Database machines and some issues on DBMS standards*

by STANLEY Y. W. SU,
University of Florida

HSU CHANG,
IBM
Yorktown Heights

GEORGE COPELAND,
Tektronix

PAUL FISHER,
Kansas State University

EUGENE LOWENTHAL,
MRI Systems

and

STEWART SCHUSTER,
TANDEM

INTRODUCTION

There are several co-related activities in the database area and computer architecture that make the discussion of database machines and their implications on DBMS standards timely and meaningful. First, in the database area there is a drive toward more powerful database management systems which support high-level data models and languages. The motive for this drive is the requirement to greatly improve user/programmer productivity and to protect applications from changes in the user environment. However, supporting these interfaces with software means often introduces inefficiency in database management systems because of the many levels of complex software which are required to map the high-level data representation and languages to the low level storage representation and machine codes. Second, the need for systems which handle very large databases is increasing rapidly. Very large databases complicate the problems of retrieval, update, data recovery, transaction processing, integrity, and security. Software solutions to these problems work well for both small databases supporting many applications and large databases supporting only a few applications. However, the labor-intensive cost, time delays and reliability problems associated with software development and maintenance will soon become prohibitive as large and highly shared databases emerge. The search for hardware solutions to these problems is a necessary and viable alternative for balancing functionality and price/performance. Third, the progress made in hardware technology in the past decade is phenomenal. The cost of memories, processors, terminals and communication devices has dropped and will continue to drop at a drastic rate. It is time for a reevaluation of the traditional role of hardware and software in solving problems of today and tomorrow in database management.

Fourth, there is a vigorous drive toward DBMS standards led by NBS (26,27) aiming to "1) protect the federal investment in existing data, programs, and personnel skills, 2) improve the productivity and effectiveness of database systems available to federal agencies, 3) assist federal agencies with guidelines on the selection, procurement, use, and availability of database systems, 4) perform the research necessary to identify future federal needs and to foster the development of necessary database tools."

Research on database machines is relevant to the study of DBMS standards in the following ways. First, when a standard is to be proposed for adoption it is important to consider how easy the standard can be implemented and the cost involved in its implementation. Database machines may drastically change the ways database management functions are implemented and new technologies may alter the picture of cost involved in database management. A standard is not practical unless it can be implemented with efficiency and reliability. Database machines hold promise to provide more efficient and reliable ways to implement the database functions. Second, very often several alternative designs (e.g. data models or data languages) exist and can be the candidates for standards. Good evaluation and proper selection of these alternatives based on criteria such as "user/programmer productivity," "ease of use," "natural to the user and DBA," etc., are extremely difficult to obtain. In this situation, the selection of one of the alternatives as the standard can be based on, among other variables, how well the se-

* This work is supported by the National Bureau of Standards under contract #NB 79NAA B4369-1.
lected standard is supported by the present database machines and can be supported by the future machines. Third, the change of hardware architecture of a computing machine will have great effect on the design and implementation of a database management system. In particular, new hardware may change the interfaces among the components of a DBMS. Thus, the study of the standards for DBMS interfaces should take into consideration the present and expected progress in database machine research and development.

This paper reports on the results of a study conducted under the support of the National Bureau of Standards (contract #NB 79NA A B4369-1) to examine some of the proposed DBMS standards from the point of view of database machines. The emphasis is on the discussion of several issues related to data models and data languages and on how they can be supported by database machines. The study aims to: 1) assess the progress made in the database machine area, 2) determine the functional capabilities and limitations of the present database machines, 3) examine the issues on DBMS architecture, data models, and data languages from the point of view of present and future database machines, and 4) address some technical issues on the technology, the hardware, and software architectures of database machines.

II. SOME LIMITATIONS OF CONVENTIONAL COMPUTERS FOR DATABASE APPLICATIONS

Several limitations found in the conventional computers motivate the study of database machines. They are:

A. Mismatch of conventional computers for database applications

In 1948, von Neumann designed the programmable electronic digital computer for numeric applications. The design matched the technology of that day very closely to numeric applications. The semantic definition of numeric data was matched very closely to the storage representation:

<table>
<thead>
<tr>
<th>data semantics</th>
<th>random access storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, 26</td>
<td>location 1, 26</td>
</tr>
<tr>
<td>y, -5</td>
<td>location 2, -5</td>
</tr>
<tr>
<td>z, 1.7 x 10^23</td>
<td>location 3, 1.7 x 10^23</td>
</tr>
</tbody>
</table>

Using a random access storage, only a simple and efficient one-to-one mapping was necessary. Also, the semantics of numeric operations were matched very closely to the hardware instructions:

<table>
<thead>
<tr>
<th>semantics of operations</th>
<th>hardware instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>ADD</td>
</tr>
<tr>
<td>subtract</td>
<td>SUB</td>
</tr>
<tr>
<td>store in memory</td>
<td>STR</td>
</tr>
</tbody>
</table>

This close match allowed a very simple and efficient mapping. However, two significant things have happened since that time. First, hardware technology has changed drastically. Cost and speed per function have improved by many orders of magnitude in the last 30 years. The rules of cost-effective packaging have changed from minimization of the number of logic gates and memory bits to minimization of the number of IC pins and packages. Secondly, the primary application for digital computers is shifting from numeric to non-numeric applications. In non-numeric applications, the user retrieves and manipulates data by specifying the attributes and values of the data he is interested in, i.e. addressing data by contents, rather than by addressing the memory locations where the interested data are stored. The basic operations required are Search, Retrieve, Update, Insert, Delete, Move-data, etc., rather than Add, Subtract, Shift, etc. The mismatch of the von Neumann design to non-numeric applications is the main cause of the complexity and inefficiencies of the present systems. The large and growing market for database systems warrants a reevaluation of the relationship between technology and database applications. If technology can be matched more closely to database applications, then perhaps advanced functionality, ease-of-use, and data independence can be achieved cost effectively.

In all considerations however, the most serious drawback is the lack of appropriateness of the sequential machine for the parallel process of data manipulation. One can liken this to the analogy of viewing a three-dimensional cube on a two-dimensional surface. Some forms are still recognizable; however, many others are skewed and hence do not appear 'normal.' So it is with processors. The software problems become necessarily more complex, simply because the representation is not appropriate. By providing a more appropriate environment, perhaps the 'skewness' of present problems can be reduced.

B. Many levels of mapping

Recent research efforts show that high-level data models and data languages which exhibit a high degree of data independence and ease of use are requirements to improve human productivity as well as act as logical interfaces with database systems. Currently, the implementation of high-level data languages and data models requires many levels of complex software to be executed, causing inefficiencies in system utilization and response. The software complexity and system inefficiency are due to the requirement that high-level commands and data views be translated into the low level machine codes and structures. In particular, software implementation of high-level data representations requires that auxiliary data structures such as inverted files, directories, and pointers, etc., be introduced to speed up data accesses for a particular set of applications. These auxiliary data structures must be properly maintained. This requirement complicates the updating operation, one of the most important database management functions, and significantly decreases its efficiency. Also, since these auxiliary data structures are tailored for a particular set of applications, a change in application often requires a large labor-intensive
software maintenance project. This considerably increases cost and time delays and decreases reliability.

C. Performance bottlenecks

There seem to be two major performance bottlenecks in the present systems: the staging bottleneck and the communication bottleneck. In conventional systems, data are not stored at the place where they are processed. To “stage” data into main memory for processing is very time consuming, and often ties up the important resources of a computing system, such as communication channels. Database applications will continue to demand larger and more complicated databases, requiring more time to stage and process the data files. In order to support very large databases (greater than \(10^{10}\) bytes), or databases requiring both fast update and/or complex query, it is necessary to exploit specialized hardware to eliminate unnecessary data staging and to carry out database management functions efficiently. Data communication over long distances is expensive and limited in speed. This forces many database systems to physically distribute data to locations where usage is highest. Data redundancy is often purposefully introduced in distributed systems to avoid excess amounts of data transfer and to improve performance and reliability. However, many additional problems on data updating, recovery, integrity, and security in distributed systems are introduced by the above techniques. Special purpose hardware tailored toward managing distributed database management and supporting data communication would be very useful.

D. User’s increasing demands for DBMS capabilities

Database management system users are continuously demanding more sophisticated DBMS capabilities. Capabilities such as automatic database restructuring and system tuning, automatic data distribution and redistribution, backup and recovery, integrity and security controls, etc., are generally handled by software in the traditional systems. Tremendous overhead is generated in implementing these capabilities. Because systems are currently pushing software complexity barriers, performance improvements in this area are not likely without dedicating hardware to unburden saturated systems.

III THE OBJECTIVES AND CHARACTERISTICS OF THE EXISTING DATABASE MACHINES

A database machine (DBM) can be defined as any hardware, software, and firmware complex dedicated and tailored to perform some or all of the functions of the database management portion of a computing system. The DBM may range from a small, personal query machine (intelligent terminal) to a large, public-utility information machine. We shall categorize the existing database machines into four categories based on their architectural distinctions and their

differences in objectives and characteristics. Each category of machine attempts to remove some or all of the limitations discussed in the preceding sections. In the following presentation, only the recent systems designed for general purpose database management applications are covered. Systems designed for text processing, document retrieval, sorting, etc., which are database machines in their own right, are not included.

Category 1: cellular-logic systems

A cellular-logic system consists of a linear array of cells each of which contains a processor and memory element [47]. The general architecture of cellular-logic systems is illustrated in Figure 1. A database operation such as Search, Retrieve, Update, Delete, or Insert is broadcasted simultaneously to all the processors which carry out the operation against the data residing in their associated memory elements. Thus, in one rotation of the memory, the entire database is reached in \(1/n\)th of the time needed for a sequential search over \(n\) segments of data. Efficiency in data searches and other database operations is gained by the parallel processing elements. The memory elements of these devices can be disk tracks, bubble memories, CCD’s, RAM’s or other types of memories. The cells in these devices communicate with their adjacent neighbors. This category of devices, thus, refers to a more general class of machines than the logic-per-track concept introduced by Slotnick [46].

The basic idea of cellular-logic systems is to move some of the frequent database management functions to intelligent secondary storage devices so that these functions can be carried out by the storage devices without the attention of the main processor. The data stored on the rotating devices such as disks, drums, CCD’s, or magnetic bubble memories are systematically and exhaustively searched by the processing elements, one for each physical or electronic track of the rotating memory. Thus, data are processed on the same device where they are stored. Irrelevant data can be filtered out by the secondary storage devices and only the relevant data are brought into the main memory for further processing, thus avoiding the problem of staging described in the preceding section. Furthermore, since the entire database is exhaustively searched in each circulation of the memory, data can either be searched associatively by contents (i.e., by specifying what data are to be searched rather than where the data can be found) or by contexts (i.e., by specifying the neighborhood where relevant data can be found). The content and context search techniques in the cellular-logic devices offer uniformity and fast response time for search and update operations without the need to build and maintain special supportive structures such as indexes, hash tables, pointers, etc., used in the conventional systems. Data can be stored in these machines in a form very similar to the data structure defined in the conceptual schema of a database. Thus, the difference between the conceptual schema and the internal schema of a database in these machines is not as distinct as in conventional systems. The
Figure I—Cellular-logic configuration.

complex mapping between the two data representations can often be avoided.

Four basic architectural decisions which lead to an improved packaging of the technologies for the exhaustive associative search are examined as follows:

(A) The hardware consists of a regular arrangement of identical cells. The argument for this decision is as follows. First, the development and manufacturing costs of LSI and circuit boards are minimized, since only a single generic chip need be developed and manufactured and arranged uniformly on circuit boards. Second, reliability is improved because of the overall simplicity of this approach and because several simple schemes can be used to provide dynamic recovery from hardware failures. Third, the system can easily be expanded modularly without causing disruption to the system organization. As the database grows, increased storage is accompanied by increasing processing power, so response time remains independent of database size.

(B) Instead of using higher order arrays or tree structures, a one dimensional array of cells is used. The reasons are as follows. First, a one dimensional array minimizes the number of LSI pins per cell, since communication is restricted to fewer cells. Second, the number of pins per package is independent of the number of cells per package. This is very important, since it allows us to directly exploit the drastic, yet consistent, improvement in density, without increasing the number of pins per package. No other arrangement can accomplish this. Improved lithography and circuit designs promise to make further improvements by a factor of 100 in area density by 1990. Third, hardware utilization is most easily achieved using a one dimensional array, since fewer (only one) restraint must be met. For example, a two dimensional array requires two restraints. Users of the IL-LIAC IV (Kuck [29]) have found this to be very awkward. Furthermore, variable-length data objects can easily be linearized onto a one dimensional array.

(C) Each cell has a dedicated processor and memory. The reasons for taking this approach are as follows. First, experience has shown that using \( N \) processors that can access \( M \) memories leads to severe interconnection contention, so that neither processors nor memories are well utilized. A fixed one-to-one relationship between processors and memories allows an efficient utilization of both. Second, it also removes the complex reliability and packaging problems involved in a large interconnection switch. Third, the parallelism inherent in the exhaustive search can be directly exploited. Fourth, the amount of memory per processor can be varied to allow a family of database machines to be built using the same architecture. This allows trade-offs between cost and response time to be matched to different user environments and changing technology.

(D) Block-organized memories that are serially accessible within each block can be used, such as charge-coupled devices (CCD's), magnetic bubbles, and discs. These memories are generally cheaper per bit than memories that allow random addressing at the character level. These memories will generally be classified as slow access. However, they are slow only when used to emulate a random access memory. When used for the exhaustive associative search, they are as efficient as a truly random access memory. In addition to searching efficiency, these devices offer efficient storage management for updates. Because of their dynamic nature, data can be inserted in place at the maximum data rate of the memory. Also, supportive data structures such as indexes, pointer, hash tables, etc., are eliminated and the effective cost per bit is further reduced. In summary, the block serial nature of these devices can be fully exploited to improve simplicity, efficiency, and data independence.
Several systems have been designed based on the cellular-logic approach and some have gone through prototype implementation. A few of these systems are briefly mentioned here. More details can be found in the papers included in the special issues on database machines [9, 24].

The CASSM project began at the University of Florida in 1972. The aim was to invest the hardware and software characteristics of various associative techniques. Direct hardware support of relations, hierarchies, networks, and string processing was investigated [11, 32, 48]. These hardware data types were implemented without any restrictions on length. Also, storage retrieval of instructions directly from the associative memory (associative programming) was studied. Associative programming is presently being studied at the University of Florida [50] under a continued NSF grant, with the CASSM architecture simulated in software [49].

The RAP project began at the University of Toronto in 1974. RAP [41, 44] was intended to provide direct hardware support for the normalized relational model, with restrictions on the length of tuples. The RAP project also contributed to the understanding of several systems level considerations, such as the use of RAP as a staging device for very large file systems, and system throughput under a multi-user environment. Since its initial design, RAP has gone through some substantial changes. The most recent version, which is described in Schuster et al. [45], reduces the restrictions on the length of tuples.

The RARES project began at the University of Utah in 1976. RARES [30] provided hardware support for normalized relations with length restrictions. The RARES storage structure was chosen to optimize output efficiency.

A research project called INDY began at Tektronix in 1977. INDY [10] directly implements a kernel language that is based on strings and classical sets with no hardware restrictions on length or cardinality. This kernel language acts as a meta-language that is generalized enough to directly describe various data languages and views, providing a simple closed mathematics for facilitating translations between views.

A recent project undertaken by Chang [7] at IBM, Yorktown Heights, investigates the use of magnetic bubble memories for supporting relational databases. A modular, configurable, electronically-timed magnetic bubble storage has been studied. The system follows the general concept of logic-per-track while a track in this case is a magnetic bubble chip with a modified major-minor loop organization. The proposed bubble chip configuration is shown in Figure 2. The storage minor loops are grouped to correspond to domains in a relation. The transfer line is segmented to allow the selection of a minor-loop group (i.e. a domain) to be accessed individually. The short buffer loops between the major and minor loops alleviate the problems arising from the rigid synchronization of the major and minor loops. The off-chip marker loops, being one-bit wide in contrast to being interspersed with many-bit large data records, can be quickly scanned to identify previously marked tuples. Since the minor loops allow parallel advance of data while the major loops only permit serial read-out of data, the quick scan fea-

Figure 2—Modified major-minor loop organization.
nature of the marker loop can eliminate the output of unqualified data, thus greatly enhancing performance. The project clearly demonstrates that bubble memories have several desirable characteristics which can be utilized advantageously to support database management.

In summary, the distinguishing features of the cellular-logic approach are 1) increased processing capabilities in secondary storage devices to reduce the need for data staging in the main memory, 2) search time is independent of the database size, 3) elimination of the need for building, updating, protecting auxiliary structures, 4) the use of identical cells to increase reliability, flexibility in adding or reducing the number of cells and to reduce the cost of production, and 5) the potential for extremely high speeds as cell sizes decrease and memory density and speed increase (i.e., increase in the ratio of processing power to memory). Although most of the systems described here have gone through prototype implementation and testing, performance data from a real application environment is still lacking. The existing prototypes have rather limited processing capabilities. Many of the DBMS functions will still have to be handled by a conventional computer. Also, the staging problem described in Section II will not be totally eliminated if large databases are stored on archival memories and have to be moved to cellular-logic devices.

Category 2: backend computers

Backend computers in database systems are dedicated computers for carrying out databases processing functions such as the retrieval and manipulation of databases, the verification of data access, the formulation of responses, the enforcement of integrity and security rules and constraints, etc. Backends are usually general purpose computers even though special purpose machines can very well be used. Figure 3 shows one possible configuration, the operating system, application programs, and DBMS interface run on the host computer, and the actual DBMS runs on the backend computer.

The key concept of backends is to off-load the database management functions from the host computer to dedicated processor(s) in order to 1) release the host from tedious and time-consuming operations involved in database manipulation, maintenance and control, and 2) increase system performance through functional specialization of and through parallel processing among the host and the backend(s). The primary impetus for the backend approach is, of course, to reduce the cost of managing data. The backend approach can be viewed as a cost-effective alternative to upgrading the host or to achieve the level of functionality and performance that no conventional system can provide.

The isolation of the DBMS, the mass storage devices and the database from the host can bring a number of additional advantages. First, several hosts, possibly dissimilar, can share on-line data in the configuration shown in Figure 4. A single backend may handle the processing of the database and present data in forms suitable to the dissimilar hosts. Second, databases and the DBMS itself can be transported from an old mainframe to a new one with relatively little conversion effort. Similarly, changes to the databases, the mass storage devices, and the DBMS (e.g., adopting a standard DBMS) can be made without entailing changes to the host. Third, storage devices including special purpose cellular-logic devices or bubble devices can be made available through backends to mainframes that do not otherwise support these devices because of I/O or operating system constraints. Fourth, multiple numbers of backends (see Figure 5) can be used to process large databases which can be stored either in a distributed manner across secondary memory devices to facilitate parallel processing or in a manner such
that one database can be processed by one backend. Lastly, the enforcement of database integrity and security can be separated from that of operating system integrity and security; thus the failure of one will not endanger the other.

The first development of the backend system occurred at Bell Laboratories [5]. This system was called the Experimental Data Management System (XDMS) and was undertaken to both demonstrate the capability of the backend concept as well as implement the new CODASYL DBMS specifications. The implementation required eighteen months and six man years of effort. The system was implemented to a level of experimental usefulness and the concept was verified.

The Data Computer is another example of the backend processor approach. It is a large-scale database management system running on a PDP-10 and has been implemented for use in ARPANET [36] by Computer Corp. of America. The Data Computer essentially provides facilities for data sharing of a single database among dissimilar host computers in a network environment. That is, it is implemented through a communication scheme involving the identification of the host processor type so that data to be retrieved and sent by the Datacomputer can appear in the format expected by the requesting host. Likewise data which are to be stored by the Datacomputer are converted upon receipt from the identified host and stored for use as the originator sees it. With such a scheme, the amount of storage can be continually expanded, performance can be maintained by replicating the systems, and the backend machines are available to all hosts in the network.

Some additional developments indicate the possible direction in which this movement may be heading. In the past few months, Cullinane Corporation made available to four government agencies IDMS implemented on a PDP 11/70 capable of supporting an IBM or IBM-compatible host. One participating group (within the Navy) is just now beginning a very serious evaluation of the utility of such a system in their production environments to extend the useful life of their existing computing facilities.

During the period of time Cullinane Corporation was implementing IDMS for use in a backend, Kansas State University [16,17,37], under a grant from the U.S. Army Computer Systems Command, was developing a prototype network system built around a machine independent, high-speed bus system (20 mega bytes/sec transfer rate) which would permit heterogeneous computers to communicate in any topology desired. With this communications support software finished, a natural application was the backend environment. The software design documents were furnished to Cullinane along with the host software. Addition-
ally, Cincom's DBMS system (TOTAL) was modified to run on an Interdata 8/32 backend from either the IBM host or another mini in the network acting as a host.

A great deal of database machine activity is occurring in Japan. One project defines a database machine called GDS—a generalized database subsystem—which has a sufficiently low-level interface to provide potential support for any data model [18]. One major contribution is its ability to interface directly to the main memory of its host so that I/ O overhead incurred by the host CPU during large data transfers can be avoided.

The existing backend systems are still experimental in nature. The desirability of backend is yet to be proven by performance evaluation and measurement of "real" systems. In conclusion, the idea of extending the functionality and performance of a mainframe by dedicated backends is a sound one. However, this approach does have its adverse problems. For example, the backend(s) introduces different hardware with the attendant problems of maintenance, software support, and the additional procurement effort and cost. Also, the balanced assignment of DBMS tasks to the host and the backend(s) is not a simple problem. More discussions on backends can be found in [33,42].

Category 3: integrated database machines

This category of systems uses a number of functionally specialized processors, which can be general-purpose and/or special-purpose processors, to implement the processes of a DBMS. Systems of this type may use, for example, specialized associative processors for the processing of directories and mapping data, intelligently controlled disks and mass storage devices for the storage and processing of the major portions of the database, a system processor for general coordination, and dedicated hardware for security control. By the use of the functionally specialized hardware and the parallel processing capabilities of a family of machines, these systems aim to achieve greater efficiencies in database management. The highly modular family of machines gives users the opportunities to mix and match process and storage capacity.

Different from the cellular-logic systems in category 1, this category of systems is larger and more complete systems of which a category 1 system can be a component. The specialized hardware units used in these systems are quite different. They lack the uniformity of the cells in category 1 systems. This category also differs from category 2 systems in that functionality and performance are achieved mainly by hardware (and thus software) specializations rather than software specialization alone used in the existing backends. It should be noted, however, that the distinction made would not be clear if special-purpose hardware devices were used in the backend systems. Nevertheless, we can say that the design of this category of systems involves treating hardware, software, DBMS, and databases as a whole rather than simply extending the capability of a given mainframe using backends.

Some example systems of this category are the following. The Data Base Computer (DBC) project at Ohio State University proposes an architecture where every major DBMS function has a dedicated processor and whose overall organization exploits pipe-line parallelism [1,3,20,21,22,23]. It contains various associative processors for logical data model and disk memory mapping. It also proposes several architectural changes to moving head disks to increase bandwidth an order of magnitude over today's secondary storage data rates. The integration of the security function into the DBC's architecture is also considered.

The RAP.2 effort at the University of Toronto has expanded its research by formulating the RAP (a category 1 machine by itself) associative processor's role in an integrated database machine. Most of the work has centered around data partitioning or staging strategies where database and schema data reside partially on disk and partially on associative processors [45].

The INFOPLEX system proposed at MIT is an example of integrated database machine architecture [35]. It utilizes new microprocessor capabilities by organizing a memory and processor hierarchy which takes advantage of the parallelism inherent in concurrent requests to maximize performance.

Another direction is to make use of low cost currently available microprocessors to form a simple network system for processing distributed databases using a single-instruction multiple-data stream architecture (SIMD). In this case, segments of data files are stored across memory devices each of which is dedicated to a microprocessor. Software tasks for a database management system are simultaneously carried out by the processors against the contents of the local memory. This alleviates much of the switching time overhead found in a network systems with shared memory. A recent example of this approach is the MICRONET system being developed at the University of Florida [51] using a PDP 11-60 and four LSI-11 computers.

Another multiprocessor system called DIRECT [15] is designed for supporting relational database management systems using a multiple-instruction multiple-data stream (MIMD) architecture. Microprocessors are dynamically assigned to a query depending on its priority and the type of relational algebraic operators it contains and the size of relations referenced. The system is being implemented using LSI-11/03 microprocessors and CCD memories which have associative search capability.

In summary, the main characteristics of this category of database machines are 1) the use of functionally specialized hardware to achieve efficiency, 2) the system's approach to the design of hardware, software, DBMS, and databases, and 3) the modular family of machines allows users to exploit parallel processing and pipelining techniques. However, the hardware interconnection, the data and program communication, and the operating system support in a system using dissimilar hardwares can be rather complex. The proper identification of DBMS functions for implementation in hardware remains a challenge.

Category 4: high-speed associative memory systems

In this category of machines, a high-speed associative memory is used together with conventional memory devices.
The principal benefit of this approach is improved performance. The use of high-speed associative memory reduces the effective access time of the mass memory where databases are stored. However, due to the high cost of building this type of memory and processor, the size of associative memory is rather small. In a database management environment, considerable amounts of data will have to be paged in and out of the associative memory to take advantage of its capability. Although data can be searched in high speed once the data are in the memory, to stage data into the memory can become a bottleneck of this type of system. For certain types of applications such as table look-up and directory processing, the use of high-speed associative memory will result in an order of magnitude improvement in performance at relatively low incremental cost. Where there is little locality of references, however, the potential cost benefit will not be realized.

IV. DATABASE MACHINES AND SOME ISSUES ON DBMS ARCHITECTURE, DATA MODEL AND DATA LANGUAGE DESIGNS RELATED TO DBMS STANDARDS

Having described the motivation, objectives, functionalities, and challenges of the existing database machines, we shall now look into some of the issues on DBMS architecture, data model, and data language design from the viewpoint of database machines. Many issues discussed here have often been raised by researchers and practitioners. They are very relevant to the standardization of DBMS architectures, data models, and data languages.

A. DBMS architecture issues

DBM's support of multi-schema architectures

The DBM technology could conceivably make those DBMS architectures which involve multiple numbers of schemas (e.g. the ANSI/SPARC architecture) very cost-effective. That is, it could have performance features that reduce the cost and complexity of the various schema mappings. The commitment to separate user views, logical data structure, and physical data structure stands on its own right. It is not compromised by the fact that we are limited to von Neumann processors, disk, tapes, etc., today and it should not be compromised by what happens tomorrow, particularly since we can make the separation increasingly economical through DBM technology. With respect to the standardization of the DBMS architecture, it cannot be stated categorically that DBM technology as such is going to push us toward a particular conceptual data model and external data models. Rather the DBM will probably support whatever is wanted as the "best" conceptual data model (by whatever criteria) and its mappings to external models and the mapping to the internal data model (including the internal data model of the DBM itself, the distribution among various mass storage devices, and distribution among geographically separated database systems). The internal data model is
probably not "standardizable," because, first, it does not need to be. Programs and end users do not see it or depend on it. Secondly, it must adapt to changing storage technologies including the DBM, storage hierarchies, geographically distributed databases, etc. Therefore it is important to separate the internal schema from the conceptual schema and keep it flexible and extensible.

Mappings between external and conceptual schemas

The mapping between external and conceptual schemas may involve a subset mapping and a restructuring mapping. Subset mappings are necessary to provide privacy from unwanted queries, security from unwanted updating, and user convenience by removing all data that are not of concern to the user. Restructuring mappings are necessary to provide data structures that are convenient for user applications, and to provide support of multiple user models and languages.

A DBM can play an important role in implementing these mappings with efficiency and simplicity. It is possible to store and manipulate these schema descriptions on database machines as simply another database, where mappings are accomplished using queries to the schema descriptions. However, to do this, database machines must be capable of a more generalized pattern matching capability for strings and sets. This is necessary since these schema descriptions usually involve searching abstract or axiomatic (e.g., set theoretic or predicate calculus) representations, rather than simply searching actual data instances. Ideally, the same hardware would be used for actual data and for both external and conceptual schema descriptions.

Mappings between conceptual and internal schemas

Some database machines can allow the storage structure of a database as defined by the internal model to be very similar to the structure defined in the conceptual model, and thus simplify the mapping process. For example, a relation in the community view can be stored and searched in an associative memory without index tables, hash tables, pointer arrays, etc., commonly introduced in conventional systems. This means that any data stored on these machines requires only the simplest of mappings to its internal schema. However, this does not necessarily mean that the internal schema of the entire database system will be simpler. In a large database system, an associative memory would probably be one out of a whole hierarchy of memory devices, each featuring its own tradeoff between cost per bit and response time. If the associative memory is used and managed as yet another component in a large system, it could add some complexity to the overall internal schema. Instead, the architecture of the entire database system should be reexamined with database machines in mind. Its unique qualities can be exploited to simplify the overall system. The unique features of associative machines are fast response times and simple mapping between the conceptual and the internal schemas but with a higher cost per bit than mass storage devices. The following three systems functions seem appropriate for associative machines.

One function is the direct storage of databases whose requirements for speed warrant a higher cost per bit. A second function is to manage the mappings between the conceptual and internal schemas for databases stored on mass storage devices or for geographically distributed databases. The distribution of data among various mass storage devices or among geographically separated systems can be described and stored directly in associative machines as simply another database. Schema mappings can be implemented using queries from the internal and conceptual schema descriptions. Associative machines offer the potential for storage and querying of abstract representations. An internal schema that uses abstract representations, rather than involving actual data instances, has the potential advantages of a more compact description and one that requires no updating when updates are made to the actual data.

A third function of associative machines is to act as a staging device for large blocks of mass storage. Most mass storage devices are accessed by location. Efficient use of these devices usually requires clustering of data into many large physical blocks, which is biased to certain access paths. After queries to the internal schema (directories) have reduced the number of blocks involved in a retrieval to a small number, associative machines can then be used to further search these blocks.

B. Data model issues

Database machines support of data models

A DBM can be implemented to support any existing data model. For example, RAP, RARES, and DIRECT were designed specifically to support the relational model. The CASSM and INDY systems can support hierarchies as well as a subset of relational algebra operators and string pattern searches. The ASP system was designed to support a form of the network model. Although it was not compatible with the DBTG model, such an implementation should not present any major problems. Finally, any general purpose back-end computer can be programmed to support any or all of the models simultaneously.

The implementation in hardware of a single model does not preclude it being used to support other models. For example, a system that directly supports relations can be used to simulate hierarchical and network models. They can be implemented by setting aside items called "associative links" or "context pointers," in record occurrences (tuples) to store identification and structural data.

Implementing hierarchies and networks requires the ability to implement "functional associations" between occurrences of record-types [52]. A record-type is analogous to a relation. A functional association can be defined as a 1:N (i.e., a one-to-many) linkage or mapping between record occurrences of two relations. That is, if a 1:N linkage exists between relations A and B, then one record occurrence of A can be associated or linked with zero or more unique records of B. Each B record will have at most one A record for a particular association. An association or link is equivalent to a "set" in DBTG terminology. Restrictions on the ap-
Application of functional associations between record-types determine if the database schema is hierarchical or network.

One way to implement an association is to allocate an item called ASSOC in the relation that acts as the domain of the functional association. This scheme is shown in Figure 7. The item ASSOC acts as the associative link. Each record occurrence must have one item whose value uniquely identifies the relation and each particular occurrence within the relation. This item will be called ID for identification. For each record of B that is associated with one record of A, the record ID value of A is stored in the ASSOC item of B. Finding records of B associated with a particular record of A or vice versa is simply a matter of using the associative cross selection or join instructions which interrelate two relations through comparable ID and ASSOC values.

A second way to associate records of the same or of different types is to create a new linking relation which contains two (or more) ID items—one for each record-type. This relation, called LINK, associates one record of A to one record of B by storing the associated ID's of the two records in one occurrence of LINK. This scheme has the advantage of implementing M:N, or many to many, associations between record-types. An example is shown in Figure 8.

It should be noted that only "information carrying" associations need be implemented with links. All other relationships which can be derived directly from the values in the records can be handled directly through associative cross selection or join instructions of relational DBM's.

Of the three data models, the relational model is the most general in terms of the types of associations it can represent. It also requires the least number of basic or primitive operations to implement a relationally complete data manipulation instruction set. Also, its simplistic record structure and orientation to sets-of-records operations makes it a natural candidate for DBM implementation.

From the above comments, it may appear that the relational model may be the easiest to implement and result in the best performance. However, we must be careful about jumping to conclusions. Many of the additional features of hierarchical and network models were proposed because of the need to improve transaction processing performance. The same techniques that have served software implementations will likely serve hardware as well. Also, the users application may better lend itself to hierarchical or network modeling. In such cases, hierarchical or network hardware will probably out-perform relation hardware using software and data to simulate other models' primitives. Also, many transaction applications do not require complex search nor are the sets of records to be processed large. In fact, today's online transaction processing applications are dominated by having a large number of concurrent transactions requiring relatively simple search and update interactions. These types of operations are the least likely to take advantage of the set-oriented associative processing capabilities of relational or set theoretic DBM's. Of course, a major reason why existing computerized database applications predominately require simple searches and updates is that an adequate implementation of more complex models is not available.
Judging from existing examples, the DBM will very likely make the more advanced conceptual data models (e.g. relational or set-theoretic) more feasible to implement, where as today they are frequently judged very complex to implement efficiently as a general purpose system for a broad base of applications. Thus we should be able to choose a standard model based on user benefits and assume with confidence that the performance gap will gradually close.

C. Data language issues

We now turn to data languages, collectively consisting of all languages for directly manipulating database data on behalf of application programs or end-users. Thus data languages include data sublanguages, which are extensions to conventional programming languages, and self-contained languages (such as query languages, report generators, “query by example” and other end-user interfaces). Data sublanguages in particular are the target of standards efforts because of the need to protect the user community’s investment in computer programs that use these interfaces. Any practical standard takes into consideration user requirements: e.g., proper functionality and ease of use, and feasibility—is there a reasonably efficient, economical implementation of the proposed interface? The feasibility condition creates tension in times of rapid technological innovation, when ground rules for judging what is possible or economical are subject to radical change. This appears to be the case for data languages, not only because of DBM development, but also in view of the slow but steady trend toward hierarchies of storage and geographically distributed data processing. The following paragraphs tell this story: The bad news is that the ability to improve price/performance through technology is very sensitive to the character of the data language. The good news is that we can predict well in advance what features data languages must have to fully exploit emerging technology. Furthermore there is a strong indication that these same features are desired by the user community independent of technology considerations. If so, then the standards makers have their work cut out for them.

High level vs. navigational data languages

As will be seen, the underlying technical considerations generally motivate the development of very high level data languages, by which we mean languages in which the user/programmer expresses to the database system what results are expected instead of, or in addition to, how the results are to be obtained. With regard to high level data languages it must be recognized that:

- "high level" and "low level," like "procedural" and "non-procedural," are relative terms;
- self-contained languages are not the only languages that can be high level or non-procedural. There is no intrinsic reason why a data sublanguage cannot be high level even if the programming language in which it is embedded is low level. See, for example, the use of ALPHA in [8].

Whereas everyone appears to agree that end-user oriented languages should be high level, there is an ongoing controversy concerning whether high level data sublanguages are desirable. On the one side are those who argue that programmers should have relatively low level facilities so that they can fine-tune performance tradeoffs. The other side contends that in an era of increasing programming costs and decreasing hardware costs it is best to optimize programmer productivity through the use of high level facilities and let the system worry about efficient hardware utilization. Technology trends and the DBM in particular strongly support the latter position. We will briefly examine some reasons for this.

A database machine can sometimes be “tightly coupled” to the hardware which makes use of it. For instance a mainframe manufacturer could develop a backend which is enclosed within the host itself and communicates with main memory through a very high speed bus. Or a multifunction terminal might be plugged directly into a small “query machine.” In such cases there is no concern that communication with the DBM will be a performance bottleneck. But suppose that the DBM is not developed by the host manufacturer, or that is designed to serve multiple hosts. Or suppose that a DBM is required to communicate with remote hosts in a network, or even with other DBMs to support a disturbed data base (Figure 9). The need to do all of these is bound to arise, so the DBM developer must evaluate the response and throughput implications of loosely coupling the DBM to an external I/O interface or an even slower telecommunications channel. There is nothing in the ten-
year picture to suggest that the price/performance penalty for loose-coupling will go away (otherwise the economic argument for distributed data processing would lose most of its force).

The DBM developer is therefore motivated to minimize the amount of data that must go in or out of the DBM in order to get a user's job done. He must also strive to minimize the number of separate messages, large or small, to reduce the communication burden. All of this has a direct bearing on the data language available to the user. In the extreme case, if the user can express his job in a single data language statement, and if that statement can be directly interpreted by a DBM, then obviously the communication overhead has been reduced as much as is possible. If, in contrast, the job must be decomposed by the user into a program with several lower level sublanguage statements, possibly executed in a loop, then the number of messages and amount of data transferred will increase dramatically. For example, suppose the user is to mark "inactive" all posted accounts for which there have been no debits or credits during the last twelve months. Given a powerful data language capable of dealing with entire sets of data, this transaction can be expressed with a single statement—a single "call" to the database system and no database records transferred. Given a record-at-a-time ("navigational") data language, there would be at least two calls to the system for each inactive account, one to retrieve the record and the other to store the modified version.

There are halfway measures which preserve the navigational nature of low level data languages, but would still reduce some of the DBM interaction. For instance, high level intention declarations are a possibility (Lowenthal [34]). If, in the above example the user could state in some fashion, "I intend to update all accounts for which there have been no debits or credits posted during the last twelve months," then the system could subsequently buffer blocks of multiple records between the host and DBM, but move one record at a time to or from the user's program. This wouldn't reduce the amount of database data transferred, but it would cut down the number of messages between the host and DBM (each message would be longer). This technique is useful when sequential treatment of data is ultimately unavoidable by any means, such as if a program is required to produce a list of the accounts that have been marked inactive.

Consider another method of capturing the high level meaning of an operation expressed in a low level data language. Suppose that the results to be obtained are such that the programmer can write a special kind of subroutine in which the only data referred to are the parameters, the database data retrieved in the subroutine, and some constants established in the sub-routine. This subroutine does not refer to global (common) data, does not read or write non-database files, and does not call other subroutines. Given such constraints, it is feasible to transfer the entire subroutine to the DBM as a single operation, either in source or object form. The DBM can perform internal retrievals, returning only the subroutine's output to the host. Using the above example, a subroutine $X$ would be catalogued (in the DBM) which retrieves each qualifying account and stores it back with the "inactive" indicator set. The only interaction between the host and the DBM is the command to execute $X$ and status returned upon completion. An additional benefit of this approach is the opportunity for the DBM to optimize the execution of $X$ since it "sees" the entire collection of database operations instead of individual data language statements. CASSM is an example of a DBM which supports catalogued subroutines.

There are several data language features that could be included if the aim were to minimize communication. Most of these motivate or force the user to express at a high level what is to be ultimately accomplished. They cause the language to be less procedural, or supplement procedural sequences with non-procedural declarations.

We point out in passing that the cost of inter-task communication in a typical mainframe operating system is surprisingly high, so that even in a conventional software database environment there is a strong motivation to reduce the traffic between the application task and the database task. Another independent motive is fueled by the advent of hierarchies of storage, which are inevitable if very large databases are to be addressed in the context of foreseeable price/performance trends for different types of secondary storage; no single device is expected to emerge both cheaper and faster than any other device (see Figure 10). It has been argued that storage hierarchies will be more effective if the data staging algorithm can anticipate in advance exactly what data will be required [34]. This again relies on a language through which the user can express with some refinement his data needs. High level, set-oriented statements, intention declarations and the like would all marry quite well with an intelligent data staging mechanism. Thus it is the broad direction of computer technology encompassing distributed processing, storage hierarchies, and software engineering, and not just the DBM which calls for a reassessment of data language standards efforts.

**Set oriented vs. record oriented processing**

The DBM concept most directly and vividly exposes the relationship between a data language and the hardware mechanism which ultimately does the work. In previous sections it has been established that conventional computer architecture is not particularly well suited for database management, that dramatic improvements in cost/performance can be achieved with fundamentally new approaches. In nearly every proposed architecture, be it oriented to searching, sorting, list merging or the like, there is a common theme: one or more sets of data are operated upon to produce another set. This is no accident since the basis for the claimed economy is parallel processing, that is, many small inexpensive processors working effectively together to do a large job quickly. The opportunity to exploit parallelism practically depends on the ability to define operations in terms of sets instead of individual points of data. This in turn clearly depends on the ability to deal with sets of data at the level of the data languages itself.

In the world of scientific computing, scalar oriented lan-
guages like FORTRAN have been enhanced with high level array operations so that, for example, matrix inversion or multiplication can be expressed as a single statement. This enhancement is motivated not so much by software engineering principles as the industry's ability to build highly parallel machines that operate on arrays at blinding speeds. If matrix multiplication can only be expressed as a sequence of DO, IF and assignment statements, how can the underlying system figure out what the programmer intended? How can the advanced architecture be exploited? Likewise if a database programmer cannot express a predicate as a predicate ("find all accounts for which no credits or debits have been posted during the last 12 months" paraphrased as a single data language statement), but must restate it procedurally with more primitive record oriented statements embedded in loops, how can set oriented DBM's like RAP, RARES, or CASSM be effectively exploited?

In the past, set oriented data languages, sometimes (incorrectly) called "relational" languages, have been regarded as powerful but impractical—too expensive to implement and operate. The lower level record oriented languages, including the CODASYL DML, have scored high points for feasibility and economy. The emergence of DBM technology may actually reverse this situation in the next few years. In view of this, language developers working with the CODASYL basis should work out ways of enhancing the DML with set oriented operations. Not only will this result in a better fit with the DBM, but also with the trends in user requirements (people productivity), mass storage technology and distributed databases.

There is an obvious counterargument. If users rarely need to manipulate matrices, then fancy scientific computers should be built for the few and FORTRAN for the masses shouldn't be affected. Likewise if very few users need to manipulate sets of data, but rely mainly on sequential access or simple direct access ("find the unique account record with key account number 745286"), then set oriented machines will not have broad appeal. We strongly believe that although there will always be a need for record oriented access to data, there is also a great demand for set oriented capabilities. Moreover this demand can only increase as databases come to be regarded as information resources for management.

V. SOME TECHNICAL ISSUES ON DATABASE MACHINES

The following is a collection of key technical issues which must be addressed by researchers in database machine technology. The discussion is broadly grouped into three areas: basic technology, hardware architecture, and software architecture.

Basic technology

The use of the systems described in Section III will depend heavily on cost, performance, storage capacity, and reliability of such solid-state devices as LSI processors, RAMs, CCDs, and bubbles. DBM architects will be structuring systems which incorporate such large volumes of these devices that reliability will dominate the design of products. Researchers are only beginning to realize that solid
state devices are not just “electronic” disks. Bubbles and CCDs provide unique opportunities for combining logic with storage as demonstrated in IBM’s bubble query machine, RAP.2, etc. The main manufacturing problems for research and development are:

1) High density storage media

Texas Instruments introduced in 1976 the TIB 0101 bubble chips with 10^6 bits/chip at 10^8 bits/in^2 density (6 µm bubble diameter), and in 1978 the TIB 0103 bubble chips with 2.56 × 10^5 bits/chip at 4 × 10^6 bits/in^2 density (3 µm bubble diameter). A simple board of 4″ × 4″ in area containing 1MB bubble memory module as well as all semiconductor components has already appeared [25]. Research work on 1 µm and even 0.5 µm bubble diameter materials (potentially up to 10^9 bits/in^2 density) have been reported by IBM Research. The manufacturers must get ready to build devices using such materials. Investigators will continue their search for materials sustaining even smaller bubbles. Alternatively, the engineers may invent and implement device structures capable of higher densities (e.g. bubble lattices) than conventional structures (e.g. half disk types used in TIB 0303) at the same bubble diameter.

Similar advances in design are taking place in LSI semiconductor devices. One example is TI's three-dimensional MOS RAM cell design in 1978 that reduces area, power, and refresh requirements. Also, several new semiconductor materials are being discovered, such as Galium Arsenide, that reduce area and power requirements.

2) High resolution lithography

Bubble chips entered the market using high-resolution photolithography (in fact, close to the limit of its capability). Electron beam lithography will reduce the line width by at least another order of magnitude. When used with small bubble materials or various semiconductor devices, it will enable bit density increase by two orders of magnitude. Again, clever device structure (e.g. contiguous disks or three dimensional MOS devices) achieves higher device density at a given lithography capability, thus providing an alternative to high-resolution lithography.

3) Packaging

Packaging considerations can have a large impact on cost, speed, and reliability. Cost, speed, and reliability have and will continue to be substantially improved by putting more devices on a chip. Improvements in device design, better yields to allow larger chips, and higher resolution lithography are increasing the number of devices on a chip at such a drastic rate that it is difficult to comprehend. However, to exploit this requires equally drastic architectural approaches to insure that the number of LSI is minimized. The simple-minded approach of integrating more of the conventional architectures on a chip usually increases the number of pins per chip beyond cost-effective technological limits (currently about 40 pins per chip). Two approaches can be taken to improve the situation. One approach is to reduce the cost of more pins per chip. Another approach is to reduce the number of pins per chip using a different architectural approach.

Many improvements have been made or proposed to reduce the cost of more pins per chip. Gang bonding and film carrier techniques allow more of the packaging of chips to be automated with improved reliability. Also, putting multiple chips on a single substrate can reduce the cost of packaging. Another technique called wafer-scale integration (WSI) can potentially avoid much of the packaging costs by interconnecting the chips directly on the original wafer. Bad chips are removed using laser trimming or using dynamic diagnostic algorithms to locate and electronically disconnect bad chips. The dynamic approach has the advantage that it can be applied to remove chips that have gone bad in installed equipment.

Alternatively, new architectures can cluster hardware onto chips in ways that reduce the number of pins per chip as well as simplifying the interconnection among chips. The cellular-logic devices described in Section III use a one-dimensional array, a tree, or a network. A one-dimensional array requires the fewest pins per cell because each cell need only communicate to its two adjacent cells. Also, the number of pins per chip is independent of the number of cells per chip. This allows the drastic increase in devices per chip to be directly exploited without increasing the number of pins per chip. For example, if one cell per chip requires 16 pins, then 100 cells per chip would require only 16 pins. This advantage also carries over to larger packages, such as printed circuit boards, multiple chip package, and wafer-scale integration. No other topology has this property. All others must increase the number of pins per chip as more cells are integrated into one chip. In order to exploit this advantage, however, the memory and processor of each cell must be compatible technologies, so that they can be packaged (or preferably processed) together. Various semiconductor memory technologies have very compatible logic technologies. Also, magnetic bubble logic shows great promise for exploiting bubble memories. Disc and tape memories, however, have no compatible logic technologies.

The industry has already paid attention to board compatibility and voltage compatibility of bubble components with semiconductor components. Some remaining problems for bubbles with major improvement potentials are multiple-chip packaging, replacement of external bias magnets by on-chip bias, replacement or simplification of the external driving coils, and further development of bubble logic.

4) System innovation

The hardware problems are reasonably well defined and being pursued. The system problems are desperately in need of innovation, discipline, and interaction with hardware know-how. There have been enough scattered conceptual explorations of bubble device capabilities (e.g., a variety of device structures for Boolean logic, text editing, data man-
agement, sorting, associative search, etc.). Evaluation of the feasibility of these devices is lacking. No serious commercial impact is foreseen without the development of a few (indeed very few) basic chip types encompassing a collection of universal functions. System assessments are equally lacking. Detailed designs to include system performance evaluation and software requirements are needed to demonstrate the advantages of the innovative hardware designs. As usual, a multi-disciplinary area tends to become a no-man's land. Only simply problems such as simulation and performance evaluation of bubbles and CCD's as gap fillers have been examined, probably over-worked.

Tomorrow's DBM's will depend heavily on both loosely and tightly coupled inter-processor architectures. Communication considerations will begin to dominate price and performance. Realization of DBM architecture will depend heavily on progress in this area.

The design of special purpose LSI devices to fit DBM idiosyncrasies will depend heavily on cutting design and engineering costs for such devices. If costs continue to run high, the DBM implementors will have to structure their thinking toward utilization of more conventionally organized memory and microprocessor components.

5) Technology and standardization

Standardization usually comes after developments in products have been done, not before. However in the age of very large scale integration (VLSI), when design cost overshadows manufacturing cost (e.g., see Moore [39]), it would make great sense for the users to indicate what they want to see in the hardware. By adjusting their requirements to the manufacturing constraints of hardwares, they may forecast the standards before the product development, both for user convenience and for manufacturing cost reduction.

Let us clarify the issues by considering a specific technology—magnetic bubbles. At present, bubble memory modules with capacity ranging from 92kb to 1Mb are available commercially. Certainly, the technology is mature enough to consider standardization issues. In the U.S.A., bubble products are marketed by Texas Instruments, INTEL, Rockwell International, and National Semiconductors, and also produced by Western Electric and other companies for internal use. In Japan, Fujitsu, Hitachi, and NEC are manufacturing bubble modules as commercial products (see Yamagishi [53]). Certainly, there are enough manufacturers to make standardization issues relevant and urgent from the user's viewpoint. Moreover, steady improvements of device density and chip capacity have been predicted, and various functional enhancements have been proposed. Certainly, the technology will undergo highly dynamic evolutionary stages and need standardization to prevent unbridled developments.

The maturity of manufacturing technology will encourage the pursuit of associative search, sorting, data management, simple Boolean logic, etc. (see Chang [63]). Although the detailed device configurations must await the gradual hardware evolution, the terminal characteristics of the chips of concern to the users could be responsive to the users, and early interactions between the manufacturers (or their fore-runners—the researchers and developers) and the users will be worthwhile. Some proposals for standardization may be a reasonable way to initiate the dialogues.

Hardware architecture

1) Clearly, the proper mix of families of device architectures and speeds will be a major concern of DBM technologists in the '80's. Because of the expense of prototyping such systems, there will be a heavy reliance on modeling and performance evaluation simulations.

2) The need to define logical interfaces and protocols for I/O architectures will become a dominant theme in the '80's [38]. This will be required so that the systems can more easily incorporate various DBM components into integrated systems to meet user application needs. One can anticipate the same controversies to arise in this area as have occurred in communication and networking standardization efforts.

3) The success of category 1 and category 4 DBMs will depend heavily on being able to optimize their usage in broad application environments. For example, they appear to be most cost effective where searching requires complex relationships be satisfied on secondary keys and when multiple records respond to such requests. This feature is expected to become more important in the future when applications are hypothesized to rely heavily on on-line queries. Nevertheless, these devices will have greater applicability if they can also efficiently search for single records. The ability to handle many data types of varying lengths would also broaden their market.

4) The protection mechanism required by databases to control concurrency, security, integrity, and recovery have barely been considered by workers in DBM technology. This is often passed off as a software problem. A fruitful area for DBM researchers will be in designing DBM architecture to support these functions. The inherent speed of associative processors indicates that enforcement of protection rules may become one of their primary functions.

Software architecture

1) Because database machines will incorporate many diverse processors, bulk memories, and intelligent memories with varying price, performance, and capacity, an extensive amount of work will continue to be needed in studying data clustering, partitioning, staging, and virtual memory strategies for files. Magnetic disks are not likely to disappear in the '80's. Also, other low price/bit large file technologies may come of age in the '80's, e.g., laser video disks and EBAM. They will be used to store the majority of on-line data. Accessing strategies will continue to optimize resources by attempting to minimize the number of disk accesses required to complete an operation. Algorithms that use intelligent controllers and associative memories will be sought to improve access for these bulk memories.

2) An important contribution that is needed to unify database machine research will be the identification of com-

From the collection of the Computer History Museum (www.computerhistory.org)
monality and compromise between the individual requirements of text, formatted files, signal, graphic, and map databases.

3) An important issue raised in the past is whether or not database machines should be user programmable. That is, should software be provided to allow users to code data processing and systems programs or should the system limit itself to the execution of database management functions. Precluding the ability to run machine or compiled code will eliminate many of the mechanisms or avenues that allow database security and integrity breaches today. It will also increase the designer’s degree of freedom in customizing the DBM for its intended function.

4) The collection and dissemination of user statistics relating to query complexity, file characteristics, locality of database access, etc., are currently non-existent. Without this data, researchers can only hypothesize the relative importance of various architectural tradeoffs. We cannot deliver good solutions until the problems are well understood and parameterized. On the other hand, we cannot parameterize user statistics until we deliver good solutions. Users adapt to whatever system is available. Any statistics gathered from existing systems is only valid pastbound and may not have any resemblance to the future. Improvement must be made iteratively. Because of improvements in hardware, new and improved system strategies will be developed and used. This will, in turn, provide feedback to aid in further hardware improvements.

VI. CONCLUSION

What impact do hardware technologies and database machines have on the database management area? The answer is: They are all making data processing less expensive and more accessible (to both large and small users). The low-cost, computational, logic and control capabilities have already made microprocessors ubiquitous. Bubbles and CCD’s offer modular storage coupled with data storing, arranging and managing capabilities. Their impact will be twofold: First, they will extend database management capabilities to smaller data collections for smaller users in smaller machines. Second, they will be useful in large database systems as nodes in a network, as servers, and as components amenable to parallel operations.

Advances in database machine technology will be required to solve many database management system problems so that the promise of the database gospel can be delivered to users. Progress toward producing these machines will depend heavily on the improvements in price/performance of basic memory and processor technologies. A better understanding of the partitioning of the total problem will also aid special device development. The trend will be toward defining integrated database machines. Thus, workers in this area will find it necessary to have a good understanding of database application and software issues, as well as hardware architecture and technology issues.

The advances in DBM technology will not only have great impact on the implementation of DBMS software but also have profound effect on the designs of DBMS architectures, data models, and data languages. Database machines can make it very cost-effective to support high-level data models and data languages which are necessary for improving user/programmer productivity and to support multi-schema DBMS architectures which are necessary for achieving data independence. The existing database machines have demonstrated their capabilities to make data mapping between schemas a simpler task and to support the existing data models with considerable improvement in cost/performance. Furthermore, database machines are particularly suitable for supporting high level, non-procedural, and set oriented data languages. Thus, we should establish a standard DBM architecture or a data model based on user benefits and assume with confidence that the performance gap will gradually close up. High level, non-procedural and set oriented operations which score high in both user productivity and technology considerations should be incorporated in a standard data language.

REFERENCES