and controller software is a mixed blessing.

A PDP 11/40 outperforms an LSI 11/23 (of which eight are used for query processors in the present DIRECT implementation) only marginally. Thus INGRES (read "DIRECT host software") is incredibly slow. Replacing the PDP 11/40 with a VAX 11/750 will significantly enhance respone time and will permit us to perform a number of benchmarks on DIRECT. On the other hand, VAX-INGRES runs as two processes and is written in Verison 7 C. Thus, we must redo all of our modified INGRES software and some of the DIRECT back-end controller software. This will probably take at least one-man year of effort.

Another change in the DIRECT configuration is that we are presently replacing the nonDMA parallel word interfaces that have been used to connect the LSI 11/23s to the 11/40 with DMA, "ethernet-like" interfaces. Since these interfaces permit DMA I/O, DIRECT performance should be enhanced.

3.2. Research Activities

3.2.1. Statistical Database Machines

We have recently begun to investigate the possibility of constructing a prototype database machine intended to improve performance for users of very large statistical databases. Currently we are extending the aggregate algorithms presented in [BORA80] to cover a larger class of operations (e.g., median, cross tabulation). We are also investigating the effect of different data layout strategies, data compression, and mass storage technologies on each of these algorithms.

3.2.2. Development of Database Machine Performance Evaluation Tools

Most of the research in the database machine area is devoid of performance analysis. There are no analytical or simulation tools that are suitable for measuring the performance of a new design or comparing the performance of two differnet designs. We have recently begun to develop such tools. Our basic idea is to provide the user with an environment that includes a number of building blocks, an easy way to specify the machine organization and workload using the building blocks, and the ability to interface both simulated and analytic descriptions of the behavior of particular components.

3.2.3. Filter Design

In [BORA82] we evaluated alternative associative disk designs under the assumption that the processor used could execute any arbitrarily complex selection condition fast enough to keep up with the rotational speed of the mass storage device. If a conventional microprocessor is used as the basis of the filtering element, such an assumption is probably valid only for simple selection operations and not for complex (multiple tests on multiple attributes) selection operations.

An alternative is the use of a specialized processor that executes selection queries in the form of a finite state automaton (FSA). Together with Francois Bancilhon of INRIA we are presently conducting a performance evaluation of hardware versus software filtering.

3.2.4. Parallel Sorting

While [BORA80] included the evaluation of several external parallel sorting algorithms, this evaluation did not accurately measure the loads placed by the algorithms on the underlying mass storage system. We are presently extending the parallel sorting models presented in [BORA80] to model I/O activity and loads.

4. Future Activities

In Section 3 we discussed a number of ongoing projects, in particular statistical database machines and performance evaluation tools for database machines. Both of these projects have only recently been started and thus almost qualify for a mention in this section. At this point we have no plans for beginning new research work.

One event that will affect our future work is the award by NSF to the Computer Sciences Department at Wisconsin of one of the NSF Experimental Computer Science Research Grants. This grant will be used to construct a large multicomputer. The facility, when operational (about 1983), will consist of between 50 and 100 processors (each processor will be a 32-bit processor with about 1 Megabyte of main memory and have the performance of a small VAX) and a number of mass storage units. The processors will be connected together with frequency-agile communications devices that will permit us to efficiently emulate a large number of interconnection topologies.

Our plans at this point are to use this "supercomputer" for two different purposes. First, we intend to implement FLASH and whatever statistical database machine we design. Second, we would like to implement various primitives to be used by simulations, emulations, or other tools for performance analysis of database machines as described in Section 3.2.2.

5. References

[BABB79] E. Babb, "Implementing a Relational Database by Means of Specialized Hardware," ACM Transactions on Database Systems, Vol. 4, No. 1, March 1979, pp. 1-29.

[BORA80] H. Boral, D. J. DeWitt, D. Griedland and W. K. Wilkinson, "Parallel Algorithms for the Execution of Relational Database Operations", University of Wisconsin Computer Science Technical Report no. 402, submitted for publication, October 1980.

[BORA81a] H. Boral and D. J. DeWitt, "Processor Allocation Strategies for Multiprocessor Database Machines", ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, pp. 227-254.

[BORA81b] H. Boral, "On the Use of Data-flow Techniques in Database Machines", Ph.D. Dissertation, University of Wisconsin - Madison. Also University of Wisconsin Computer Science Technical Report no. 432, May 1981.

[BORA81c] H. Boral, D. J. DeWitt, D. Friedland, N. Jarrell, and W. K. Wilkinson, "Implementation of the Database Machine DIRECT", University of Wisconsin Computer Science Technical Report no. 442, submitted for publication, August 1981.

[BORA82] H. Boral, D. J. DeWitt, and W. K. Wilkinson, "Performance Evaluation of Four Associative Disk Designs", to appear in Journal of Information Systems, Vol. 7, No. 1, January 1982.

[DEWI81] D. DeWitt and P. Hawthorn, "Performance Evaluation of Database Machine Architectures", Proceedings of the 1981 Very Large Database Conference, September 1981.

[FRIE81] D. Friedland and D. J. DeWitt, "Duplicate Record Elimination in Large Data files", University of Wisconsin Computer Science Technical Report no. 445, submitted for publication, August 1981.

[GOOD80a] J. R. Goodman and A. M. Despain, "A Study of the Interconnection of multiple Processors in a Database Environment", Proceedings of the 1980 Conference on Parallel Processing.

[GOOD80b] J. R. Goodman, "An Investigation of Multiprocessor Structures and Algorithms for Data Base Management", Ph.D. Dissertation, University of California, Berkeley. Also, Memo No. UCB/ERL M81/33, University of California, Berkeley, May 1981.

[GOOD81a] J. R. Goodman and C. H. Sequin, "HYPERTREE: A multiprocessor interconnection topology", to appear in IEEE Transactions on Computers, December 1981.

[GOOD81b] J. R. Goodman, "Implementing Join Algorithms Using the Hypertree Interconnection Network", paper in preparation.

[HAWT82] P. Hawthorn and D. J. DeWitt, "Performance Analysis of Alternative Database Machine Architectures", to appear in IEEE Transactions on Software Engineering, January 1982.

[WILK81] W. K. Wilkinson, "Database Concurrency Control in Local Broadcast Networks", Ph.D. Dissertation, University of Wisconsin - Madison. Also University of Wisconsin Computer Science Technical Report no. 448, May 1981.

The Intelligent Database Machine

Michael Ubell
Britten-Lee, Inc.
1919 Addison St. #304
Berkeley, Ca. 94702

Database machines have been discussed in computer literature for many years. We are just now beginning to see the first of these special-purpose machines reach the market place. With the exception of the ICL CAFS machine [MITC76], which has been shipped in small quantities for several years, no database machines have left the laboratory and few have even left paper. The Intelligent Database Machine is the first of these special-purpose computers to reach the general marketplace. It is also the first to be designed based on the idea that fast database management can be provided at a low cost. This paper describes the major architectural features of the system.

The Intelligent Database Machine (IDM) provides a complete relational database management system combined with special-purpose hardware to provide a high transaction rate at a low cost. The IDM is a backend machine and requires some host intelligence to handle communication with the user. The architecture of the machine is designed around the idea that there are a few well-defined tasks which a relational DBMS does very often in processing transactions. These tasks can be microcoded in a specially designed processor, the Database Accelerator. It is fast yet inexpensive since the amount of microcode is very limited. The memory system is also designed around data management tasks and provides high-speed access for these tasks using standard-speed memory components, again keeping the cost low. Another central idea in the IDM is that a database management system is more efficient in a dedicated environment.

The IDM software implements most of the facilities of a relational database management system except for actual source-language parsing. These include validity checking, protection, query optimization, access-path selection, logging and crash recovery. The parsing of the source language query is done on the host system. The division of tasks between the host and the IDM at this level provides for the most efficient use of a backend system without restricting the user interface to a particular language. If the interface were at a lower level, say the "fetch a record" level of the Relational Storage Interface in System R [ASTR76], the host would have to access the IDM almost as much as it would have to access a disk if it were to do the same work itself. The issues of recovery and concurrency control also become more difficult at this level. For example, to do concurrency control in a reasonable manner the host would have to declare how much of a relation (or file) it was going to access or update so that the backend system could determine whether to lock the whole relation or just lock pages as they are touched. Since the IDM does all the access-path selection it can determine this without the need of directives from the host.

The IDM software takes advantage of a dedicated system environment. Operating systems have traditionally tried to manage system resources based on a guess as to what the application programs are likely to do next. This does not work well for database management systems because their I/O access is very different from most programs [HAWT79]. Only the DBMS code "knows" how to optimally schedule disk accesses. What there is of an operating system in the IDM can be best

described as an anarchistic operating system. Processes do their own disk I/O except when it is better to have it queued for latter. They request to stop running at appropriate points after they have used up their time slice or have to wait for queued I/O. These two features allow for optimal disk read ahead in that a process will never read a page and then not process it before giving up the CPU. If the process needs I/O which will take some time (a long seek) it will schedule itself out and be woken up when the I/O is finished. In this way the IDM can overlap the CPU and I/O when it will improve the performance of the system. The operating system keeps track of where each disk arm is and where on disk each process needs to access the disk so that it may schedule intelligently. The DBMS software on the other hand is in complete control of the in-memory disk buffering.

The IDM hardware consists of a high-speed bus and 6 different boards:

Database Processor
Database Accelerator
Channel
Memory Timing and Control
Memory Storage
Disk Controller

The IDM 500 has a 16-slot bus. Slots can be filled with extra memory or channels as required for the application. One or more hosts communicate with each IDM Channel. The channel implements a communication protocol with the host and buffers data coming into and leaving the IDM. It also translates between host data types and the data types used within the IDM. These two functions shield the rest of the system from many interrupts and allow it to see just a uniform command stream. The channel consists of a microprocessor, memory and hardware to implement 8 serial (rs232c) or one parallel (IEEE-488) interfaces. Once a command is accumulated in the channel it notifies the Database Processor which has it transferred to the main memory of the system for the rest of its processing.

The Database Processor is responsible for controlling the other boards in the system and also implements most of the functionality of the system. It uses a standard 16-bit microprocessor chip (Zilog Z8000). It must execute about 45,000 lines of code written in a high-level language to implement the relational DBMS. Part of this code emulates the Database Accelerator and the system can run without the Accelerator, but at a slower rate. The Database Accelerator is made with standard bit-slice chips (Signetics 2900 series) and contains 4 kilo-words of microstore. The routines in the Accelerator are mainly those involved with looking at data within a page from the database, in other words, the inner loop of most processing activities. Since the processing strategy is often single-threaded, the Accelerator also has the ability to look at data while it is being read into memory. Once a command is given to the Accelerator it can schedule subsequent reads from disk and look at data until it has found enough qualifying tuples or looked at enough pages as determined by the Database Processor.

The memory system of the IDM provides for up to 3 megabytes of disk buffering and additional space for user processes. The memory system can correct one-bit errors and detect two-bit errors. This, together with address and data parity on the IDM bus, insures data integrity throughout the system. The memory has two modes of operation: one is byte- and word-oriented for the Database Processor and the other, faster mode, is block-oriented for the Accelerator and Disk Con-

trollers.

The IDM Disk Controller interfaces to up to four Storage Module Device disk drives. The Disk Controller is responsible for the reliable transfer of data to and from the disk. It implements burst-error correction and retry without any intervention from the Database Processor. There can be up to 4 disk controllers on the IDM with a total of 16 disks. While this limits the storage capacity to about 9 gigabytes with current disk technology, the IDM can address up to 32 gigabytes of disk storage when disk drives become denser.

We are just beginning to collect performance information on the IDM. The Database Accelerator is not yet in production, but some results against a 2-megabyte database run without the Accelerator are available. Without the Accelerator the IDM can run in the 3-5 transactions/sec range. To retrieve a record through a two-level index takes .15 seconds. A simple append through unique index can be done in .2 seconds. A "debit-credit-log" transaction, involving two replaces and an append, takes .38 seconds. The speed and proportion of usage of the Accelerator depend on the task being done. The Accelerator runs 15-30 times faster than the Database Processor and the system spends from 50-90% of its time in the Accelerator emulation routines; in addition the Accelerator reduces latency by analyzing data as it comes off the disk. With the Accelerator the IDM can be expected to reach 30 transactions/sec in some applications. Another way the IDM improves system performance is by off-loading the host system. In initial tests one large main-frame computer manufacturer estimated that the IDM could off-load 1/3 of their computer at 3 times the speed for their application.

The end-user price of an IDM is about $70,000. The IDM is not a complete system and must be packaged with a host computer. With a small host computer and disks a complete system could be as little as $120,000. Complete DBMS software packages cost $30,000-50,000 for minicomputers and typically only run quickly on large minicomputers. Such a software system and computer would cost about $250,000. Off-loading the DBMS from the host frees the host for other work. The IDM provides better functionality and greater speed at half the cost.

REFERENCES

ASTR76 Astrahan, M. M., et. al. "System R: relational approach to database management", ACM Trans. Database Systems. V1 N2 (June 1976).

HAWT79 Hawthorne, P.B., "Evaluation and enhancement of the performance of relational database management systems", Ph.D. Dissertation, University of California, Berkeley, California, 1979.

MITC76 Mitchell, R.W. "Content addressable file store," in Proc. Online Database Technology Conf., April 1976.

A Methodology for the Determination of Statistical Database Machine Performance Requirements

Paula Hawthorn
Lawrence Berkeley Laboratory
Berkeley, California

1. Introduction

This paper describes work in progress on a research topic in the design of a statistical database machine. A statistical database is one that is used mainly for the management and analysis of large amounts of data, where the data are accessed and updated in large quantities. Such databases are commonly used in applications where there is statistical analysis of the results of experiments or surveys, hence the name "statistical databases". Applications include sociological and epidemiological studies of data derived from census records or surveys; analysis of economic data for forecasting and modeling; management information systems; and the analysis of data resulting from instrumentation of physical measurements (e.g., temperature, flow, etc.).

Data management research has traditionally focused on business applications, where the problems center around those found in business databases: e.g., transaction management for debit/credit consistency, backup and recovery mechanisms, concurrency control, protection, and so on. Although those problems exist in statistical databases, the focus shifts: in a statistical database the emphasis must be on the efficient management of large amounts of data and the calculation of summary quantities such as means and medians. The need for compression [EGGE81], for user models of an often unwieldy amount of data [CHAN81], for special database operations [JOHN81] are often more important in a statistical database than in business databases.

In this paper we discuss the determination of the cost/performance ratio necessary for a statistical database machine to be a viable alternative to current computing facilities. By database machine, we mean a specialized computing system dedicated to data management and using especially designed hardware to increase the performance of the system. It has been shown that database machines can be built which, because they are designed to perform a single function, are usually more cost-effective than general purpose machines performing the same function [DEWI81]. Such database machines commonly operate as back-end systems, where the front-end computer handles general-purpose computing tasks and sends all data management tasks to the back-end system. Business database machines exist [EPST79, BABB79] but there are no statistical database machines.

In [HAWT81] it is shown that statistical databases have ideal performance characteristics for implementation within a back-end system that is dedicated to the management of statistical data. In particular, users of a statistical database tend to reference large amounts of data with a single instruction (e.g., "find

---

This work was supported by the Applied Mathematics Sciences Research Program of the Office of Energy Research of the U. S. Department of Energy under Contract No. W-7405-ENG-48.

the mean and variance of this field", where there are 10M bytes of data in that field) so that the overhead to process the command in the front-end system is much less than the time required to execute the command in the back-end machine. It may also be true that statistical database machines can be built to cheaply provide very high performance through the use of parallel multi-processors for computation necessary for statistical analysis. An investigation of the design of a statistical database machine is proceeding at the University of Wisconsin-Madison by the designers of the more general-purpose database machine, DIRECT [DEWI78].

The study described in this paper is to determine the performance and functionality necessary for a statistical database machine, in order to avoid the situation of designing a candidate machine, implementing a prototype of it, and later finding that it does not meet the needs of the target community. This will be the first such study to appear in the literature on database machines. [DEWI81] points out the habit of database machine designers of designing a machine from the perspective of using interesting hardware, rather than the perspective of solving general problems in data management performance. The methodology described may be useful to others who are contemplating special-function hardware design.

To determine whether a new architecture is better than the architectures currently in use, models of the current architectures must be developed. Section 2 defines four models of computer architecture which can be used for the management of statistical data. These models are: (1) a large central system, (2) a network of minicomputers, (3) a network of minicomputers where some of the computers are dedicated to data management (that is, using software modifications but no specialized hardware), and (4) a network of minicomputers with back-end database machines attached. Cost/performance parameters are defined. These parameters can be measured to determine what the cost/performance of a database machine would need to be to make (4) a reasonable alternative to (1), (2) and (3). Section 3 defines the study that will take place. Section 4, the conclusion, describes further research to be based on the results of the study.

2. Defining Parameters to be Measured

The purpose of this study is to determine the necessary cost/performance ratio of a statistical database machine such that the use of such a machine is a viable alternative to current computing facilities. Such a study must be done before the machine is designed because the design of such a machine can be highly influenced by that ratio. It is apparent that database machines can be designed with different cost/performance ratios: the Britton-Lee machine, the IDM 500, sells for about $70,000; it is a single instruction, single data stream machine. The ICL machine, CAFS, sells for close to $1,000,000. It is a single instruction, multiple data stream machine, and for some applications is extremely fast [HAWT81].

Therefore the problem is to define the current computing environment, to define parameters to measure to determine the cost/performance ratios within the current environment, to measure those parameters, and to forecast changes to those parameters due to shifting environments. Then the necessary cost/performance ratio can be determined. We will first define the current environment, then the parameters to be measured. The measurement techniques and forecasts will be discussed in later sections.

2.1 Models of Current Systems

It must be noted that statistical database management systems are not yet widely used. Instead, scientific researchers use several different methods to analyze and manage data: some use statistical packages that have rudimentary facilities for data management; some use data management systems that have no statistical analysis facilities and must move the data from the data management system, forming temporary files on which to run statistical analysis programs; some simply use the file management facilities of the operating system, writing their own programs, using statistical libraries, to perform data analysis. There is some use of statistical data management systems but that use is not yet widespread. Therefore, when we refer to the "current working environment" we mean the current use of the system for statistical data management and analysis, not the current use of statistical data management systems per se.

The following four models are not the only alternatives in computing environments for statistical analysis of large data sets. They are, however, representative of current or easily possible environments. The proposed models for computer systems are listed below.

2.1.1 Large Central Systems

One of the most common computer architectures in use currently is the large, central system. This is shown in Figure 1. In the Large Computer System (LCS) the computer is timeshared among many batch and interactive users. The major secondary storage devices are disks, with other devices, such as tape drives and/or automatic tape libraries attached to the system.

2.1.2. Minicomputers

A second approach to providing computing facilities is to purchase separate minicomputer systems for separate applications. This is shown in Figure 2. The minicomputer network that we are modeling is one based on our experience at Lawrence Berkeley Laboratory. In systems common at LBL, there is a loose connection between some of the minicomputers. Often the minicomputers are bought for different applications and by different departments according to varying requirements, so the computers tend to be made by different manufacturers; even those made by the same manufacturer may be running different operating systems. Additionally, a given application may support a number of homogeneous computers that are tightly coupled with each other, but loosely coupled to the rest of the laboratory. There are also applications where the minicomputer system is stand-alone due either to protection and privacy requirements, or because there is no perceived need to net it to other minicomputer systems.

Figure 1. Central System

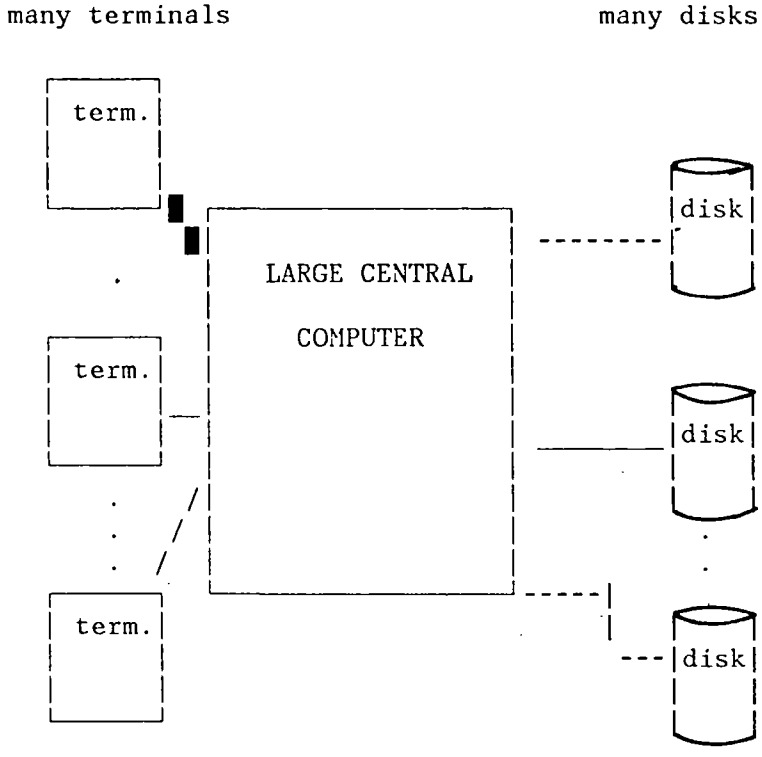
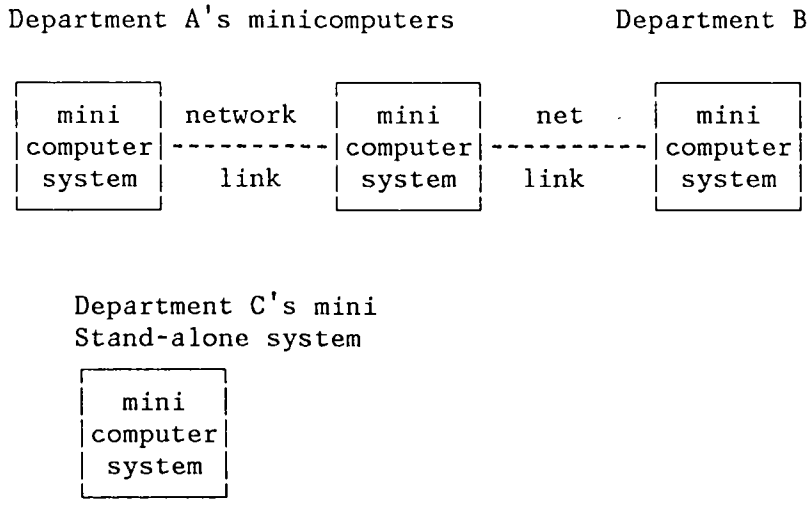


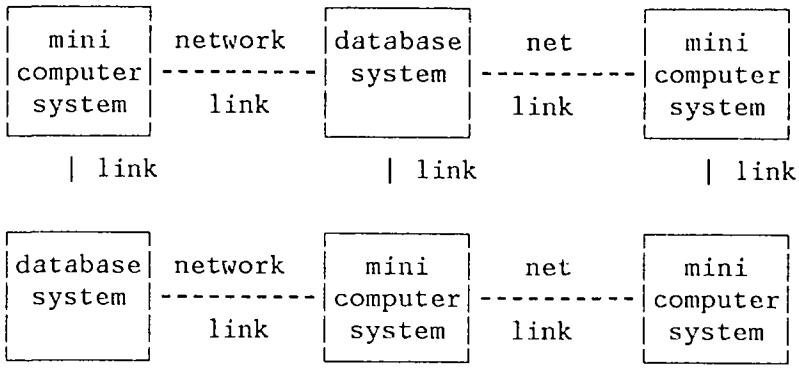
Figure 2. General-Purpose Minicomputers



2.1.3 Networks with Dedicated Database Systems

The two systems described above are those that commonly exist; the third system is one that may be very useful. It is a network of minicomputers, where some of the minicomputers are dedicated strictly to running the data management system. This is shown in the following figure.

Figure 3. Network with Specialized Machines



The dedicated systems would include no specialized hardware; however, operating systems would be modified to more efficiently interact with the data management system. There are several advantages with the above approach. These are listed in the following table.

Table 1. Advantages of Dedicated Systems

1. Operating system Efficiency
   It has been observed that if a computer system is running only a data management system, the operating system can make use of special information available from the data management system [HAWT79, STON81]. The incorporation of such knowledge into the operating system will increase the efficiency of the system as a whole.

2. Data Sharing
   Centralizing the databases on a few computer systems means that the users can easily share data.

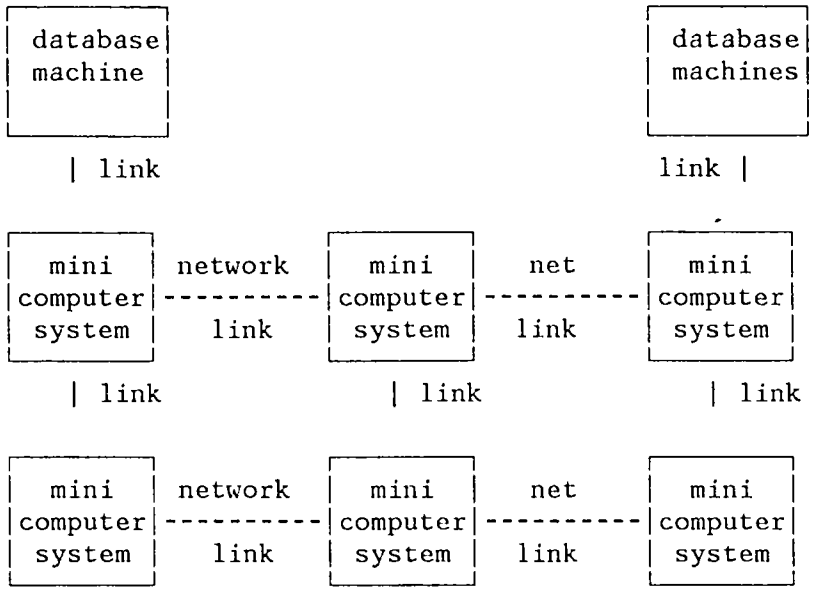
3. Lower Cost of Software
   The data management system resides on only a few machines, making it easier to maintain. Also, vendors charge per CPU, so the fewer systems there are that have the data management system installed, the lower the cost.

The dedicated system architecture is to be modeled because it is a reasonable alternative to the expense of developing specialized hardware.

2.1.4. Statistical Database Machines

The fourth architecture to be explored is the use of specially designed database machines. It is proposed that these machines be back-end systems; their use is shown in Figure 4.

Figure 4. Network with Database Machines



The above architecture includes a network of minicomputer systems, some of which are attached to statistical database machines.

2.1.5 Model Choice

The following section justifies the choice of the particular models above. The larger central system (LCS) and the network of minicomputers were chosen because those two models are in common use. The use of dedicated minicomputers for data management (model 3) is included because such systems may have a lower cost/performance ratio than systems that include database machines, and should therefore be included in comparisons. We exclude making an LCS a dedicated database machine because the expense of an LCS is such that it appears to be prohibitive to dedicate one to data management.

The fourth model, the network of minicomputers with statistical database (SDB) machines attached to some, is included because that appears to be the most reasonable architecture to develop at this time. In order to achieve high efficiency at low cost, the SDB machines are back-end systems that cannot directly communicate with users but must be front-ended by other computers. A design where the front-end system is a minicomputer is chosen, rather, than using an LCS as the front-end, for development purposes: for communications between the two systems, it is sometimes necessary to have dedicated use of the front-end system. The cost of using an LCS system for that purpose would be prohibitive at this time.

2.2 Parameter Definition

The basis for this study is strictly cost/performance. It is assumed that the same functionality can be provided across all the models, and that the only reason to use an SDB machine is that it has a lower cost/performance ratio than the

other models. This is a simplifying assumption; in reality, decisions are made to use database machines based other criteria, such as the ability to connect the machines to heterogeneous systems, etc. Such functional requirements should be the basis of other studies.

The parameters to be measured are listed below.

2.2.1 Cost

Selling price is the cost associated with each system; this includes the selling price of software (data management and operating systems) as well as hardware. Selling price is used, rather than raw component cost, because the selling price truly reflects the cost to the target population. Additionally, selling price reflects the cost of development, which must be included, especially in the proposal of new systems.

2.2.2. Performance

The performance of a system is defined as the ratio of the number of users and the response time for an "average" job step:

P = n / r
P = performance
n = number of users
r = response time for average job step

The maximum performance for a given system will be used (i.e., the values of n and r such that the performance is maximum). For example, if an LCS has maximum performance at 200 users, response time at 10 seconds for an average job step, P = 20. Then, if the LCS costs $2,000,000 (hardware + software), the cost/performance ratio is 100,000. If the same "average job step" is run on a $300,000 (hardware + software) minicomputer, the maximum performance might be obtained at 20 users, with a response time of 20 seconds. Then the cost/performance ratio is 300,000/1 = 300,000.

2.2.3. Average Job Step

The average job step is defined by measuring current use of existing systems for the purpose of statistical data analysis and management. An average job step is a unit of work: a differentiable portion of a job which can be identified by the user as performing a given high-level task and perhaps even expressed in a single statement in an SDB query language. The method of determining the average job step is discussed in Section 3.

2.2.4. Loading Parameters

To estimate the performance of the nonexistent model (3) and to aid in guiding the development of the SDB machine, the loading parameters of the average job step will be measured for existing systems. These include:

d = number of disk references
cpu = amount of CPU time required

mem = amount of main memory required

3. Measuring the Parameters

The methodology for this study is as follows:

1) Determine average job step.
2) Measure it on current systems (models (1) and (2)) to determine their cost/performance ratios, and to discover system loading factors.
3) Use system loading factors to forecast the cost/performance ratio of (3).
4) Based on the above cost/performance ratios, conclude what the cost/performance ratio of model (4) must be.

The key problem is a statistically valid approach to determining the "average job step". This will be accomplished by measuring the current use of the two existing systems. A combination of approaches will be used: user surveys, and performance analyses of some of the current computer systems.

The test bed will be the Lawrence Berkeley laboratory. With 3000 workers in many areas of scientific research, LBL provides a diverse environment which appears to reflect the general use of computing for the analysis of data. Determining the cost/performance ratio for an SDB machine that would make it an alternative to current systems at LBL will fix the result for at least one laboratory. The results must be compared with others, of course, to identify possible anomalies.

To determine the average job step:

1) Randomly sample computer users at the laboratory. Ask them to fill out questionnaires concerning their use of the computing systems. A part of that questionnaire is a list of 20 candidate SDB "job steps". The user is asked which of his/her computer jobs contain which of the job steps, and which systems those jobs regularly run on. Some jobs will be found that are composed entirely of job steps.

2) Using the list of jobs identified in (1), measure the loading factors and response times of the job steps. Insure that there is a large enough set of steps at this point that standard statistical tests can be used to determine the average job step.

It can be expected that step 1 above is an iterative process in that the "candidate job steps", which are our ideas of the basic processes involved in the analysis and management of large data sets, will not necessarily correspond to what users regard as the basic job steps. Therefore, the questionnaire may have to be redesigned several times to capture the information needed.

Once the users have identified which jobs contain which job steps, the response times and system loading patterns of the job steps can be determined from examining logging facilities for most of the systems to be studied. Where those facilities do not exist, measurements will have to be taken.

The purpose of this study is to obtain the cost/performance ratio for the SDB machine, not to define a detailed performance profile for statistical data man-

agement queries. Therefore, extensive tracing facilities for finer determination of loading factors according to subprocesses within the job step will not be used for this study. It may be necessary to have such information when predicting the cost/performance ratio of a given SDB machine architecture, since, as found in [HAWT81, DEWI81] the performance profiles of applications greatly influence the utility of a given database machine architecture. Instrumentation and tracing of the SDB job steps may then be a logical extension of this study.

At the end of steps (1) and (2) above, there may be a single, "average job step" which has associated with it an average CPU time on each of two systems (models (1) and (2)), an average number of I/O requests, and an average response time from each of the two systems. This result would be very nice, since such a single step would be easy to model analytically, and is conceptually easy to deal with. However, such an "average" may mean nothing since the variation may be huge - different job steps may have radically different performance patterns. In that case, a suite of job steps will have to be devised, with each included job step indicative of a certain type of process used in the statistical analysis and management of data. This job-step suite can then be used in the place of a single average job step.

4. Conclusion

This paper has described work in progress on the design of a statistical database machine. A methodology for the exploration of the necessary cost/performance ratio for the database machine has been specified. This methodology is: (1) determine the target application for the database machine; (2) find a candidate user population for that machine; (3) devise tests to quantify the user population's current computing usage; and (4) forecast the necessary cost/performance ratio based on its current value.

Such a methodology does not account for a change in computer usage due to a change in functionality. That is, if a functionally complete statistical data management system existed, and if that system were very fast due to its implementation on a special-purpose database machine, the user population would probably change drastically due to the change in environment. In simply surveying present use of a computing system, it is difficult to determine if a given operation is not performed often because it is not needed, or because it is too hard, inconvenient, or slow on the current system.

Conversely, it is difficult to forecast the actual use of a new system, such as an SDB machine, if its use means extensive user re-training. The users of statistical databases may not wish to change from the systems they are familiar with because they may resist having to change systems.

Further research is required to determine how such changes in the user population should be reflected in the study of the design of an SDB machine.

REFERENCES

[BABB79] Babb, E., "Implementing a Relational Database by Means of Specialized Hardware", ACM Trans. on Database Systems, Vol. 4, No. 1, March 1979, pp. 1-29.

[BRIT80] Britton-Lee, Inc. "IDM 500 Intelligent Database Machine Product Description", Britton-Lee Inc., 90 Albright Way, Los Gatos, Calif., 95030.

[CHAN81] Chan, P., and A. Shoshani, "SUBJECT: A Directory Driven System for Organizing and Accessing Large Statistical Databases", Proc. VLDB, 1981.

[DEWI78] DeWitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Data Base Management Systems", Proc. Fifth Annual Symposium on Computer Architecture, 1978.

[DEWI81] DeWitt, D.J. and P. Hawthorn, "A Performance Evaluation of Database Machine Architectures", Proc. VLDB, 1981.

[EGGE81] Eggers, S., F. Olken and A. Shoshani, "A Compression Technique for Large Statistical Databases", Proc. VLDB, 1981.

[EPST80] Epstein, R. and P. Hawthorn, "Design Decisions for the Intelligent Database Machine", Proc. 1980 NCC, AFIPS Vol. 49, pp.237-241.

[HAWT79] Hawthorn, P. "Evaluation and Enhancement of the Performance of Relational Database Management Systems", Memo. no. M79/70, Electronics Research Laboratory, Universtiy of California at Berkeley.

[HAWT81] Hawthorn, P. "The Effect of Target Applications on the Design of Database Machines", Proc. SIGMOD, 1981.

[JOHN81] Johnson, R. "Modelling Summary Data", Proc. SIGMOD, 1981.

[STON81] Stonebraker, M. "Operating System Support for Database Management", Commun. ACM, July, 1981.

The NON-VON Database Machine: A Brief Overview[1]

David Elliot Shaw
Salvatore J. Stolfo
Hussein Ibrahim
Bruce Hillyer

Department of Computer Science
Columbia University

Gio Wiederhold
J. A. Andrews

Department of Computer Science
Stanford University

Abstract

The NON-VON machine (portions of which are presently under construction in the Department of Computer Science at Columbia, in cooperation with the Knowledge Base Management Systems Project at Stanford) was designed to apply computational parallelism on a rather massive scale to a large share of the information processing functions now performed by digital computers.

The NON-VON architecture comprises a tree-structured Primary Processing Subsystem (PPS), which we are implementing using custom nMOS VLSI chips, and a Secondary Processing Subsystem (SPS) incorporating modified, highly intelligent disk drives. NON-VON should permit particularly dramatic performance improvements in very large scale data manipulation tasks, including relational database operations and external sorting. This paper includes a brief overview of the NON-VON machine and a more detailed discussion of the structure and function of the PPS unit and its constituent processing subsystems.

---

[1]This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-80-G-0132.

1 Introduction

Recently, a great deal of commonality has become apparent among the fundamental operations involved in a surprisingly large number of superficially disparate computational approaches to high level database management. Although these operations have been formulated in different ways by different researchers, their essential characteristics are captured by the primitive operators of the relational algebra defined by Codd [1972]. Among these operators are the set theoretic operations union, intersection, and set difference, the relational operators equi-join and projection, and several other operations derivable from these five.

Although the best sequential algorithms known for these operations are still quite inefficient on a von Neumann machine, particularly in the case of very large databases, we believe it possible to implement alternative machine architectures supporting the highly efficient, but cost-effective, parallel execution of each of these relational algebraic operations, along with a number of other operations of practical importance, including large-scale external sorting. It is this belief which motivated the design of the NON-VON database machine.

NON-VON comprises a Secondary Processing Subsystem (SPS), based on a bank of "intelligent" rotating storage devices and designed to provide very high access and processing bandwidth, and a smaller, but faster Primary Processing Subsystem (PPS), again utilizing a high degree of parallelism, in which the relational algebraic operators may be very quickly evaluated. Transfer between the two devices is based on a process of hash partitioning, which is performed entirely in hardware by logic associated with the individual disk heads, and which divides the argument relations into key-disjoint buckets suitable for "internal" evaluation. The top-level organization of the NON-VON machine is illustrated in Figure 1.1.

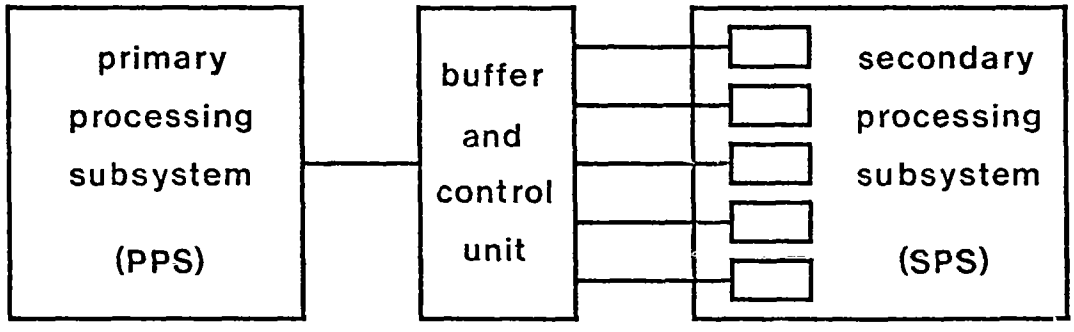


Figure 1.1 Organization of the NON-VON Machine

This paper examines the organization and behavior of the NON-VON machine, illustrating the essential mechanisms involved in its operation in database management applications. After tracing the theoretical origins of the NON-VON architecture in the next section, we focus in Section 3 on the central elements of the PPS unit. The most important functions of the PPS are described in Section 4, while the fifth section illustrates the way in which these functions are employed in the rapid parallel execution of the most "difficult" relational algebraic operations. Sections 6 and 7 outline the organization of the Secondary Processing Subsystem and illustrate its use in the external evaluation of relational database operations.

2 Theoretical Foundations

The theoretical basis for the NON-VON architecture was established in the course of a doctoral research project at Stanford [Shaw, 1979], and was accompanied by a mathematical analysis of the attainable time complexity of the equi-join and projection operators on such a machine. The architecture was shown to permit a rather surprising O(log n) increase in efficiency over the best evaluation methods known for a conventional computer system, without the use of redundant storage, and using currently available and potentially competitive technology. In many cases of practical import, the proposed architecture was also found to permit a significant improvement (by a factor roughly proportional to the capacity of the Primary Processing Subsystem) over the performance of previously implemented or proposed database machine architectures based on associative secondary storage devices.

Subsequently [Shaw, 1980a], algorithms for evaluating the selection, restriction, union, intersection and set difference operators (each with comparable or more favorable performance improvements) were also described, and the key procedure on which the architecture is based was contrasted with a related, but in this application, inferior method based on an associative sorting technique described earlier in the literature. More recently, we have been studying several highly efficient, linear expected time algorithms for external sorting on the NON-VON machine.

3 Organization of the Primary Processing Subsystem

The PPS unit functions as the site of what we call internal evaluation of the relational algebraic and other operations performed by NON-VON. Borrowing from the terminology of sorting, we use the term "internal" to distinguish that case in which the operands are small enough (or can be broken into small enough pieces) to fit entirely within the primary storage device -- in our case, the intelligent PPS unit; "external" evaluation refers to the case where the data exceeds the capacity of the PPS, and must be selectively partitioned and transferred from SPS to PPS.

For purposes of this discussion, the PPS may be thought of as composed of a large number of very simple processing elements (on the order of several thousand, if a full-scale prototype were to be built using 1981 technology, and between a hundred thousand and a million during that period during which NON-VON-like machines would in fact be targeted for practical use), interconnected to form a complete binary tree. With the exception of minor differences in the "leaf nodes", each processing element (PE) is laid out identically, and comprises:

1. A single common data bus

2. A very simple (and area-efficient) one-bit-wide ALU capable of manipulating a small set of local flag registers

3. An intelligent memory/comparator unit containing a small amount (perhaps 32 to 64 bytes) of local random-access storage, and capable of arithmetic comparisons (including equality) between values taken from the bus and from specified memory locations

The top-level structure of a single PE is illustrated in Figure 3.1.

By contrast with a conventional microprocessor, no finite-state control logic is incorporated within the constituent PE's. Instead, a single programmable logic array (PLA) associated with each chip services all PE's on that chip, as described below.

The PPS will be implemented largely using two custom-designed VLSI chips, which we call the PPS Bottom Chip and PPS Middle Chip. Bottom Chips will each contain a subtree of the full PPS tree, and will thus embody 2^k-1 constituent PE's for some k depending on device dimensions. Rough preliminary estimates based on 2.5 micron design rules suggest that a value of k = 3, corresponding to 7 PE's per Bottom Chip, might be feasible for our initial prototype. Within a single Bottom Chip, the PE's will be configured geometrically according to a "hyper-H" embedding of the binary tree [Browning, 1978], as illustrated in Figure 3.2.

Because of its fixed I/O bandwidth requirements, independent of the size of the embedded subtree, the realizable capacity of the PPS Bottom Chip will increase quadratically with inverse changes in minimum feature width, thus permitting dramatic increases in the computational power of the NON-VON PPS unit as device dimensions are scaled downward with continuing advances in VLSI technology. (During the target time frame for a production version of a NON-VON-like machine, a k value of 7 or 8, corresponding to several hundred processing elements per PPS Bottom Chip, seems feasible.)

The PPS Middle Chip, on the other hand, will embed 2^m-1 "internal nodes" of the PPS tree (where m is a constant determined by pinout limitations, and independent of device dimensions), serving to combine 2^{m-1} subtrees, embedded either in separate Bottom Chips or (recursively) in lower-level subtrees

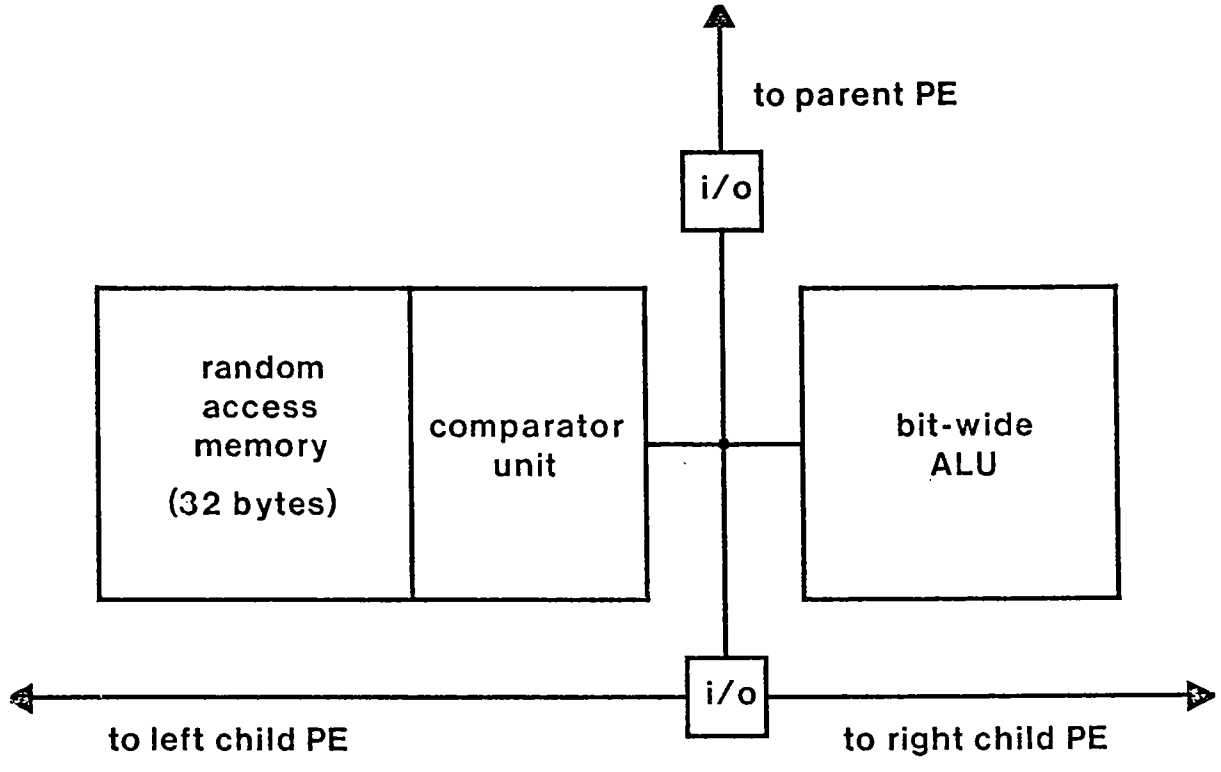


Figure 3.1 Components of a Single Processing Element

rooted in other Middle Chips, into a single complete binary subtree. Because the number of processors per Middle Chip will be constrained by pinout limitations, and not by by minimum feature width, the capacity of the PPS Middle Chips will not benefit from the effects of scaling as will the Bottom Chips. This (provably unavoidable) I/O bandwidth limitation, however, will result in only a small, constant waste factor; the tree-structured intra- and inter-chip interconnection topology of the NON-VON Primary Processing Subsystem is in fact extremely well suited to the effects of future downward scaling.

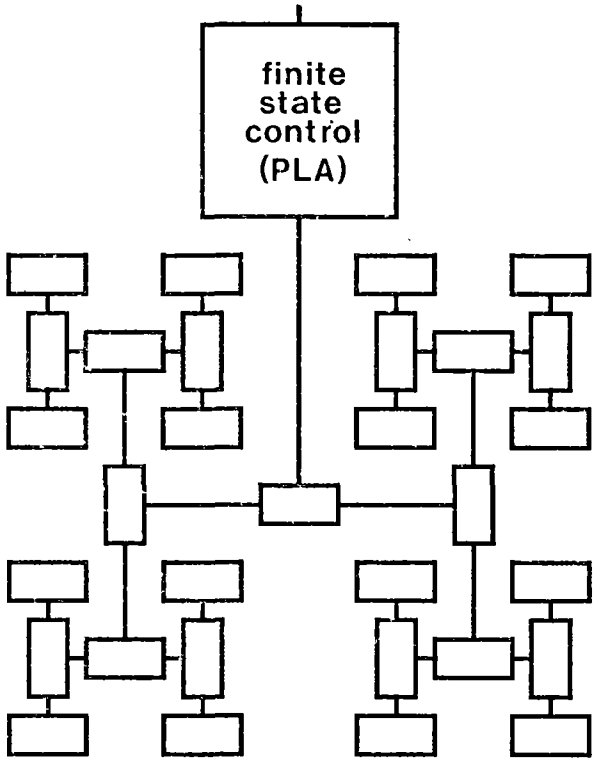


Figure 3.2 Structure of the PPS Bottom Chip

4 Function of the Primary Processing Subsystem

The NON-VON PPS functions as a SIMD (Single Instruction stream, Multiple Data stream) processing ensemble, with all PE's executing the same operation on different data at any given point in time. The tree structured inter-PE bus structure functions in three distinct modes in the course of such execution:

1. Global broadcast by any PE to all other PE's in the PPS

2. Physically adjacent neighbor communication (to the Parent, Left Child and Right Child PE within the physical PPS tree

3. Linearly adjacent neighbor communication (to the Next or Previous PE in an arbitrary linear logical sequence)

The global broadcast function supports the rapid (especially as VLSI device dimensions scale downward) parallel communication of instructions and data to the individual PE's as required for SIMD operation, and is employed for most I/O operations. In some algorithms (real-time parallel sorting, for example), data is passed between parent and child PE's. In others (associative matching of arbitrary-length tuples, for instance), data and control information is exchanged with the immediate predecessor or successor PE in some predefined total ordering. Several mappings between the linear logical sequence and the hierarchical physical topology of the PPS are possible, but are beyond the scope of this paper.

The NON-VON PPS instruction set supports a number of operations involving associative retrieval, arithmetic comparison, logical manipulation of local (to the individual PEs) flags, various kinds of I/O, and a number of incidental functions. While space does not permit a discussion of all these functions, two associative operations executed by the PPS hardware are of sufficient importance in the implementation of database management applications to merit special attention here.

In performing associative operations, the NON-VON PPS unit functions as a relatively fast, but inexpensive content-addressable memory which performs what is essentially the relational selection operation in a short, fixed amount of time, independent of the size of the argument relation. In both of the associative operations under consideration, a partial match criterion--that is, a set of attribute/value pairs which must be satisfied by all "matching" tuples--is broadcast in parallel to all PE's. NON-VON is then capable of either

1. Associative marking: Simultaneously setting a flag bit in all PE's associated (in a manner to be explicated shortly) with a matching tuple, or

2. Associative enumeration: Reading successive matching tuples out of the PPS unit (and into the control module) irredundantly, with each new tuple produced in a small, fixed amount of time.

For simplicity, we may assume (at least in the context of this paper) that a given PE will store at most one tuple. The converse, however, is not the case: a single tuple, or even a single attribute value, could well exceed the capacity of one PE, there being no restriction on the length of either. Such tuples are stored in linearly adjacent PE's. Inter-PE propagation of the matching activity is effected by passing flags in parallel from each "successful-so-far" PE to its linear successor.

At the end of an associative marking operation, every PE that contains a matching tuple (or, in the case of large tuples, every PE in which a matching tuple starts) will have one of its internal one-bit flags set to 1. The corresponding register in all other PE's will be set to 0. In some cases, associative marking may be followed by another parallel operation involving this "mark register"; in other applications, however, it may be necessary to output all marked tuples (or the relevant portions thereof) through the root of the tree in an arbitrary sequential order, using the global communication bus. This associative enumeration operation is supported by a simple and elegant multiple match resolution scheme which uses the tree-structured communication path to very rapidly clear the mark register in all but an arbitrary "first" marked PE.

It is expected that the NON-VON PPS architecture will perform associative matching operations extremely rapidly--in fact, at a pace limited largely by the speed at which the partial match specification itself can be input. Through the exploitation of recent architectural advances applicable to VLSI systems, however, along with the careful balancing of storage capacity against distributed intelligence, we hope to bring the cost of PPS storage to within a small constant multiple of the price of an equivalent amount of ordinary random access memory implemented using comparable technology.

5 Internal Evaluation of the Relational Algebraic Operators

As noted above, the PPS unit's associative marking and enumeration operations may themselves be regarded as implementations of the relational selection operator, which returns a relation consisting of all tuples satisfying a particular attribute-value specification. Selection, though, can be performed entirely within NON-VON's Secondary Processing Subsystem, obviating the need for transfer to, and processing within, the PPS. The importance of the associative marking and enumeration operations instead derives from its use as a building block in the implementation of the "difficult" operations (project, equi-join, and the set theoretic operations, for example) which, in contrast with relational selection, can not be evaluated by the SPS alone. While a detailed exposition of the algorithms for each of these "difficult" operations is beyond the scope of this paper, the essential behavior of the NON-VON PPS and SPS units may be illustrated by considering a single, particularly demanding operation which has a particularly simple realization within the NON-VON PPS: the equi-join.

The join operation may in general be extremely expensive on a conventional von Neumann machine, since the tuples of the two relations must be compared for equality of the join attributes before the extended cartesian product of each group of matching tuples can be formed. In the absence of physical clustering with respect to the join attributes (whose identity may vary in different joins involving the same relations), joining is most commonly accomplished on a von Neumann machine by pre-sorting the two argument relations with respect to the join fields. The order of the tuples following the sort is actually

gratuitous information from the viewpoint of the join operation. From a strictly formal perspective, the requirements of a join--that the tuples be paired in such a way that the values of the join attribute match--are significantly weaker than those of a sort. The distinction is moot in the case of a von Neumann machine, where no better general solution to this pairing problem than sorting is presently known. On the NON-VON machine, on the other hand, we are able to exploit the weaker requirements of the join operation to eliminate the need for pre-sorting in favor of a more straightforward associative approach.

The algorithm for the internal equi-join on the NON-VON machine is in fact extremely simple; it may be regarded as an associative version of the naive join algorithm in which each tuple in the source relation in turn is compared with all tuples of the target relation. On a von Neumann machine, this naive algorithm has quadratic (in the size of the argument relations) time complexity--specifically, the number of sequentially executed steps is equal to the product of the combined cardinalities of the source and target relations. Within the NON-VON PPS, however, only the source tuples are processed sequentially, since a given source tuple may be compared with all target tuples in parallel using the associative marking operation introduced earlier.

In applications where the result relation is to be retained in the PPS (as, for example, in the case where the join is to be followed immediately by another parallel operation on the result relation), the number of steps is thus equal to the cardinality of the source relation, and is independent of the cardinality of the target relation. Where it is necessary to sequentially output the result relation (either to the SPS, or to some system external to the NON-VON machine), a number of additional steps proportional to the size of the result relation is required; the complexity of our algorithm, however, remains linear in the cardinalities of the source and result relations.

It should be noted that, in the worst case (in which the join attributes of both argument relations have only a single value, so that the join produces the full extended cartesian product of the two relations as its result), the result relation will itself contain a number of tuples equal to the product of the cardinalities of the two argument relations. As must be the case for any algorithm involving fixed-bandwidth sequential enumeration of the result relation, the NON-VON internal join thus has a worst case which is quadratic in the size of the argument relations considered alone.

Our analysis of time complexity in terms of the size of the argument and result relations is motivated by empirical observations involving the relationship between argument and result cardinalities in realistic relational database applications. In practice, the degenerate case of a single-valued join appears to appear so infrequently by comparison with joins in which the argument and result relations are of roughly comparable size that the argument/result model seems to better reflect performance differences of practical significance. Under the assumptions of this model, the NON-VON internal join algorithm offers a very significant advantage over the best algorithms known for a join on a von Neumann machine.

6 Organization of the Secondary Processing Subsystem

The SPS unit is based on a potentially large bank of highly intelligent circulating mass storage devices. Either single- or multiple-head disk drives are suitable as a basis for the NON-VON PPS unit, but it is assumed (in the interest of economy) that the number of tracks considerably exceeds the number of read/write heads--that is, that "ordinary", as opposed to "head-per-track", drives are employed. Each head must have its own sense amplification electronics, permitting all heads to simultaneously read their respective tracks.

A small amount of hardware is associated with each disk head. The following capabilities are assumed for this "per-head" logic. First, it must be possible to examine an arbitrary attribute in each tuple that passes under the associated head, comparing each such value against a single specified pattern value. As in the case of many earlier database machine designs (RAP [Ozkarahan, Schuster and Smith, 1974], CASSM [Su, Copeland and Lipovski, 1975] and DBC [Baum and Hsiao, 1976], for example), the SPS is able to collect for output all tuples found to match, or to bear some specified arithmetic relationship (less than, less than or equal to, etc.) to the given pattern value.

In addition, though, the NON-VON SPS is capable of sequentially computing a hash function for which the resulting hashed value falls within the range [0, 1]. (Sequential computation of an exclusive or function is sufficient, and requires little additional logic.) Tuples whose selected values hash to within specified subranges of the [0, 1] interval may be dynamically identified and transferred to the PPS unit through the Buffer and Control Unit.

7 External Evaluation of the Relational Algebraic Operators

In the case where the arguments to the join exceed the capacity of the PPS device, NON-VON attempts to partition the argument relations into a set of key-disjoint buckets, the vast majority of which are small enough to fit entirely within the PPS. A set of buckets is called key-disjoint if no join value is represented in more than one bucket. In general, one such bucket will then be transferred into the PPS during each successive revolution of the disk-based SPS, and an internal join performed on the subrelations in question.

To accomplish the partitioning, the [0, 1] range of the hash function is divided into a number of equal subintervals somewhat larger than the combined size of the argument relations in PPS-fulls. Unless the argument relations consume an unusually large share of the total amount of storage available within the system, it is possible to associatively examine, and perform simple processing on, all tuples in both argument relations within a single disk

revolution. During the first revolution of the SPS drives, all disk heads hash each join value passing beneath them, and those tuples whose join values hash into the first subinterval are transferred "on the fly" into the PPS for internal evaluation. Tuples hashing into the second interval are transferred during the second revolution, and so on. Even after recovery from (statistically unlikely) "bucket overflows", the hash partitioning procedure preserves the linearity of the internal join algorithm in the case of large argument relations.

References

Baum, Richard E. and Hsiao, David K., "Data Base Computers--a Step Towards Data Utilities", IEEE Transactions on Computers, v. C-25, December, 1976.

Browning, Sally, "Hierarchically Organized Machines", in Mead, Carver and Conway, Lynn, Introduction to VLSI Systems, Addison-Wesley, 1978.

Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus", Proceedings of the 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, Association for Computing Machinery, 1971.

Codd, E. F., "Relational Completeness of Data Base Sublanguages", in Rustin, Randall (ed.), Courant Computer Science Symposium 6: Data Base Systems, Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1972.

Gallaire, Herve, Minker, Jack, and Nicolas, J. M., "An Overview and Introduction to Logic and Data Bases", in Gallaire, Herve and Minker, Jack, Logic and Data Bases, New York, Plenum Press, 1978.

Kaplan, S. Jerrold, Cooperative Responses from a Portable Natural Language Database Query System, Ph.D. Thesis, Department of Computer and Information Science, University of Pennsylvania, May, 1979.

Knuth, Donald E., The Art of Computer Programming, vol. 1: Fundamental Algorithms, Addison-Wesley, 1969.

Lee, C. Y. "Intercommunicating Cells as a Basis for a Distributed Logic Computer", Proceedings of the AFIPS 1962 Fall Joint Computer Conference, Baltimore, Spartan Books, pp. 130-136, 1962.

Ozkarahan, Esen A., Schuster, Stewart A., and Sevcik, K. C., "A Data Base Processor", Technical Report CSRG-43, Computer Systems Research Group, University of Toronto, September, 1974.

Shaw, David Elliot, "A Hierarchical Associative Architecture for the Parallel Evaluation of Relational Algebraic Database Primitives", Stanford Computer

Science Department Report STAN-CS-79-778, October, 1979.

Shaw, David Elliot, "A Relational Database Machine Architecture", Proceedings of the 1980 Workshop on Computer Architecture for Non-Numeric Processing, Asilomar, California, March, 1980. (Also reprinted in joint issue of ACM SIGARCH, SIGIR and SIGMOD publications.)

Shaw, David Elliot, Knowledge-Based Retrieval on a Relational Database Machine, Ph.D. Thesis, Department of Computer Science, Stanford University, 1980a.

Su, Stanley Y. U., Copeland, George P., and Lipovski, G. J., "Retrieval Operations and Data Representations in a Content-Addressed Disc System", Proceedings of the International Conference on Very Large Data Bases, Framingham, Massachusetts, September, 1975.

Wiederhold, Gio, Kaplan, S. Jerrold, and Sagalowicz, Daniel, "The Knowledge Base Management Systems Project", ACM SIGMOD Record, 1981.

THE SYSTEM ARCHITECTURE OF A DATABASE MACHINE (DBM)

S. Bing Yao, Fu Tong, and You-Zhao Sheng

Database Systems Research Laboratory
University of Maryland
College Park, Maryland 20742

1. Introduction

Recent advances in computer hardware technology have made it possible to design special processors for dedicated functions. The internal organization of a data base system contains many concurrent operations which may be implemented by dedicated hardware that functions concurrently. This will have the advantage of reduced system complexity and increased performance through parallel processing.

Database machines using conventional technology have been proposed [DEW79,EAH80]. In this paper we will briefly describe the architecture for a new approach to database machine DBM. Figure 1 illustrates the use of DBM in a local network environment. The DBM nodes are the back-end machines that manage the data. The workstation nodes are the front-end processors that interact with the user. Data base access requests can be expressed by user in languages such as SEQUEL or QUEL [AST76,SWK76]. These queries are parsed by the workstations and the resulting query packets are sent to the appropriate DBM nodes. The DBM commands contained in the query packets cause the DBM to send instructions to its special processors to initiate query processing. When the processing is completed, the query result is sent by the DBM to the appropriate workstations in result packets.

We assume that the data base consists of a collection of relations and the query received by the DBM is encoded in relational calculus. In an earlier paper [YAO 79], we have shown that relational queries can be decomposed into a series of simple data base operations including selection, projection and join. These operations will be implemented in hardware in the proposed DBM architecture.

Special processors to perform selection and projection have been proposed in other database machine. For example, in CAFS a micro-programmed processor evaluates the data stream from the secondary storage for satisfying a given query. The result is coded in a one-dimensional array [MAL79]. In [PAD 79] a logic matrix structure is proposed to sequentially evaluate Boolean functions. The "systolic" array [KUN 80] processes the selection operation using a pipelining algorithm. Other proposals include [HAN77,LSZ78]. In our DBM system, selection operations are processed by a "data filter" which extends the concepts of these previous designs. We will briefly discuss the architecture of our "data filter" in Section 3.

Several designs for join processors were also proposed. The "systolic" array join processor [KUN 80] is basically a one-dimensional processor array. The two relations to be joined are piped into the array from two opposite directions. The two pipelines move in synchronous, one step for each time unit. Join processing takes place when tuples from different relations meet in a join processor element. The join processor array in DBC [MEN 81] is arranged in a circular fashion. One of the relations to be joined is partitioned and loaded into the memories of the join processors. The other relation to be joined is then piped into the join processors. Except for the use of associative memories and hash functions, this approach is basically the same as the systolic array. In the bit-sliced associative join processor of DIALOG [WAY 80], the join values from one of the relations to be joined are stored in a bit-sliced associative memory. The tuples from the other relation are processed serially by matching their join values in bit-slice. All of these three designs are, in fact, variations of one-dimensional processor arrays. In this paper we will introduce a two-dimensional join processor array used in DBM.

It should be pointed out that, contrary to the belief of some researchers in this field (see e.g., [DEW 81]), the DBM components using array processors can be readily implemented with existing VLSI technology. A small-scale prototype of DBM is presently being implemented. Experiments and performance of the system will be reported in forthcoming papers.

2. DBM System Architecture

The general architecture of the system is similar to a conventional computer with the exception of a few additional processors. Figure 2 shows that the system is composed of memory (M), central processor (CP), direct memory access control (DMA), data filter (DF), and join processor (JP). The disk controller (DC) is interfaced to the system by the data filter and direct memory access controller. Communication with the network is performed by the communication controller (CC).

The query sent to the DBM is assumed to be represented in a query tree structure and coded in a standard notation. For example, the following query

```
RETRIEVE Name
FROM Employee
WHERE (salary > avg (salary)) and (dept = CS)
```

will be represented in the tree structure as shown in Figure 3. Multiple query trees may be entered into the system. The evaluation and coordination of the queries are controlled by a system monitor. The main functions of the system monitor include: 1) Concurrency control; 2) backup recovery; 3) receiving query packets from the communication controller; 4) interpret the query tree and decompose the query into one variable queries; 5) initiate the operations for the data filter and join processor; 6) assemble query result in system buffer area; and 7) initiate the process of sending result packets.

The decomposition of the queries uses an algorithm similar to that found in System R [AST76] and INGRES [SWK76]. The transaction management is based on an implementation of a data base operating system [GRA78]. Similar to many relational data base systems, all the meta data for the purpose of system control and directory are also stored as relations in the system. In summary, the system architecture resembles a conventional computer system. The unique aspect of the system is the use of a data base operating system and the implementation of special processors for performing selection and joining. In the following sections, we will further describe the design of these special processors.

3. The Design of a Data Filter

The function of the data filter is to perform selection and projection operations. The data filter always performs the processes on a single relation. An instruction to the data filter always specifies a one-variable query in disjunctive (or conjunctive) normal form. The symbolic reference to relation names and attribute names are mapped to physical addresses by the DBM using an internal schema. The internal schemas are stored in special relations maintained by the data filter. Indices are used to quickly reduce the scope of search. Given a query, the existence and utilization of indices are determined by the data filter. Once the physical location for data access is determined, the data filtering process begins. The data stream retrieved from the disk is examined by the data filter for satisfying one query. If a match is found, attributes in the tuple are then projected to form the query result.

The central component of the data filter is a m*n processor array as shown in Figure 4. Each row of the array is connected by an AND/OR network. The result of all the rows are then evaluated by another OR/AND network. Each processor unit performs a simple comparison between an input attribute and a query value. It is easy to see that this processor array evaluates a disjunctive (conjunctive) normal form query depending upon the selection of the AND/OR operations.

The processor array has two input data streams: one comes from the query and the other comes from the stored relation. Before the comparison can take place, the input data streams are loaded into each processor rows in parallel and within each row of processors the data are propagated in a pipeline fashion. If we assign each row of the processor array to a conjunct (disjunct) then the input data can be loaded to all the rows in parallel. Similarly, if we reserve one processor for each term within a conjunct (disjunct) then the query values could be loaded to the comparators in parallel. In the case when there is not sufficient processors to handle a query, multi-pass operation that serially loads and compares the query values with the data stream must be performed. That is, there are two ways to load the query values [Figure 5]: a) serial query loading. Only one single processor is required. For each input data value, the processor compares each query value in turn. The results are accumulated in the AND/OR network, and b) parallel query value loading. In order to load all the query values into the processors simultaneously, a large number of processors may be required to handle arbitrarily complex queries.

The scheduling of the processors in a data filter must insure that the loading of query values is synchronous with the flow of data streams. The constraint is that all the query values must be loaded and compared before the next data value is loaded from the data stream. In [STY81] this constraint was found to be $t_0 \geq t_a + t_b$, if the selection is to be processed by a single processor. Where t_0 is the minimum time required for input a byte from the data stream, t_a is the time required for loading a byte of query value, and t_b is the time required for obtaining the comparison results of the processor. Assuming that it takes two clock periods to load a query value and one clock period to read the query result, the synchronization constraint becomes $t_0 \geq 3 * t_{cp}$. This condition imposes a constraint on the lower bound of the DBM system clock frequency.

As an example, assume that the disk has a transfer rate of 2 megabyte per second ($t_0 = 0.5$ μs). This implies that the system clock cycle must be less than 0.16 μs, which corresponds to a frequency of 6.25 MHz.

4. The Join Processor Array

The join operation is the most complex relational operation, because two relations to be joined must be accessed simultaneously and iteratively. If the number of tuples in the two relations are m and n, respectively, then the complexity of this operation is proportional to m * n. If we divide the two relations into x and y subfiles, respectively, and process all of the subfiles in parallel, as shown in Figure 6, then the complexity of the join operation can be reduced to $\frac{m*n}{x*y}$.

In order to increase the processing speed of the join processor, data are broadcasted to all the processors simultaneously. The join processor array could be considered as an independent peripheral device which could communicate with the system through a data bus. When a join operation is initiated, the DBM controller transfers the initial parameters to the join processor array, sends the selected values of both relations into the buffer memories in the join processor array, and then issues a start signal to activate it. The addresses of the matched tuples will be returned to the system.

We must emphasize that before putting the join processor unit into action, the two relations to be joined are taken from the result of selection/projection and stored in buffers. Only the values from the specified domain (not the entire tuples) are stored in the internal buffer memories of the join processor. These values are masked out and stored into the internal buffers concurrently with the selection/projection process. In this sense, the join processor is not really an independent unit; there must exist some common control logic which controls both the selection/projection unit and the join processor unit.

The architecture of the join processor array is shown in Figure 7. The detail of its design is given in [TAY81A,TAY81B]. There are four components of the system: the internal buffer storages for partitioned key values from the relations R and S; the processors which compares the key values read from the buffer storages; the output buffer storages

which store the addresses (or pointers) of the matched records; and the control/timing logic which issues the micro-operation control signals. Each of the four components, except the control/timing logic, is in turn composed of a set of identical cells. The highly regular structure makes it a candidate for VLSI implementation.

Assume that the relations R and S are partitioned into x and y subrelations respectively, the selected values from the tuples of these two relations are pre-stored in the buffer storages $R_1,..,R_x$ and $S_1,..,S_y$ respectively. To process the join operation, a set of x R-values are broadcasted to x rows of processors. At the same time, a set of y S-values are broadcasted to y columns of processors, in which they are compared with x R-values simultaneously. The comparison results are stored as a bit-matrix for the controller to generated the partial join results. The next comparison operation will not take place until the contents of the bit-matrix are consumed. After all S-values in S_j's are broadcasted and compared, we then start to process the next set of R-values in R_i's in the same way. This continues until all R-values in R_i's are processed.

5. VLSI Implementation Considerations

One of the most important problems in hardware implementation of relational database operations is the high development cost caused by the hardware complexity. Recent advances in VLSI technology have removed many of the hardware design and implementation difficulties [PAT80]. Using computer aided design tools, an enormous amount of logic circuits can be integrated on a single silicon chip.

There are three major constraints that must be considered for VLSI implementation:

1) the equivalent number of gates or transistors required must fit the present state-of-the-art of VLSI technology;

2) the chip area occupied by the internal bussing and connections must be confined to a reasonable percentage of the total chip area;

3) the input/output terminals, including both the data and the control signals, must not exceed the maximum allowable pins on the package.

The regular structure of the processor arrays and the limited input-output data lines makes them suitable for VLSI implementation. The complexity of the processor can be evaluated by counting the number of transistors required. The total number of transistors for a single data filter processor is estimated to be approximately 13.2 K. In addition, 12.3 K transistors are required for the output buffer storage. The estimation of the total transistors required for the data filter depends upon the number of processors used in the processor array. If we assume a 4 x 2 process array and a 10% overhead for control logic, then the estimated number of transistors required by the data filter is 130 K [STY 81]. Suppose the data bus width is 8 bits, the address bus width is 16 bits, and there is only one data input line from the disk to

the data filter. A 40-pin package should satisfy all the input-output requirements.

The number of equivalent transistors contained in each functional logic block of a two-dimensional join process array is evaluated and listed in Table 1.

Table 1

Functional block	Number of equivalent transistors
comparator cell	400
key-value memory block	2216
out-put memory block	4958
controller	3654
auxiliary logic	1306

Suppose we have a 16 x 16 processor array. The maximum number of key values that could be processed in parallel is 4096 bytes. Each memory block of the double output buffer has a capacity of 17 * 8 = 136 16-bit words. We could evaluate the approximate number of transistors required for each functional block as listed in Table 2. The total number of transistors required is approximately 260K.

Table 2

Functional block	Number of basic cells	Estimated number of transistors required
Processor matrix	256	102400
Internal key-value buffers	32	70912
Output buffer	16	79328
Control logic	1	8614
Total		261254

An alternative way to implement the two-dimensional join processor array, is to integrate each row of the array into a single chip. The design has a fewer number of transistors per chip, but is more flexible in structuring a join processor array. Figure 8 shows a block diagram of such organization. In this example, only one R-value storage module and one output buffer module are needed. The number of processor cells is also greatly reduced.

6. Summary

We have briefly described the architecture for DBM. The system has an architecture similar to conventional computers. Its unique feature is the addition of a few special processors to perform critical data base operations. The system architecture is also based on system components of software database systems. A small-scale prototype system is presently being developed. We have adopted many algorithms developed in other relational data base systems. The first implementation will also include a small-scale processor array for selection and joining. We plan to further study the performance of the prototype system. Investigation of the implementation of the special processors in VLSI devices will also be undertaken.

REFERENCES

[AST76]Astrahan,M.M.,et. al,(14 authors), "System R: A Relational Approach to Data Base Management,", ACM Trans. on Database Systems, Vol. 1, No.2, June 1976.

[BAB79]Babb,E., "Implementing a Relational Database by means of Specialized Hardware", ACM TODS, Vol. 4, No. 1, Mar. 79, pp 1-29.

[BAN78]Banerjee,J. and Hsiao,D.K., "Concepts and Capabilities of a Database Computer", ACM Trans. on Database Sys., Vol.3, No.4, Dec, 1978.

[CAB74]Chamberlain,D.D. and R.F.Boyce, "SEQUEL: A Structured English Query Language", Proceedings, 1974 ACM SIGFIDET Workshop, Ann Arbor, Michigan, May 1974.

[DEW79]DeWitt,D.J., "DIRECT - A Multiprocessor Organization for supporting Relational Data Bases Management Systems", IEEE Trans. on Computers, Vol. C-28, No. 6, June 1979.

[DEW81]DeWitt,D.J. and Madison,WI., "A Performance Evaluation of Database Machine Architectures", 7-th International Conference on VLDB, Cannes, France, Sept. 9-11,1981.

[EAH80]Epstein,R. and Hawthorn,P., "Design Decisions for the Intelligent Database Machine", NCC, 1980.

[EPS80]Epstein,R. and Hawthorn,P., "Aid in the '80s", Datamation, 1980.

[GRA78]Gray,J., "Notes on Data Base Operating Systems", IBM Research Report RJ2198, Feb. 1978, San Jose, California 95193.

[KUN80]Kung,H.T., and Lehman,P.L., "Systolic (VLSI) Arrays for Relational Database Operations", ACM SIGMOD, 1980.

[LSZ78]Leilish,H.O., Stiege,G., Zeidler,H.Ch., "A Search Processor for Database Management Systems", IEEE, 1978.

[MAL79]V.A.J.Maller, "The Content Addressable File Store - CAFS", ICL Tech J. Nov. 1979, 265-279.

[MEN81]M.J. Menon, and David K. Hsiao, "Design and Analysis of a Relational Join Operation for VLSI," Report, Dept. of Computer and Information Science, The Ohio State University, February, 1981.

[PAD79]Piavsic,V.M. and Danielsson,P.E., "Sequential Evaluation of Boolean Functions", IEEE Trans. on Computers, Vol. C-28, No. 12, Dec. 1979.

[PAT80]Patterson, David A., and Sequin, Carlo H., "Design Considerations for Single-Chip Computers of the future", IEEE Trans. on Computers, Vol. C-29, NO. 2, Feb. 1980.

[STY81]Sheng,Y.Z., Tong,F., Yao,S.B., "Data Filter -- A Relational Selection Processor", Tech. Report, Database Research Laboratory, University of Maryland, College Park, MD 20742, October 1981.

[SWK76]M. Stonebaker, E. Wang, and P. Kreps, "The Design and Implementation of INGRES", ACM Trans. on Database Sys., Vol. 1, No. 3, September 1976.

[TAY81A]Tong,F. and Yao,S.B., "Design of a Two-Dimensional Join Processor Array", 6-th Workshop on Computer Architecture for Non-Numerical Processing, Hyeres, France, June 1981.

[TAY81B]Tong, F., and Yao, S.B., "Logical Organization of Two-Dimensional Join Processor Matrix", Technical report, Database Research Laboratory, Univ. of Maryland, College Park, MD 20742, 1981.

[WAY80]Wah,B.W. and Yao,S.B., "DIALOG---A Distributed Processor Organization for Database Machines", AFIPS Press, Vol. 49, 1980.

[YAO79]Yao,S.B. "Optimization of Query Evaluation Algorithms", ACM TODS, 4, 2 (June 1979).

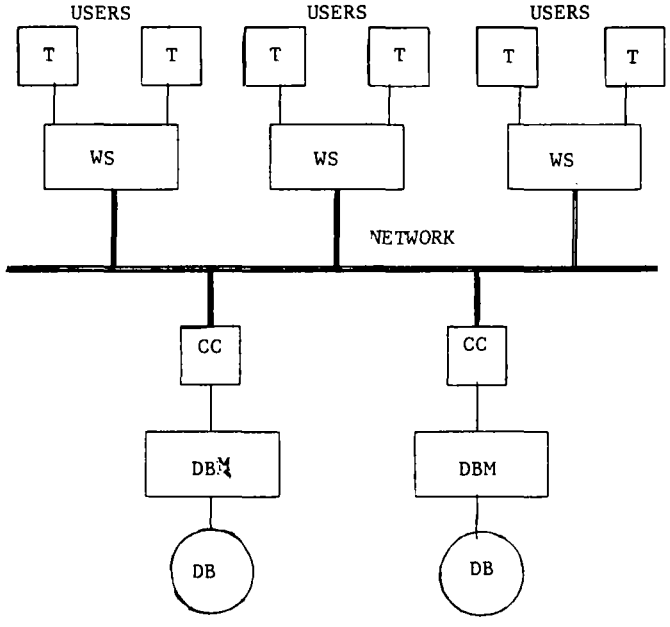


Figure 1 DBM as a Node in a Local Network

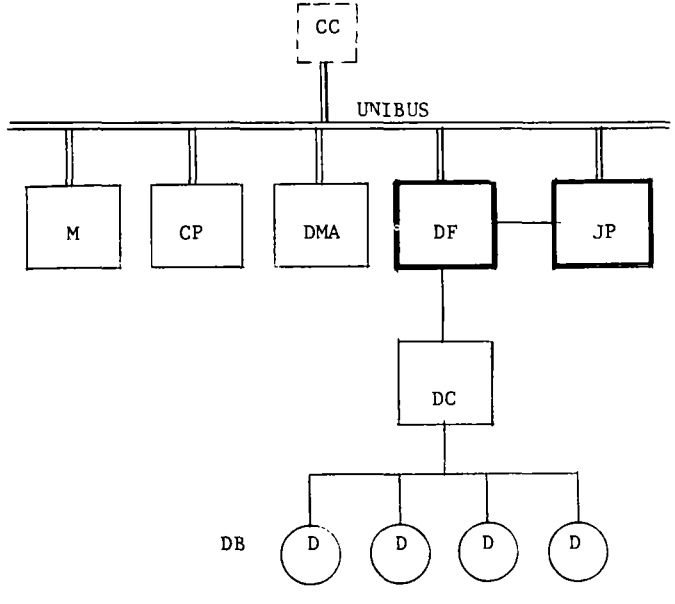


Figure 2 The System Architecture of DBM

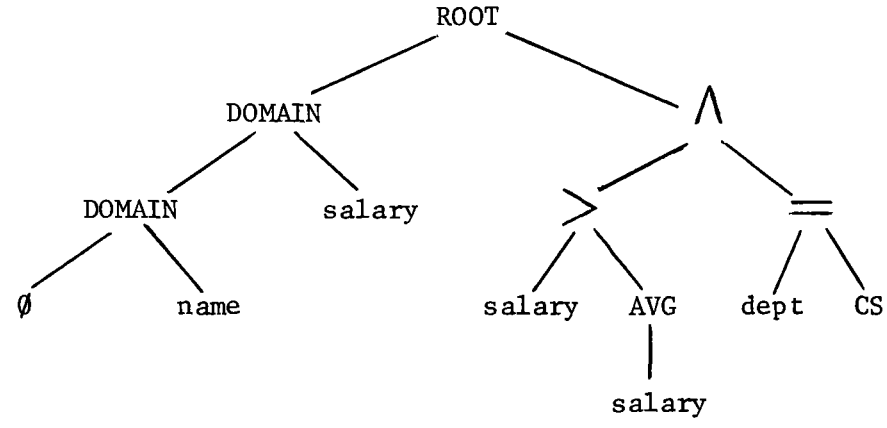


Figure 3 Tree Representation of Query

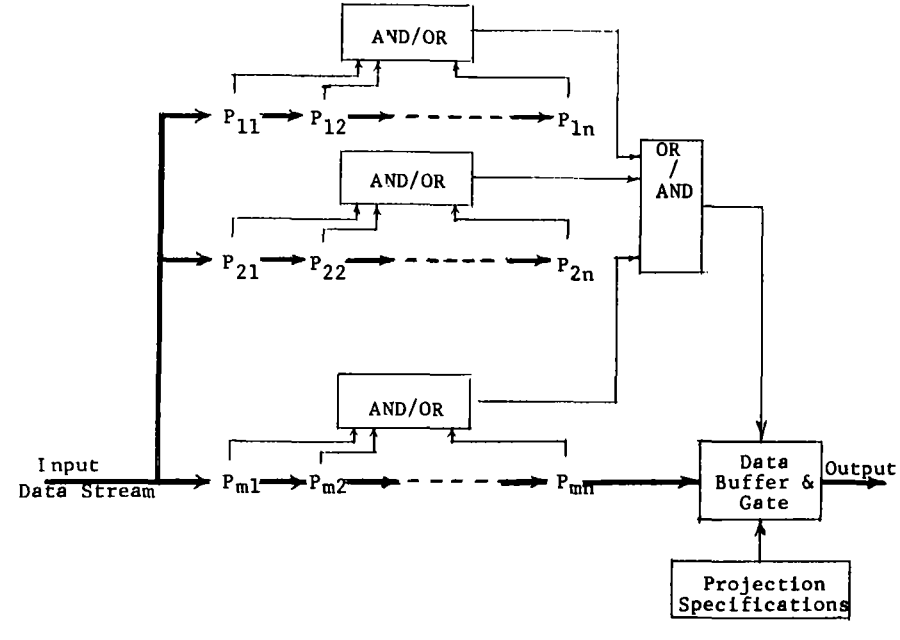


Figure 4 The Selection Processor Array

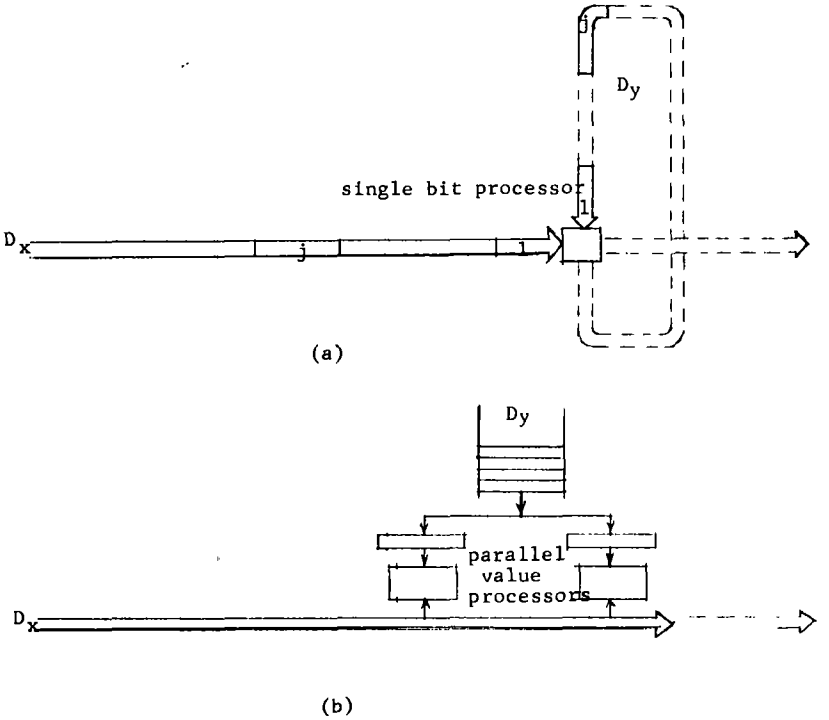


Figure 5 Serial and Parallel Query Value Loading

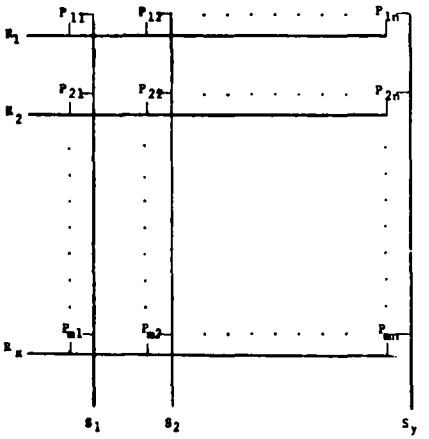


Figure 6 The Join Processor Array

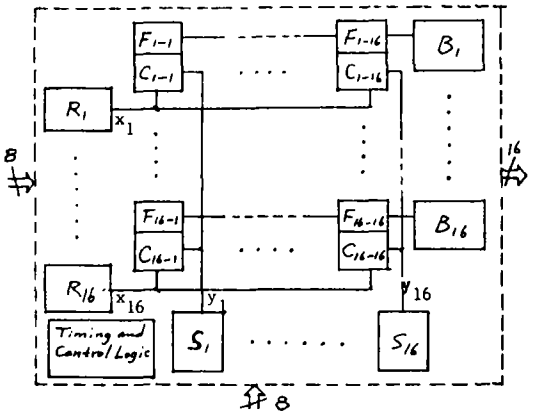


Figure 7 System Organization of the Join Processor Array

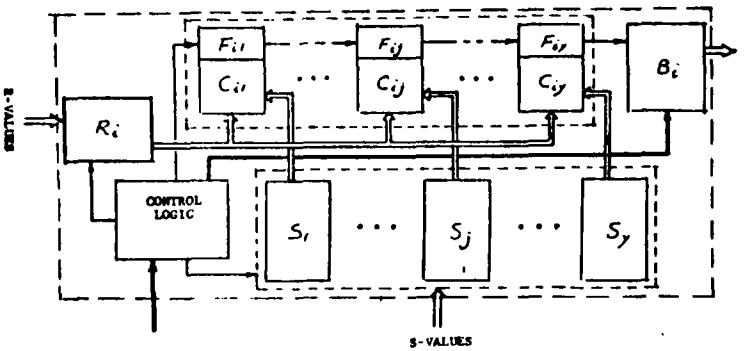


Figure 8 One Row of the Join Processor

Well-Connected Relation Computer

Sudhir K. Arora*
Surya R. Dumpala**

Abstract

Well Connected Relation Computer (WCRC) is a data base machine architecture which is intended to support different data models simultaneously on the same physical data. The computer stores data as binary partitions (PCP's), supports a conceptual level in the Entity-Relationship model and provides the user the flexibility of operating in the relational, the network or the hierarchical model. This is one approach to implementing the ANSI SPARC proposals for a data base system.

I Introduction

Several data base machine architectures have been studied in the literature - CASSM [COPE 73], RAP [OZKA 75], DIRECT [DEWI 78], RARES [LIN 76], DBC [BANE 79], RELACS [OLIV 79], SEARCH PROCESSOR [LEIL 78], XDMS [CANA 74], IFAM [DEFI 73] etc.. Some of these machines address only specific operations on a data base; some implement only one of the three major data models - Relational, Network or Hierarchical; some implement all three data models but not simultaneously on the same physical data.

There is a need for a data base machine architecture which can support different data models simultaneously on the same physical data. In such a machine users should be able to view the data according to the relational, network or hierarchical data model. This is called "logical data independence". Further, changes to the physical data should have minimal effect on the user's view of the system - "physical data independence". Such an architecture would conform to the ANSI/X3/SPARC [ANSI 75] proposals or the coexistence model [NIJS 76] which envisage three levels for a data base system - external, conceptual and internal. At the external level the system should support different data models depending on user needs. At the conceptual level a stable common view of data and its semantics must reside. The physical data is stored at the internal level and may be altered to take advantage of evolving technologies.

To the best of our knowledge there are only two projects addressing this problem -- GDBMS and WCRC [DOGA 80, AROR 81]. In this paper we present an overview of WCRC.

Note: This research was supported by a Science and Engineering Research Board grant number 214-7248.

* Department of Computer Science, Wayne State University, Detroit, Michigan.
** Department of Electrical and Computer Engineering, McMaster University, Hamilton L8S 4L7, Canada

II Background

The theory of Well Connected Relations (WCR's) has been presented in [AROR 79]. For this paper, it is sufficient for the reader to know only a few definitions given below.

A well connected relation (WCR) is a binary relation W on two sets A and B such that

$$(\forall a)\ (a \in A)\ (\forall b)\ (b \in B)\ (aWb)$$

The sets A and B are called the first and the second constituents of the WCR.

An elementary well connected relation (EWCR) is a WCR in which the first constituent has a single element. The second constituent is then called the image set of the first constituent.

A relation R[A,B] can be expressed as

$$R[A,B] = \sum_{i=1}^{n} R\,[A_i,B_i]$$

$$= R_1[A_1,B_1] \cup R_2[A_2,B_2] \cup \ldots \cup R_n[A_n,B_n]$$

$$= \pi(R) = \text{a partition of } R$$

where $R_i[A_i,B_i] \cap R_j[A_j,B_j] = \phi$

for $i \neq j$, $1 \leq i, j \leq n$ and $A = \bigcup_{i=1}^{n} A_i$ and $B = \bigcup_{i=1}^{n} B_i$

A partition, W_i, of a binary relation R[A,B] is a canonical partition if

$$R[A,B] = \sum_{i=1}^{n} W_i[A_i;B_i]$$

where, i) $W_i[A_i;B_i]$ is a WCR for $1 \leq i \leq n$,

ii) A_i is a set with a single element for $1 \leq i \leq n$, and

iii) $A_i \neq A_j$ for $i \neq j$ and $1 \leq i, j \leq n$.

A partition of a binary relation R[A,B] is a pseudo canonical partition (PCP) if

$$R[A,B] = \sum_{i=1}^{n} W_i[A_i;B_i]$$

where, 1) $W_i[A_i;B_i]$ is a WCR for $1 \leq i \leq n$, and

ii) A_i is a set with a single element for $1 \leq i \leq n$.

Later in this paper we use PCP's for storing data in WCRC. In [AROR 80] we have proposed a language based on WCR's for data base systems. The language is data model independent and can apply equally well to the network, relational and hierarchical data models. The need for data model independent languages has become apparent in recent years. They can be used for the conceptual level of an ANSI SPARC architecture, for communication in a distributed data base system, for data base restructuring and so on. Data model independent languages reported in the literature include FQL [BUNE 79], QUEST [HOUS 79], LSL [TSIC 76] and WCRL [AROR 80].

III Basic Organization

WCRC (Well Connected Relational Computer) is a data base computer intended for non-numeric processing. As shown in Fig. 1, the system consists of three major components corresponding to the three levels: the External Processor, the Conceptual Processor and the Internal Processor.

The External Processor performs the following main functions:

i) Queueing of jobs (or queries)
ii) Priority encoding of queries
iii) Translating queries from a user language into a conceptual level language (WCRL)
iv) Security checking to protect the data base from unauthorized operations.

It has three memory areas: The User Work Space (UWS), the Interface Buffer Area (IBA) and The Processor Memory Area (PMA). A portion of UWS is allotted to each user as his work space. In this area, the user can define his own view, a part of the data base as seen by him and specify the constraints on it such as granting other users to use this view or some additional integrity constraints.

The IBA is organized as a first-in first-out (FIFO) memory where the queries are stored after they are translated from user languages into the conceptual level language. The translation is handled by three translators, corresponding to the three data models, which are stored as software modules in the processor memory. The jobs may also be ranked on a preassigned job priority encoding scheme.

The external level may be implemented on a host computer by software. This would involve developing program packages in the host language such as COBOL or PASCAL

to handle the translation as well as other housekeeping activities.

The conceptual level supports the conceptual model and the user subschemas in the three major data models. The conceptual level model is based on the Entity-Relationship model [CHEN 76]. The language at the conceptual schema is the WCRL [AROR 80]. The conceptual level has the facilities for query optimization and translation from WCRL into the machine language (WCRML) and a data dictionary. In addition, it also provides facilities for Data Base Administrator (DBA) to act directly on the conceptual model of the data base. A special high level language (DBAL) is provided to handle the DBA requirements.

The conceptual level consists of eight functional blocks (Fig. 1): The Controller, The Query Analyser (QA), The Query Translator (QT), Three User Schema Descriptors (USD's), Conceptual Schema Descriptor (CSD) and the Buffer Memory (BM). The Conceptual Processor maintains the information about the storage structure at the internal level as well as about the conceptual model. The Query Analyser breaks up the queries into subqueries on the storage structure at the internal level. The CSD contains the information about the schema definition operations on the schema, the constraints and the data dictionary. It describes the logical units of data in the data base and specifies the integrity constraints and security measures such as access restrictions to certain units of data. The description of a view by an application is called a subschema. It is a logical subset of a schema. The USD's at the conceptual level contain the subschema definition, operations and constraints corresponding to views in three models. The information in CSD and USD's is used by the QA during query analysis. The QT performs the task of translating analysed queries in WCRL into machine level primitives (WCRML) that can be directly executed by the Internal Processor and stores them in the Buffer Memory.

The internal level consists of a number of Query Processors (QP's) and an array of Cell Processors (CP's). Each QP is a master processor which is responsible for executing one query in WCRML using the cells. The cells are logic-per-track devices. The data is stored on the cell tracks as binary partitions, known as pseudo-canonical partitions (PCP's). The PCP's are partitioned binary relations. All the cells are independent of each other, i.e., there is no direct communication among them. They can only communicate through the query processor which controls them. The data on the cells can be read by any number of QP's but only one QP can write onto the track at a time. The QP selects the cells required by the query it is handling and makes them slaves. They are released once the query is processed. The QP co-ordinates its slaves and also computes the overall set results. Several queries can be handled at the same time, since the QP's can work in parallel. This is a multiple-instruction/multiple-data stream organization (MIMD).

IV Query Processing

In this section, an overview of how a query is processed at three levels is discussed. These various stages of query processing are illustrated in Fig. 2. The queries originate at the user end in three languages corresponding to three different models. These are translated into WCRL, priority encoded and put in a queue in the IBA by External Processor. The system also supports the DBA who works on the conceptual level directly but reaches it through the external level like any other user. Therefore, some queries may be originated by the DBA.

Once a query is received by the External Processor, it is checked for syntax and violation of any security or integrity constraints. After it successfully goes through these stages, it is translated and put on the job queue. Then the Conceptual Processor is activated. It takes one query at a time from the IBA and performs query analysis on it. It breaks up the query on logical well-connected relations (WCR's) into subqueries on the stored PCP's. It checks whether logical WCR's can be constructed from the stored ones and also subjects them to the security and intergrity constraints. Any violation of these constraints would terminate the query at this stage and an error message would be channeled to the user through the external level. Successful queries would now be translated by the Query Translator into WCRML and submitted to the internal level through the Buffer Memory. The query translator also provides the information about the required cells along with the machine primitives. This minimizes the search time required to select the cells. The QP's take the queries from the Buffer Memory and select the required tracks by a polling scheme. The machine language primitives are executed on the slave processors. For the retrieval queries, the data is sent to the external level directly through an I/O read mechanism. For the update queries, the success/failure is communicated to the user.

V The Strorage Structures

The data is stored on the cell tracks as PCP's. Since in a canonical partition the WCR's may be of arbitrarily large size (number of tuples), it cannot be implemented directly as a storage structure. Instead, two types of physical storage structures are proposed in this section for the storage of data on cell tracks: PCP-Option I, and PCP-Option II.

PCP - Option I

A partition of a binary relation R[A,B,] is a pseudo canonical partition - Option I, if

$$R[A,B] = \sum_{i=1}^{n} W_i[A_i;B_i]$$

where

i) $W_i[A_i;B_i]$ is a WCR for $1 \leq i \leq n$,

ii) A_i is a set with a single element for $1 \leq i \leq n$, and

iii) A_i or B_i values are ordered.

It may be noted that, elements of A_i may be repeated in several WCR's, unlike the CP's. In such a PCP all WCR's can be made equal in size by breaking the larger WCR's into several smaller ones. The size of each of these PCP's can be standardized at the system level or cell level. With the WCR size fixed, each of them can be assigned a "fixed" position on the track. This reduces movement of data and eliminates the need for garbage collection. The track format for this data structure is shown in Fig. 3. The track is divided into two halves. Each half

contains one constituent of a PCP. Logically, any one of the two constituents of a PCP may be made the first constituent. This gives rise to two possible PCP's for the same binary data, one forward PCP and one reverse PCP. A header is provided at the beginning of each half for this purpose. The header contains the number of WCR's in that half, the attribute name(s) of that constituent and some flags, common to all the WCR's in that half. Each standard sized WCR consists of a set of mark bits, an address field containing a value of the first constituent and a fixed number of pointers. The pointers contain addresses of the values in the second constituent of the WCR in the other half of the track. Thus, all pointers in one half of the track point to addresses in the other half of the same track and excessive pointer chasing is avoided. An example of a binary relation stored as a PCP on a track is shown in Fig. 4.

This storage structure has the following advantages. The pointers in each half of a track are always forward pointing. Therefore, there is no backward referencing and the hardware implementation would be simpler. The data is organized in such a way that queries based on both the forward and the reverse PCP's can be answered with equal ease. It also avoids duplication of data values. However, if a PCP overflows one track, it may give rise to duplication of some values on these cell boundaries. If the WCR size on a track is chosen in such a way that the "average" WCR size in the PCP is an integral multiple of this, the extra storage for the duplicate values and the pointers may be minimized. Also, if we use a counter to locate the WCR's (standard sized) on the track, there is no need to store the WCR address with each WCR.

PCP - Option II

Though use of pointers in the PCP - option I provides an efficient utilization of memory space, the pointer chasing would increase either the complexity of hardware or computation time to perform operations like join which require data contiguity. In view of this, another storage structure is presented here, which involves no pointers.

This second type of storage sturcture exploits an inherent characteristic of the E-R model. In all the CP's corresponding to entities and relationships, the first constituent is always a key or a system-defined key. Therefore, a CP would always be a set of EWCR's, where each non key value may be connected to several key values, but not the other way around. Such a CP may be viewed as a set of disjoint EWCR's. It is formally defined as follows. It is different from a PCP - Option I in that the elements of the constituents are not ordered.

A partition of a binary relation R[A,B] is a pseudo canonical partition - Option II if

$$R[A,B] = \sum_{i=1}^{n} W_i[A_i;B_i]$$

where

i) $W_i[A_i;B_i]$ is a WCR for $1 \leq i \leq n$,

ii) A_i is a set with a single element for $1 \leq i \leq n$

iii) The elements of A and B are unordered.

In option II, the EWCR's may be of arbitrarily large size by breaking the larger EWCR's into several smaller ones. Now, the hardware size of each EWCR can be standardized in a manner similar to the PCP's of Option I. The track format of this storage structure is shown in Fig. 5. The track is continuous and undivided, unlike the Option I format. The header of the track contains the information about the track address, size of the EWCR's in the track and some flags common to the whole track. It also contains the attribute names of the first and second constituents. Each EWCR consists of several mark bits, a first constituent value field and certain number of second constituent value fields. The number of second constituent value fields (key values) per EWCR is standardized at the system level or track level. This makes the size of the EWCR fixed and it is specified in the track header. A limit is placed on this size to facilitate hardware implementation. Thus the EWCR size may be variable within this hardware limit. An example of a binary canonical partition stored as PCP's of Option II on a track is shown in Fig. 6.

This storage structure offers the following advantages. There is no duplication of value fields as long as the EWCR size is not greater than the hardware limit. There are no pointers and hence storage requirements for pointers and pointer chasing are avoided. Also it is possible to view the PCP's as both forward and reverse PCP's for the same stored binary data because the second constituent values are key values and are never duplicated.

VI Concluding Remarks

In this paper we have described the Well Connected Relation Computer (WCRC) [AROR 81]. WCRC implements the ANSI/X3/SPARC proposals or the coexistence model of data. The architecture of WCRC is based on a simple data structure, WCR which has been studied in detail elsewhere [AROR 79]. The architecture has 3 levels - external, conceptual and internal. It can simultaneously support the 3 major data models - network, relational and hierarchical at the external level. At the conceptual level a stable view of the data base can be implemented in the Entity Relationship model. A data model independent language based on WCR's (WCRL [AROR 80]) is used at the conceptual level. At the internal level the physical data is stored in logic-per-track pseudo associative memory. The approach taken by other researchers in this area has been to store related data contiguously as n-ary relations (for example). We depart from this approach radically. We store data as partitions of binary relations - Pseudo Canonical Partitions (PCP's). Two possible storage structures for PCP's - Option I and Option II - have been described.

REFERENCES

[ANSI 75]
ANSI/X3/SPARC study Group on Data Base Management Systems, Interim Report, FDT, ACM-SIGMOD, Vol. 7, No. 2, 1975.

[AROR 79]
Arora, S.K., Smith, K.C., "A Theory of Well-Connected Relations", J. of Information Sciences, 19, 1979, pp. 97-134.

[AROR 80]
Arora, S.K., Smith, K.C., "WRCL: A Data Model Independent Language for Data Base Systems", To appera, Int. J. of Comp. and Inf. Sciences, 1980.

[AROR 81]
Arora, S.K., Dumpala, S.R., Smith, K.C., "WCRC: An ANSI SPARC Machine Architecture for Data Base Management", To appear, Proc. Eighth International Symp. on Comp. Arch., Minneapolis, May 1981.

[BANE 79]
Banerjee, J., Hsiao, D.K., Kannan, K., "DBC - A Database Computer for Very Large Databases", IEEE Trans. on Computers, Vol. C-28, No. 6, June 1979, pp. 414-429.

[BUNE 79]
Buneman, P., Frankel, R.E., "FQL - A Functional Query Language", ACM-SIGMOD, Boston, 1979, pp. 52-58.

[CANA 74[
Canaday, R.H. et al., "A Back-End Computer for Data Base Management", CACM 17, 10, Oct. 1974, pp. 575-582.

[CHEN 76]
Chen, P.P.S., "Entity Relationship Model - Towards a Unified View of Data:, ACM Trans. on Database Systems, March, 1976, Vol. 1, No. 1, pp. 9-36.

[COPE 73]
Copeland, G.P., Lipovski, G.J., Su, S.Y.W., "The Architecture of CASSM: A Cellular System for Non-numeric Processing", First Annual Symposium on Computer Architecture, 1973.

[DEFI 73]
DeFiore, C.R., Berra, P.B., "A Data Management System Utilizing an Associative Memory", Proc. ACM National Computer Conf., 1973, pp. 181-185.

[DEWI 78]
DeWitt, D.F., "DIRECT - A Multiprocessor Organization for Supporting Relational Data Base Management Systems", Proc. Fifth Annual Symp. on Comp. Architecture, 1978, pp. 182-189.

[DOGA 80]
Dogac, A., Ozkarahan, E.A., "A Generalized DBMS Implementation on a Database Machine", ACM SIGMOD, Santa Monica, California, May 1980, pp. 133-143.

[HOUS 79]
Housel, B.C., "QUEST: A High Level Data Manipulation Language for Network, Hierarchical and Relational Databases", IBM Res. Rep., RJ2588(33488) 7/25/79, 1979.

[LEIL 78]
Leilich, H.O., Stiege, G., Zeidler, H. Ch., "A Search Processor for Data Base Management Systems", Proc. Fourth VLDB, West Berlin, Sept. 1978, pp. 280-287.

[LIN 76]
Lin, C.S., Smith D.C.P., Smith, J.M.. "The Design of a Rotating Associative Memory for Relational Data Base Applications", ACM TODS, Vol. 1, Mar. 76, pp. 53-65.

[NIJS 76]
Nijssen, G.M., "A Gross Architecture for the Next Generation Database Management Systems", Modelling in Data Base Management Systems, North-Holland, 1976, pp. 1-24.

[OLIV 79]
Oliver, E.J., "RELACS, An Associative Computer Architecture to Support a Relational Data Model", Ph.D. Thesis, Syracuse University, 1979.

[OZKA 75]
Ozkarahan, E.A., Schuster, S.A., Smith, K.C., "RAP - An Associative Processor for Data Base Management", National Computer Conf., 1975, pp. 379-387.

[TSIC 76]
Tsichritzis, D., "LSL: A Link and Selector Language", ACM-SIGMOD, Washington, D.C., June 1976, pp. 123-134.

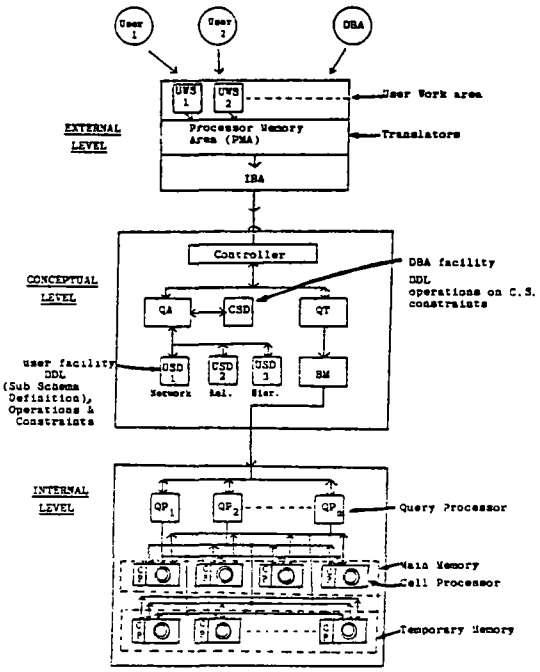


FIG. 1 OVERALL ARCHITECTURE OF WCPC

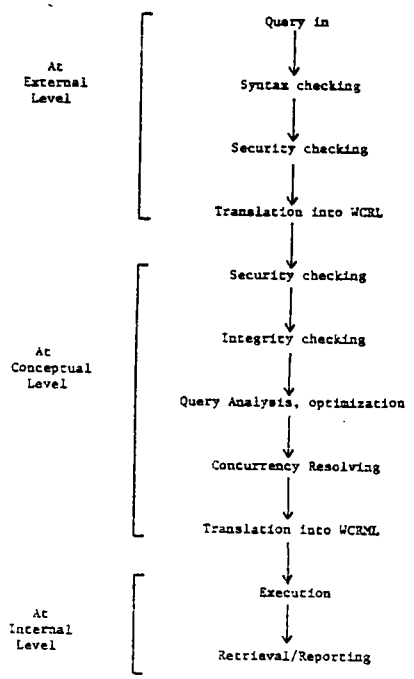


Fig. 2 The Stages of Query Processing

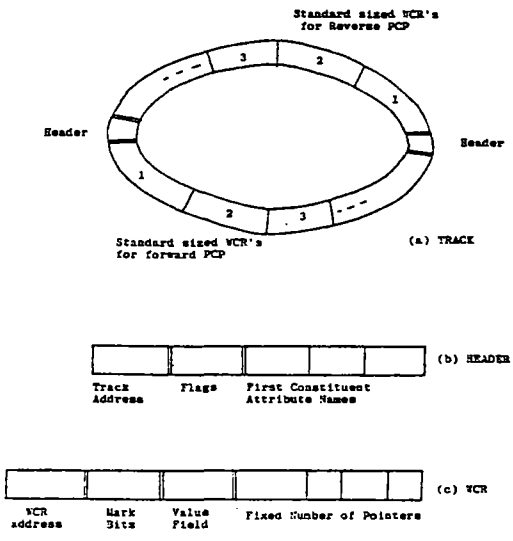


FIG. 3 TRACK FORMAT

a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_2
a_2	b_1
a_3	b_1
a_3	b_2

(a) Binary Relation

(1) (, , ,) (a_1) (1, 2, 3)

(2) (, , ,) (a_2) (1, 2, ϕ)

(3) (, , ,) (a_3) (1, 2, ϕ)

(b) One half of track

(1) (, , ,) (b_1) (1, 2, 3)

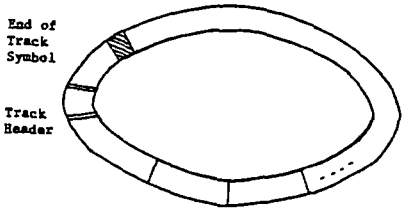
(2) (, , ,) (b_2) (1, 2, 3)

(3) (, , ,) (b_3) (1, ϕ, ϕ)

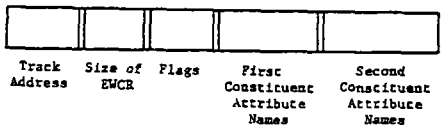
(c) Other half of track

FIG. 4 PCP's on a Track - Option I

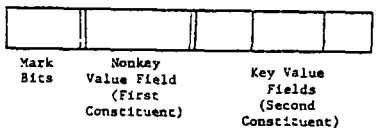
Fig. 5 Track Format of PCP - Option II



(a) Track



(b) Header



(c) EWCR

Non Key Field	Key Field
a_1	b_1
a_1	b_2
a_2	b_6
a_2	b_7
a_2	b_8
a_3	b_3
a_3	b_4

(a) A Binary Relation

1) (, , ,) (a_1) (b_1, b_2, ϕ)

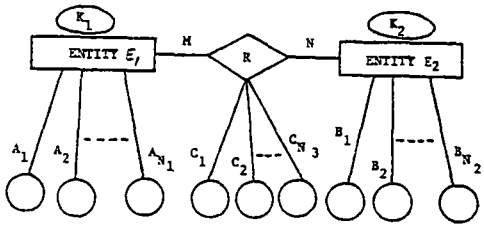
2) (, , ,) (a_2) (b_6, b_7, b_8)

3) (, , ,) (a_3) (b_3, b_4, ϕ)

Mark bits

(b) The PCP's on Track

Fig. 6 PCP's on a Track - Option II



(a) A Typical Segment in the Data Base

$[K_1; A_2]$ $[K_2; B_2]$ $[a; K_1]$ $[a; C_1]$

$[K_1; A_2]$ $[K_2; B_2]$ $[a; K_2]$ $[a; C_2]$

. . . .

$[K_1; A_{N_1}]$ $[K_2; B_{N_2}]$ $[a; C_{N_3}]$

(b) The PCP's

Fig. 7 An Example Showing PCP's Corresponding to an E-R Schema

Recent & Upcoming Selections from the

OVER 300 titles

IEEE
COMPUTER SOCIETY PRESS

1109 SPRING STREET, suite 201 / P.O. BOX 639 / SILVER SPRING, MARYLAND 20901 (301)589-3386

Alphabetical by Title/Subject Areas — October 1981

ORDER NO	PRICES NM/M	AVAIL	TITLES	YEAR PUBL	CODE	PAGES	IEEE NO
301	10.00/ 7.50	OK	PROC., APPLICATION OF PERSONAL COMPUTING TO AID THE HANDICAPPED WKSHP.	1980	P	88	80CH1596-6
393		11/81	PROC., ARTS; SYMPOSIUM ON USING SMALL COMPUTERS IN THE	1981	P		81CH1721-0
165	25.00/18.75	OK	PROC., CIRCUITS, SYSTEMS, & COMPUTERS (11TH ASILOMAR CONF.)	1977	P	492	77CH1315-1
197	25.00/18.75	OK	PROC., CIRCUITS, SYSTEMS, & COMPUTERS (12TH ASILOMAR CONF.)	1978	P	764	78CH1369-8
227	25.00/18.75	OK	PROC., CIRCUITS, SYSTEMS, & COMPUTERS (13TH ASILOMAR CONF.)	1979	P	615	79CH1468-8
328	30.00/22.50	OK	PROC., CIRCUITS, SYSTEMS, & COMPUTERS (14TH ASILOMAR CONF.)	1980	P	520	80CH1625-3
383		02/82	PROC., CIRCUITS, SYSTEMS, & COMPUTERS (15TH ASILOMAR CONF.)	1981	P		
087	20.00/15.00	OK	PROC., COMPCON '73 (SPRING)	1973	P	246	73CH0716-1
077	20.00/15.00	OK	PROC., COMPCON '74 (FALL)	1974	P	272	74CH0869-8
080	20.00/15.00	OK	PROC., COMPCON '74 (SPRING)	1974	P	310	74CH0825-0
071	20.00/15.00	OK	PROC., COMPCON '75 (FALL)	1975	P	341	75CH0988-6
067	20.00/15.00	OK	PROC., COMPCON '75 (SPRING)	1975	P	280	75CH0920-8
070	20.00/15.00	OK	PROC., COMPCON '76 (FALL)	1976	P	350	76CH1115-5
069	20.00/15.00	M/F O	PROC., COMPCON '76 (SPRING)	1976	P	231	76CH1069-4
129	20.00/15.00	OK	PROC., COMPCON '77 (FALL)	1977	P	474	77CH1258-3
133	20.00/15.00	OK	PROC., COMPCON '77 (SPRING)	1977	P	372	77CH1165-0
169	25.00/18.75	M/F O	PROC., COMPCON '78 (FALL)	1978	P	433	78CH1388-8
172	20.00/15.00	OK	PROC., COMPCON '78 (SPRING)	1978	P	384	78CH1328-4
254	25.00/18.75	OK	PROC., COMPCON '79 (FALL)	1979	P	492	79CH1465-1
217	25.00/18.75	OK	PROC., COMPCON '79 (SPRING)	1979	P	329	79CH1393-8
303	30.00/22.50	OK	PROC., COMPCON '80 (FALL)	1980	P	750	80CH1598-2
287	25.00/18.75	OK	PROC., COMPCON '80 (SPRING)	1980	P	515	80CH1491-0
372	25.00/18.75	OK	PROC., COMPCON '81 (FALL)	1981	P	358	81CH1702-0
341	30.00/22.50	OK	PROC., COMPCON '81 (SPRING)	1981	P	482	81CH1626-1
123	25.00/18.75	OK	PROC., COMPSAC '77 (1ST CONF. ON SOFTWARE APPLICATIONS)	1977	P	834	77CH1291-4
196	30.00/22.50	OK	PROC., COMPSAC '78 (2ND CONF. ON SOFTWARE APPLICATIONS)	1978	P	832	78CH1338-3
236	36.00/27.00	OK	PROC., COMPSAC '79 (3RD CONF. ON SOFTWARE APPLICATIONS)	1979	P	912	79CH1515-6
315	36.00/27.00	OK	PROC., COMPSAC '80 (4TH CONF. ON SOFTWARE APPLICATIONS)	1980	P	900	80CH1607-1
379	36.00/27.00	12/81	PROC., COMPSAC '81 (5TH CONF. ON SOFTWARE APPLICATIONS)	1981	P	446	81CH1698-0
128	25.00/18.75	OK	PROC., COMPUTER APPLICATIONS IN MEDICAL CARE (1ST CONF.)	1977	P	373	77CH1270-8
198	25.00/18.75	OK	PROC., COMPUTER APPLICATIONS IN MEDICAL CARE (2ND CONF.)	1978	P	667	78CH1413-4
223	36.00/27.00	OK	PROC., COMPUTER APPLICATIONS IN MEDICAL CARE (3RD CONF.)	1979	P	900	79CH1480-3
326	68.00/51.00	M/F O	PROC., COMPUTER APPLICATIONS IN MEDICAL CARE (4TH CONF.)	1980	P	2080	80CH1570-1
377	60.00/45.00	11/81	PROC., COMPUTER APPLICATIONS IN MEDICAL CARE (5TH CONF.)	1981	P	1164	81CH1696-4
271	28.00/21.00	OK	PROC., COMPUTER APPLICATIONS IN RADIOLOGY (6TH CONF.)	1979	P	432	79CH1404-3
096	20.00/15.00	OK	PROC., COMPUTER ARCHITECTURE (2ND CONF.)	1975	P	231	75CH0916-7
099	20.00/15.00	OK	PROC., COMPUTER ARCHITECTURE (3RD CONF.)	1976	P	202	76CH1043-5
146	20.00/15.00	OK	PROC., COMPUTER ARCHITECTURE (4TH CONF.)	1977	P	210	77CH1182-5
174	20.00/15.00	OK	PROC., COMPUTER ARCHITECTURE (5TH CONF.)	1978	P	256	78CH1284-9
237	22.00/16.50	OK	PROC., COMPUTER ARCHITECTURE (6TH SYMPOSIUM)	1979	P	243	79CH1394-6
291	25.00/18.75	OK	PROC., COMPUTER ARCHITECTURE (7TH SYMPOSIUM)	1980	P	315	80CH1494-4
346	28.00/21.00	OK	PROC., COMPUTER ARCHITECTURE (8TH SYMP.)	1981	P	518	81CH1593-3
218	10.50/ 8.50	OK	PROC., COMPUTER ARCHITECTURE FOR NON-NUMERIC PROCESSING (4TH CONF.)	1978	P	133	
378	25.00/18.75	11/81	PROC., COMPUTER ARCHITECTURE FOR PATTERN ANALYSIS WORKSHOP	1981	P	348	81CH1697-2
017	20.00/15.00	OK	PROC., COMPUTER ARITHMETIC (3RD SYMP.)	1975	P	256	75CH1017-3
176	22.00/16.50	OK	PROC., COMPUTER ARITHMETIC (4TH SYMP.)	1978	P	274	78CH1412-6
347	22.00/16.50	OK	PROC., COMPUTER ARITHMETIC (5TH SYMP.)	1981	P	278	81CH1630-3
364	30.00/22.50	OK	PROC., COMPUTER GRAPHICS ASSN; INAUGURAL CONFERENCE	1980	P	324	
387	36.00/25.00	OK	PROC., COMPUTER GRAPHICS ASSOCIATION (2ND CONF.)	1981	P		
005	16.00/12.00	M/F O	PROC., COMPUTER HARDWARE DESCRIPTION LANGUAGES	1975	P	191	75CH1010-8
274	20.00/15.00	OK	PROC., COMPUTER HARDWARE DESCRIPTION LANGUAGES (4TH SYMP.)	1979	P	200	79CH1436-5
038	12.00/ 9.00	OK	PROC., COMPUTER NETWORKING SYMPOSIUM (1977 CONF.)	1977	P	130	77CH1252-6
216	14.00/10.50	OK	PROC., COMPUTER NETWORKING SYMPOSIUM (1978 CONF.)	1978	P	137	78CH1400-1
278	16.00/12.00	OK	PROC., COMPUTER NETWORKING SYMPOSIUM (1979 CONF.)	1979	P	180	79CH1467-0
330	16.00/12.00	OK	PROC., COMPUTER NETWORKING SYMPOSIUM (1980 CONF.)	1980	P	192	80CH1586-7
380		12/81	PROC., COMPUTER NETWORKING SYMPOSIUM (1981 CONF.)	1981	P		81CH1699-8
137	16.00/12.00	OK	PROC., COMPUTER SCIENCE & ENGINEERING CURRICULA WORKSHOP	1977	P	149	EHO126-3
145	32.00/25.60	OK	PROC., COMPUTER SOFTWARE ENGINEERING: RELIABILITY, MGT. & DESIGN SYMP.	1976	P	583	76CH1071-0
113	12.00/ 9.00	M/F O	PROC., COMPUTER SOFTWARE RELIABILITY SYMPOSIUM	1973	P	167	73CH0741-9
082	12.00/ 9.00	OK	PROC., COMPUTER-AIDED DIAGNOSIS OF MEDICAL IMAGES (1976 SYMP.)	1976	P	100	76CH1170-0
155	12.00/ 9.00	OK	PROC., COMPUTER-AIDED SEISMIC ANALYSIS AND DISCRIMINATION (1977 CONF.)	1977	P	122	77CH1224-3
014	20.00/15.00	M/F O	PROC., COMPUTERS IN CARDIOLOGY (1974 CONF.)	1974	P	235	74CH0879-7
015	20.00/15.00	M/F O	PROC., COMPUTERS IN CARDIOLOGY (1975 CONF.)	1975	P	286	75CH1018-1
016	20.00/15.00	OK	PROC., COMPUTERS IN CARDIOLOGY (1976 CONF.)	1976	P	458	76CH1160-1
193	25.00/18.75	OK	PROC., COMPUTERS IN CARDIOLOGY (1977 CONF.)	1977	P	650	77CH1254-2
224	25.00/18.75	OK	PROC., COMPUTERS IN CARDIOLOGY (1978 CONF.)	1978	P	436	78CH1391-2
235	30.00/22.50	OK	PROC., COMPUTERS IN CARDIOLOGY (1979 CONF.)	1979	P	488	79CH1462-1
324	30.00/22.50	OK	PROC., COMPUTERS IN CARDIOLOGY (1980 CONF.)	1980	P	450	80CH1606-3
384		02/82	PROC., COMPUTERS IN CARDIOLOGY (1981 CONF.)	1981	P		
285	20.00/15.00	OK	PROC., COMPUTERS IN OPHTHALMOLOGY (1979 CONF.)	1979	P	272	79CH1517-2
203	20.00/15.00	M/F O	PROC., CONFERENCE ON COMPUTING IN THE 1980'S (OREGON REPORT)	1978	P	266	

ORDER NO	PRICES NM/M	AVAIL	TITLES	YEAR PUBL	CODE	PAGES	IEEE NO
049	20.00/15.00	OK	PROC., DATA COMMUNICATIONS SYMPOSIUM (3RD CONF.)	1973	P	160	73CH0828-4
027	20.00/15.00	M/F O	PROC., DATA COMMUNICATIONS SYMPOSIUM (4TH CONF.)	1975	P	180	75CH1001-7
031	20.00/15.00	OK	PROC., DATA COMMUNICATIONS SYMPOSIUM (5TH CONF.)	1977	P		77CH1260-9
230	20.00/15.00	M/F O	PROC., DATA COMMUNICATIONS SYMPOSIUM (6TH CONF.)	1979	P	210	79CH1405-0
374	25.00/18.75	11/81	PROC., DATA COMMUNICATIONS SYMPOSIUM (7TH CONF.)	1981	P	260	81CH1694-9
034	12.50/10.00	OK	PROC., DESIGN AUTOMATION (9TH CONF.)	1972	P	375	72CH0706-2
024	20.00/15.00	OK	PROC., DESIGN AUTOMATION (11TH CONF.)	1974	P	379	74CH0865-6
025	20.00/15.00	OK	PROC., DESIGN AUTOMATION (13TH CONF.)	1976	P	501	76CH1098-3
154	25.00/18.75	OK	PROC., DESIGN AUTOMATION (14TH CONF.)	1977	P	507	77CH1216-1
178	25.00/18.75	OK	PROC., DESIGN AUTOMATION (15TH CONF.)	1978	P	493	78CH1363-1
240	25.00/18.75	M/F O	PROC., DESIGN AUTOMATION (16TH CONF.)	1979	P	567	79CH1427-4
302	44.00/33.00	OK	PROC., DESIGN AUTOMATION (17TH CONF.)	1980	P	642	80CH1550-3
361	44.00/33.00	OK	PROC., DESIGN AUTOMATION (18TH CONF.)	1981	P	900	81CH1643-6
136	12.00/ 9.00	OK	PROC., DESIGN AUTOMATION AND MICROPROCESSOR SYMPOSIUM	1977	P	110	77CH1189
270	25.00/18.75	M/F O	PROC., DISTRIBUTED COMPUTING (1ST CONF.)	1979	P	796	79CH1408-4
344	36.00/27.00	OK	PROC., DISTRIBUTED COMPUTING SYSTEMS (2ND CONF.)	1981	P	524	81CH1591-7
329	20.00/15.00	OK	PROC., DISTRIBUTED DATA ACQUISITION AND CONTROL (1ST CONF.)	1980	P	212	80CH1571-9
040	20.00/15.00	OK	PROC., FAULT-TOLERANT COMPUTING (FTCS--5)	1975	P	275	75CH0974-6
042	20.00/15.00	M/F O	PROC., FAULT-TOLERANT COMPUTING (FTCS--6)	1976	P	206	76CH1094-2
156	20.00/15.00	OK	PROC., FAULT-TOLERANT COMPUTING (FTCS--7)	1977	P	217	77CH1223-7
180	25.00/18.75	OK	PROC., FAULT-TOLERANT COMPUTING (FTCS--8)	1978	P	226	78CH1286-4
239	22.00/16.50	OK	PROC., FAULT-TOLERANT COMPUTING (FTCS--9)	1979	P	236	79CH1396-1
336	25.00/18.75	OK	PROC., FAULT-TOLERANT COMPUTING (FTCS-10)	1980	P	400	80CH1604-8
350	25.00/18.75	OK	PROC., FAULT-TOLERANT COMPUTING (FTCS-11)	1981	P	290	81CH1600-6
011	20.00/15.00	OK	PROC., FOUNDATIONS OF COMPUTER SCIENCE (16TH SYMP.)	1975	P	200	75CH1003-3
132	20.00/15.00	OK	PROC., FOUNDATIONS OF COMPUTER SCIENCE (18TH SYMP.)	1977	P	269	77CH1278-1
205	22.00/16.50	OK	PROC., FOUNDATIONS OF COMPUTER SCIENCE (19TH SYMP.)	1978	P	290	78CH1397-9
251	22.00/16.50	OK	PROC., FOUNDATIONS OF COMPUTER SCIENCE (20TH SYMP.)	1979	P	440	79CH1471-1
323	30.00/22.50	OK	PROC., FOUNDATIONS OF COMPUTER SCIENCE (21ST SYMP.)	1980	P	422	80CH1498-5
376	30.00/22.50	11/81	PROC., FOUNDATIONS OF COMPUTER SCIENCE (22ND SYMP.)	1981	P	430	81CH1695-6
200	25.00/18.75	OK	PROC., IECI '78 (INDUSTRIAL APPLICATIONS OF MICROPROCESSORS)	1978	P	233	78CH1312-8
091	25.00/18.75	OK	PROC., INT'L CONF. ON PATTERN RECOGNITION (1ST ICPR)	1973	P	600	73CH0821-9
081	25.00/18.75	OK	PROC., INT'L CONF. ON PATTERN RECOGNITION (2ND ICPR)	1974	P	550	74CH0885-4
083	25.00/18.75	M/F O	PROC., INT'L CONF. ON PATTERN RECOGNITION (3RD ICPR)	1976	P	884	76CH1140-3
183	40.00/30.00	OK	PROC., INT'L CONF. ON PATTERN RECOGNITION (4TH ICPR)	1978	P	1170	78CH1331-8
247	48.00/36.00	OK	PROC., INT'L CONF. ON PATTERN RECOGNITION (5TH ICPR)	1980	P	1425	80CH1499-3
179	25.00/18.75	OK	PROC., INTERACTIVE TECHNIQUES IN COMPUTER-AIDED DESIGN SYMPOSIUM	1978	P	479	78CH1289-8
292	16.00/12.00	OK	PROC., INTERCONNECTION NETWORKS WORKSHOP	1980	P	124	80CH1560-2
160	16.00/12.00	OK	PROC., JOINT COLLEGE CURRICULA WKSHP ON COMP. SCIENCE, ENGRING (4TH)	1978	P	138	78CH1311-0
214	16.00/12.00	OK	PROC., JOINT ENGINEERING MANAGEMENT (26TH CONF.)	1978	P	173	78CH1359-9
276	16.00/12.00	M/F O	PROC., LOCAL COMPUTER NETWORKS (4TH CONF.)	1979	P	148	79CH1446-4
320	20.00/15.00	OK	PROC., LOCAL COMPUTER NETWORKS (5TH CONF.)	1980	P	130	80CH1542-0
382	16.00/12.00	11/81	PROC., LOCAL COMPUTER NETWORKS (6TH CONF.)	1981	P	114	81CH1690-7
093	25.00/18.75	OK	PROC., MACHINE PROCESSING OF REMOTELY SENSED DATA (1973 CONF.)	1973	P	422	73CH0834-2
094	25.00/18.75	OK	PROC., MACHINE PROCESSING OF REMOTELY SENSED DATA (1975 CONF.)	1975	P	315	75CH1009-0
095	25.00/18.75	OK	PROC., MACHINE PROCESSING OF REMOTELY SENSED DATA (1976 CONF.)	1976	P	238	76CH1103-1
122	25.00/18.75	OK	PROC., MACHINE PROCESSING OF REMOTELY SENSED DATA (1977 CONF.)	1977	P	358	77CH1218-7
256	25.00/18.75	OK	PROC., MACHINE PROCESSING OF REMOTELY SENSED DATA (1979 CONF.)	1979	P	472	79CH1430-8
306	30.00/22.50	OK	PROC., MACHINE PROCESSING OF REMOTELY SENSED DATA (1980 CONF.)	1980	P	368	80CH1533-9
359	12.00/ 9.00	OK	PROC., MASS STORAGE SYSTEMS SYMPOSIUM (4TH SYMP.)	1980	P	76	80CH1581-8
290	16.00/12.00	OK	PROC., MICRO APPLICATIONS IN THE '80S (ARIZONA TECHNICAL SYMP.)	1980	P	105	80CH1525-5
192	12.00/ 9.00	OK	PROC., MICRO-DELCON '78	1978	P	68	78CH1330-0
246	14.00/10.50	OK	PROC., MICRO-DELCON '79	1979	P	122	79CH1426-6
296	16.00/12.00	OK	PROC., MICRO-DELCON '80	1980	P	141	80CH1528-9
343	16.00/12.00	OK	PROC., MICRO-DELCON '81	1981	P	162	81CH1628-7
061	20.00/15.00	OK	PROC., MICROCOMPUTER '77 CONFERENCE	1977	P	274	77CH1185-8
280	16.00/12.00	OK	PROC., MICROCOMPUTER FIRMWARE AND I/O WORKSHOP (1979)	1979	P	240	THO065-3
162	13.00/ 9.75	OK	PROC., MICROCOMPUTER-BASED INSTRUMENTATION	1978	P	104	78CH1303-7
298	16.00/12.00	OK	PROC., MICROPROCESSOR APPLICATIONS IN MILITARY & INDUSTRIAL SYSTEMS	1980	P	175	80CH1579-2
360	16.00/12.00	OK	PROC., MICROPROCESSORS AND EDUCATION WORKSHOP (1980)	1980	P	130	THO083-6
106	7.50/ 5.00	OK	PROC., MICROPROGRAMMING WORKSHOP (MICRO--5)	1972	P	98	
110	12.00/ 9.00	OK	PROC., MICROPROGRAMMING WORKSHOP (MICRO--8)	1975	P	107	75CH1053-8
111	8.00/ 6.00	OK	PROC., MICROPROGRAMMING WORKSHOP (MICRO--9)	1976	P	61	76CH1148-6
053	12.00/ 9.00	OK	PROC., MICROPROGRAMMING WORKSHOP (MICRO-10)	1977	P	133	77CH1266-6
204	14.00/10.50	OK	PROC., MICROPROGRAMMING WORKSHOP (MICRO-11)	1978	P	160	78CH1411-8
248	16.00/12.00	OK	PROC., MICROPROGRAMMING WORKSHOP (MICRO-12)	1979	P	160	79CH1516-4
327	16.00/12.00	OK	PROC., MICROPROGRAMMING WORKSHOP (MICRO-13)	1980	P	190	80CH1599-0
373	20.00/15.00	OK	PROC., MICROPROGRAMMING WORKSHOP (MICRO-14)	1981	P	214	81CH1691-5
279	22.00/16.50	OK	PROC., MICROS AND MINICOMPUTERS (1979 CONF.)	1979	P	356	79CH1474-6
135	20.00/15.00	OK	PROC., MINI AND MICROCOMPUTERS SYMPOSIUM (MIMI '76)	1976	P	250	76CH1180-9
182	20.00/15.00	OK	PROC., MINI AND MICROCOMPUTERS SYMPOSIUM (MIMI '77)	1977	P	316	77CH1347-4
115	5.00/ 5.00	OK	PROC., MODELING & ANALYSIS OF DATA NETWORKS SYMPOSIUM	1976	P	92	
153	20.00/15.00	OK	PROC., MULTIPLE-VALUED LOGIC (7TH CONF.)	1977	P	155	77CH1222-9
177	20.00/15.00	OK	PROC., MULTIPLE-VALUED LOGIC (8TH CONF.)	1978	P	298	78CH1366-4
238	22.00/16.50	OK	PROC., MULTIPLE-VALUED LOGIC (9TH CONF.)	1979	P	304	79CH1408-4
295	25.00/18.75	OK	PROC., MULTIPLE-VALUED LOGIC (10TH CONF.)	1980	P	277	80CH1577-6
348	25.00/18.75	OK	PROC., MULTIPLE-VALUED LOGIC (11TH CONF.)	1981	P	294	80CH1611-3
163	16.00/12.00	OK	PROC., OCEANIC DATA BASE INFORMATION EXCHANGE WORKSHOP	1977	P	147	EHO134-7
088	16.00/12.00	OK	PROC., OPTICAL COMPUTING CONFERENCE (1974 CONF.)	1974	P	104	74CH0862-3
084	16.00/12.00	OK	PROC., OPTICAL COMPUTING CONFERENCE (1975 CONF.)	1975	P	169	75CH0941-5
100	16.00/12.00	OK	PROC., OPTICAL COMPUTING CONFERENCE (1976 CONF.)	1976	P	150	76CH1100-7
185	20.00/15.00	OK	PROC., OPTICAL COMPUTING CONFERENCE (1978 CONF.)	1978	P	224	78CH1305-2
319	30.00/22.50	OK	PROC., OPTICAL COMPUTING CONFERENCE (1980 CONF.)	1980	P	232	80CH1548-7
107	20.00/15.00	OK	PROC., PARALLEL PROCESSING (1975 CONF.)	1975	P	255	
116	20.00/15.00	OK	PROC., PARALLEL PROCESSING (1976 CONF.)	1976	P	328	76CH1127-0
108	20.00/15.00	OK	PROC., PARALLEL PROCESSING (1977 CONF.)	1977	P	256	77CH1253-4
175	22.00/16.50	OK	PROC., PARALLEL PROCESSING (1978 CONF.)	1978	P	269	78CH1321-9
234	22.00/16.50	OK	PROC., PARALLEL PROCESSING (1979 CONF.)	1979	P	279	79CH1433-0
317	25.00/18.75	OK	PROC., PARALLEL PROCESSING (1980 CONF.)	1980	P	386	80CH1569-3
354	25.00/18.75	OK	PROC., PARALLEL PROCESSING (1981 CONF.)	1981	P	360	81CH1634-5
079	25.00/18.75	OK	PROC., PATTERN RECOGNITION & IMAGE PROCESSING (PRIP '75)	1975	P	440	75CH0981-1
120	20.00/15.00	OK	PROC., PATTERN RECOGNITION & IMAGE PROCESSING (PRIP '77)	1977	P	297	77CH1208-9
184	25.00/18.75	OK	PROC., PATTERN RECOGNITION & IMAGE PROCESSING (PRIP '78)	1978	P	516	78CH1318-5
232	25.00/18.75	M/F O	PROC., PATTERN RECOGNITION & IMAGE PROCESSING (PRIP '79)	1979	P	664	79CH1428-2
352	40.00/30.00	OK	PROC., PATTERN RECOGNITION & IMAGE PROCESSING (PRIP '81)	1981	P	626	81CH1595-8
085	12.00/ 9.00	M/F O	PROC., PATTERN RECOGNITION AND ARTIFICIAL INTELLIGENCE WORKSHOP	1976	P	182	76CH1169-2
124	12.00/ 9.00	OK	PROC., PATTERN RECOGNITION APPLIED TO OIL IDENTIFICATION WKSHP	1976	P	173	76CH1247-6

ORDER NO	PRICES NM/M	AVAIL	TITLES	YEAR PUBL	CODE	PAGES	IEEE NO
392	22.00/16.50	11/81	PROC., PERS. COMPUTERS TO AID HANDICAPPED; JOHNS HOPKINS 1ST SEARCH FO	1981	P	322	THO092-7
121	16.00/12.00	OK	PROC., PICTURE DATA DESCRIPTION & MANAGEMENT WORKSHOP	1977	P	196	77CH1187-4
316	25.00/18.75	OK	PROC., PICTURE DATA DESCRIPTION AND MGT (1980 WKSHP)	1980	P	304	80CH1530-5
284	20.00/15.00	OK	PROC., QUANTITATIVE SOFTWARE MODELS WORKSHOP	1979	P	237	THO067-9
381		12/81	PROC., REAL TIME SYSTEMS SYMPOSIUM	1981	P		81CH1700-4
351	20.00/15.00	OK	PROC., RELIABILITY IN DISTRIBUTED SOFTWARE & DATABASE SYSTEMS SYMP.	1981	P	225	81CH1632-9
052	12.00/ 9.00	OK	PROC., ROCKY MOUNTAIN SYMPOSIUM (1ST CONF.)	1977	P	310	
181	16.00/12.00	OK	PROC., ROCKY MOUNTAIN SYMPOSIUM (2ND CONF.)	1978	P	404	78CH1387-0
307	16.00/12.00	OK	PROC., ROCKY MOUNTAIN SYMPOSIUM (3RD CONF.)	1979	P	155	79CH1463-9
335	14.00/10.50	OK	PROC., SECURITY AND PRIVACY (1980 SYMP.)	1980	P	175	80CH1522-2
345	16.00/12.00	OK	PROC., SECURITY AND PRIVACY (1981 SYMP.)	1981	P	180	81CH1629-5
007	20.00/15.00	M/F O	PROC., SIMULATION SYMPOSIUM (8TH CONF.)	1975	P	324	75CH0984-5
008	20.00/15.00	M/F O	PROC., SIMULATION SYMPOSIUM (9TH CONF.)	1976	P	375	76CH1055-3
140	20.00/15.00	OK	PROC., SIMULATION SYMPOSIUM (10TH CONF.)	1977	P	413	77CH1177-5
164	20.00/15.00	OK	PROC., SIMULATION SYMPOSIUM (11TH CONF.)	1978	P	368	78CH1327-6
222	22.00/16.50	OK	PROC., SIMULATION SYMPOSIUM (12TH CONF.)	1979	P	350	79CH1376-3
289	22.00/16.50	OK	PROC., SIMULATION SYMPOSIUM (13TH CONF.)	1980	P	338	80CH1492-8
333	22.00/16.50	OK	PROC., SIMULATION SYMPOSIUM (14TH CONF.)	1981	P	297	81CH1590-9
103	8.00/ 6.00	OK	PROC., SOFTWARE ENGINEERING (1ST INT'L CONF.)	1975	P	100	75CH0992-8
104	20.00/15.00	OK	PROC., SOFTWARE ENGINEERING (2ND INT'L CONF.)	1976	P	700	76CH1125-4
187	20.00/15.00	M/F O	PROC., SOFTWARE ENGINEERING (3RD INT'L CONF.)	1978	P	341	78CH1317-7
249	22.00/16.50	OK	PROC., SOFTWARE ENGINEERING (4TH INT'L CONF.)	1979	P	464	79CH1479-5
332	30.00/22.50	OK	PROC., SOFTWARE ENGINEERING (5TH INT'L CONF.)	1981	P	472	81CH1627-9
353	16.00/12.00	OK	PROC., SOFTWARE ENGINEERING STANDARDS APPLICATIONS WORKSHOP	1981	P	152	81CH1633-7
208	16.00/12.00	OK	PROC., SOFTWARE LIFE-CYCLE MANAGEMENT WORKSHOP (2ND CONF.)	1978	P	220	78CH1390-4
001	20.00/15.00	OK	PROC., SOUTHEASTERN SYMPOSIUM ON SYSTEMS THEORY (7TH CONF.)	1975	P	342	75CH0968-8
002	20.00/15.00	M/F O	PROC., SOUTHEASTERN SYMPOSIUM ON SYSTEMS THEORY (8TH CONF.)	1976	P	340	76CH1093-4
229	22.00/16.50	OK	PROC., SOUTHEASTERN SYMPOSIUM ON SYSTEMS THEORY (11TH CONF.)	1979	P	249	THO061-2
294	25.00/18.75	OK	PROC., SOUTHEASTERN SYMPOSIUM ON SYSTEMS THEORY (12TH CONF.)	1980	P	395	80CH1576-8
221	20.00/15.00	OK	PROC., SPECIFICATIONS OF RELIABLE SOFTWARE CONFERENCE	1979	P	237	79CH1401-9
142	12.00/ 9.00	OK	PROC., TEST CONFERENCE (1973 CHERRY HILL TEST SYMP.)	1973	P	125	73CH0827-6
054	16.00/12.00	OK	PROC., TEST CONFERENCE (1974 CHERRY HILL TEST SYMP.)	1974	P	261	74CH0909-2
060	16.00/12.00	OK	PROC., TEST CONFERENCE (1975 CHERRY HILL TEST SYMP.)	1975	P	110	75CH1041-3
141	16.00/12.00	OK	PROC., TEST CONFERENCE (1976 CHERRY HILL TEST SYMP.)	1976	P	105	76CH1179-1
147	16.00/12.00	OK	PROC., TEST CONFERENCE (1977 CHERRY HILL TEST SYMP.)	1977	P	198	77CH1261-7
210	20.00/15.00	OK	PROC., TEST CONFERENCE (1978 CHERRY HILL TEST SYMP.)	1978	P	301	78CH1409-2
257	16.00/12.00	OK	TUTORIAL: AUTOMATED TOOLS FOR SOFTWARE ENGINEERING	1979	T	270	EHO150-3
261	25.00/18.75	OK	TUTORIAL: CENTRALIZED & DISTRIBUTED DATA BASE SYSTEMS	1979	T	262	EHO154-5
201	16.00/12.00	OK	TUTORIAL: COMPUTER COMMUNICATION PROTOCOLS; A PRACTICAL VIEW OF	1978	T	264	EHO137-0
233	28.00/21.00	OK	TUTORIAL: COMPUTER GRAPHICS	1979	T	433	EHO147-9
297	20.00/15.00	OK	TUTORIAL: COMPUTER NETWORKS (2ND ED.)	1980	T	520	EHO162-8
139	12.95/ 9.95	OK	TUTORIAL: COMPUTER SECURITY AND INTEGRITY	1977	T	448	EHO124-8
313	20.00/15.00	OK	TUTORIAL: COMPUTER SYSTEM REQUIREMENTS	1980	T	364	EHO168-5
325	25.00/18.75	OK	TUTORIAL: COMPUTER SYSTEMS FOR PROCES. DIAGNOSTIC ELECTROCARDIOGRAMS	1980	T	228	EHO170-1
242	12.00/ 9.00	OK	TUTORIAL: COMPUTER-AIDED DESIGN TOOLS FOR DIGITAL SYSTEMS	1979	T	174	EHO132-1
334	25.00/18.75	OK	TUTORIAL: COMPUTERS & BUSINESS	1981	T	472	EHO177-6
369	25.00/18.75	OK	TUTORIAL: DATA BASE MANAGEMENT IN THE '80S	1981	T	472	EHO181-8
260	16.00/12.00	OK	TUTORIAL: DESIGN OF MICROPROCESSOR SYSTEMS	1978	T	262	EHO155-2
199	25.00/18.75	M/F O	TUTORIAL: DIGITAL IMAGE PROCESSING AND SELECTED PAPERS	1979	T	732	EHO133-9
269	20.00/15.00	OK	TUTORIAL: DISTRIBUTED CONTROL	1979	T	382	EHO153-7
212	14.00/10.50	OK	TUTORIAL: DISTRIBUTED DATA BASE MANAGEMENT	1978	T	206	EHO141-2
209	16.00/12.00	M/F O	TUTORIAL: DISTRIBUTED PROCESSING (2ND ED.)--SEE #363	1978	T	450	EHO127-1
363	25.00/18.75	OK	TUTORIAL: DISTRIBUTED PROCESSING (3RD ED.)	1981	T	640	EHO176-8
299	28.00/21.00	OK	TUTORIAL: DISTRIBUTED PROCESSING SYSTEMS; A PRAGMATIC VIEW OF	1980	T	616	EHO160-2
258	20.00/15.00	OK	TUTORIAL: DISTRIBUTED PROCESSOR COMMUNICATION ARCHITECTURE	1979	T	526	EHO152-9
267	20.00/15.00	OK	TUTORIAL: DISTRIBUTED SYSTEM DESIGN	1979	T	414	EHO151-1
390	30.00/22.50	11/81	TUTORIAL: HUMAN FACTORS IN SOFTWARE DEVELOPMENT	1981	T	700	EHO185-9
266	25.00/18.75	OK	TUTORIAL: INTERACTIVE COMPUTER GRAPHICS	1979	T	422	EHO156-0
338	28.00/21.00	OK	TUTORIAL: INTRO TO AUTOMATED ARRHYTHMIA DETECTION	1980	T	332	EHO171-9
368	20.00/15.00	OK	TUTORIAL: LOCAL COMPUTER NETWORKS (2ND ED)	1981	T	372	EHO179-2
304	20.00/15.00	OK	TUTORIAL: LOCAL COMPUTER NETWORKS--SEE #368	1980	T	345	EHO163-6
215	12.00/ 9.00	OK	TUTORIAL: LSI TESTING (2ND ED.)	1978	T	190	EHO122-2
213	12.00/ 9.00	OK	TUTORIAL: MICROCOMPUTER PROGRAMMING AND SOFTWARE SUPPORT	1978	T	200	EHO140-4
259	16.00/12.00	OK	TUTORIAL: MICROCOMPUTER SYSTEM DESIGN AND TECHNIQUES	1979	T	432	EHO159-4
340	14.00/10.50	OK	TUTORIAL: MICROCOMPUTER SYSTEM SOFTWARE AND LANGUAGES	1980	T	232	EHO174-3
109	12.00/ 9.00	OK	TUTORIAL: MICROPROGRAMMING	1975	T	318	75CH1033-0
310	20.00/15.00	OK	TUTORIAL: MODELS & METRICS FOR SOFTWARE MGT AND ENGINEERING	1980	T	352	EHO167-7
339	14.00/10.50	OK	TUTORIAL: OFFICE AUTOMATION SYSTEMS	1980	T	201	EHO172-7
367	25.00/18.75	OK	TUTORIAL: PARALLEL PROCESSING	1981	T	498	EHO182-6
312	20.00/15.00	OK	TUTORIAL: PROGRAMMING LANGUAGE DESIGN	1980	T	536	EHO164-4
391	28.00/21.00	11/81	TUTORIAL: PROGRAMMING PRODUCTIVITY--ISSUES FOR THE EIGHTIES	1981	T	520	EHO186-7
130	13.50/10.00	M/F O	TUTORIAL: QUANTITATIVE MANAGEMENT--SOFTWARE COST ESTIMATING	1977	T	330	EHO129-7
366	20.00/15.00	OK	TUTORIAL: SECURITY OF DATA IN NETWORKS	1981	T	250	EHO183-4
309	20.00/15.00	OK	TUTORIAL: SOFTWARE CONFIGURATION MANAGEMENT	1980	T	452	EHO169-3
314	20.00/15.00	OK	TUTORIAL: SOFTWARE COST ESTIMATING AND LIFE-CYCLE CONTROL (GETTING THE	1980	T	356	EHO165-1
385	25.00/18.75	11/81	TUTORIAL: SOFTWARE DESIGN ENVIRONMENTS	1981	T		
268	16.00/12.00	OK	TUTORIAL: SOFTWARE DESIGN STRATEGIES	1979	T	426	EHO149-5
389	25.00/18.75	11/81	TUTORIAL: SOFTWARE DESIGN STRATEGIES (2ND ED.)	1981	T	530	EHO184-2
300	20.00/15.00	OK	TUTORIAL: SOFTWARE DESIGN TECHNIQUES (3RD ED.)	1980	T	425	EHO161-0
219	16.00/12.00	OK	TUTORIAL: SOFTWARE MANAGEMENT	1979	T	390	EHO146-1
211	16.00/12.00	OK	TUTORIAL: SOFTWARE METHODOLOGY	1978	T	464	EHO142-0
311	20.00/15.00	OK	TUTORIAL: SOFTWARE SYSTEM DESIGN--DESCRIPTION AND ANALYSIS	1980	T	258	EHO166-9
207	16.00/12.00	OK	TUTORIAL: SOFTWARE TESTING & VALIDATION TECHNIQUES (2ND ED.)	1978	T	430	EHO138-8
365	25.00/18.75	11/81	TUTORIAL: SOFTWARE TESTING & VALIDATION TECHNIQUES (3RD ED.)	1981	T	464	EHO180-0
362	20.00/15.00	OK	TUTORIAL: STRUCTURED PROGRAMMING: INTEGRATED PRACTICES	1981	T	290	EHO178-4
386		03/82	TUTORIAL: VLSI DESIGN & USE; TECHNOLOGY YOU NEED FOR	1982	T		
288	22.00/16.50	OK	TUTORIAL: VLSI--THE COMING REVOLUTION IN APPLICATIONS & DESIGN	1979	T	316	EHO158-7

NOTES: ALL BOOKS ARE AVAILABLE IN MICROFICHE OR IN SOFTCOVER AT PRESS TIME OR BY DATE SPECIFIED
BOOKS DESIGNATED "M/F" ARE OUT OF PRINT AND AVAILABLE ONLY IN MICROFICHE

NM = NON-MEMBER PRICE; M = MEMBER PRICE; N/A = NOTAVAILABLE AT PRESS TIME
ALL BOOKS ARE SUBJECT TO AVAILABILITY AND PRICES SUBJECT TO CHANGE WITHOUT NOTICE

PUBLICATIONS ORDER FORM

IEEE Computer Society
P.O. Box 80452
Worldway Postal Center
Los Angeles, CA 90080

Catalog No. To assure prompt processing of your order, be sure to enter the 3-digit number that appears ahead of each publication description.

IMPORTANT INFORMATION ON DISCOUNTS AND SHIPPING METHODS

- Rush handling $10 00 PER ORDER
- All unit prices include postage for 4th class book rate Overseas mail is shipped sea mail (10-12 weeks delivery) For priority shipping to U S or Canada, add $5 00 per book For airmail service (2 week delivery) to Mexico and all other foreign countries, please add $15 00 per book
- Remember, member rates apply on the **first copy for personal use only.** Additional copies of the same title are sold at full list price.
- **Overseas orders must be prepaid.** Payments must be made in U S funds drawn on U S banks
- No refunds or returns accepted after 60 days of shipment (90 days overseas)
- Occasionally, books are no longer available. In such cases, will you accept microfiche at same price? No___Yes___
- **Prices subject to change without notice.** Books subject to availability
- Minimum telephone order — $50 00

Order Number	Quantity	Title/Description	Unit Price		Amount
			Member	Non-Member	

Charge Card Number *Expiration Date* *Signature*

Name (please print) *Member No.*

Address

City/State/Zip/Country

*Phone No.*_____ *Purchase Order No.*_____

Subtotal _____

California residents add 6% tax _____

Overseas purchasers: Remit U.S dollars on U S. Bank. **TOTAL $** _____

☐ Check Enclosed
☐ Bill Visa/BankAmericard
☐ Bill Master Charge

Optional Shipping Charge _____

Total $ _____