Database Research Activities at the University of Wisconsin

Haran Boral, David J. DeWitt, Randy H. Katz, Anthony Klug
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706

1. Introduction

In this short note, we describe our current research activities in database management systems. The department's interest in database management systems has been steadily growing over the years, and now encompasses work in database machines (Boral, DeWitt), design data management (Katz), user interface and data models (Klug), and theoretical foundations (Klug). In the following sections, we describe the research projects underway at the University of Wisconsin.

We have recently perceived that we need a locally written and maintained low-level database system to support our research activities. The group of us, with some students, have undertaken its design and implementation. The system, the Wisconsin Storage System (WiSS), will permit us to experiment with building special purpose database systems.

2. Database Machines (Boral, DeWitt)

2.1. DIRECT Implementation

During the past several years we have been engaged in the design and implementation of DIRECT -- a relational database machine. The current DIRECT configuration consists of eight LSI

11/23 processors, a 1/2 Mbyte multiport CCD memory that acts as a distributed cache, a PDP 11/40 that acts as a controller, a 1 Mbit broadcast ring used for exchanging control information among the processors, and mass storage. The front-end is a skeleton of INGRES. At this time all the relational operations, as defined by INGRES, are supported except aggregate operations which are being implemented.

2.2. Parallel Algorithms for Relational Operators

To design "faster" database machines, we have recently designed and analyzed a variety of external parallel algorithms for sorting, projection, join, update, and aggregate (both scalar aggregate and aggregate function) operations. The analysis methodology utilized incorporates I/O, CPU, and interprocessor message costs. Our work is now directed towards more accurately measuring the effect of control and I/O on the performance of these parallel algorithms.

2.3. Management of Statistical Data

Statistical databases are typically very large and complicated in structure. Existing software to manage statistical databases is generally written by people who are not database experts, and is therefore not of very high quality. We are interested in developing a data management system for statistical data. This system will include both software and hardware components. These will be used for performance improvements as well as intelligent management of the data.

2.4. FLASH - A Dataflow Database Machine

We have recently completed the preliminary design of a database machine (named FLASH) that uses data-flow query processing techniques. This machine supports decentralized control of instruction execution, uses broadcasting as the basic building block for all the complex relational algebra algorithms, and combines "on-the-disk" processing of selection operations with "off-the-disk" processing of complex relational operations. We are presently beginning to evaluate the performance of FLASH relative to DIRECT and other database machines. When its design is finalized, we intend to use CRYSTAL (a 100 node multicomputer funded by the NSF experimental computer science research program) to develop a prototype.

3. Design Data Management (Katz)

We are applying database management techniques to the problem of managing the information about VLSI circuit designs. What makes VLSI design data particularly interesting is the need to 1) support a hierarchical design methodology, 2) support multiple design representations of design objects, and 3) efficiently manage large volumes of design data. Although relational database techniques have proven to be well-suited to managing large volumes of highly regular data, they do not completely solve the problems of design data management. A more sophisticated approach is needed, although it should still be founded on the concept of an integrated database for design data. We are

interested in developing new methods and techniques for the efficient management of data about a VLSI design.

Our system consists of three distinct components. At the core is the database component, whose responsibility is to provide a reliable, efficient interface to stored data, and to isolate higher level software from considerations of reliability, physical storage structure, data sharing, access control, and integrity control. The database component must provide the ability to store unstructured data, such as text and graphical images, while providing a simple interface upon which the rest of the system is built.

The design management component is built on top of the above. It is responsible for mapping a design hierarchy of leaf and composition cells, as well as the many representations and versions of the design, onto the data structures supported by the database component. It provides an interface for extracting and replacing subcomponents of a design from the design hierarchy.

The final layer of software consists of programs that interpret the data about a design. Design interpreters provide an interface to design tools that is tailored for manipulating the data at hand, e.g., layout geometries, transistors, etc.

The primary advantage of our system is that it facilitates the rapid construction of new design tools by relieving the tools designer of the burden of data management. Once a design interpreter for a particular type of data representation has been

written, it can be used by the many tools that manipulate that representation.

There are many research issues that we are currently investigating. First, how can we store unstructured data in a database system? Such data types are common in the CAD environment. Second, how does a "database snapshot," i.e., a read-only version of a database at a particular point in time, relate to the concept of a design version? Can existing techniques for implementing database snapshots be adapted to support design versions? Third, are new methods of concurrency control needed to support the long duration, data intensive transactions (Lorie's "conversational transactions") typically found in CAD applications? Can the design hierarchy, when explicitly supported by the database system, be exploited to control concurrent access to independent parts of the design? By independent, we mean that the design components reside in parallel subtrees of the design hierarchy. Hierarchical locking protocols may be applicable here. Fourth, can the consistency of the design be maintained by automatic techniques for propagating changes to dependent design components, either across design representations or through the design hierarchy? Fifth, what techniques, such as Stonebraker's hypothetical databases, are needed to support alternative designs? Finally, how can the architecture outlined above be adapted to a network of database server machines and designer workstations? In particular, how should the design data management system be partitioned across these two types of machines?

These problems are under study, and we are beginning to implement a prototype system for design data management.

4. User Interface and Data Models (Klug)

4.1. Statistical Queries

We have made natural extensions to relational calculus and relational algebra for supporting statistical operations. A relational language called "Abe" (for "aggregates-by-example") has been designed based on relational calculus has been designed and is being implemented. Abe is similar in several ways with Zloof's Query-by-Example (QBE), although it is more powerful than QBE in many respects, and its definition is actually simpler than QBE's. Abe achieves its power, not through a maze of ideas such as grouping, partitions, partition selectors, and duplicates, but through the simple idea of a subquery with parameters. Abe queries are tree structured, and at one level they resemble the tableaux of Aho et al. and the tabular structure of QBE. Abe syntax and semantics are easy to understand, and the entire specification can therefore be made visible to the user.

4.2. Multiple Data Models

We are implementing a DBMS, called System DM*, which supports the following data models: hierarchical, network, relational, role, functional, binary relational, and entity/relationship. Very general views are available from each of these data models. The questions being considered include:

(1) What small set of conceptual schema constructs will enable the largest set of user data models to be supported?

(2) How serious are the inefficiencies that are introduced by having so many software levels between the user and the data?

(3) How does one do access path selection in a multiple data model environment?

4.3. Graphics-Based Data Models

We have a simple geographic data model, called GQ, based on the relational model. Logical relationships are not restricted to just two levels as in GEOQUEL. For example, a state map may include the state as a whole, regions within the state, counties within the regions, and cities within the counties. GQ will be the basis for a much more general and powerful graphics based data model.

5. Database Theory (Klug)

With our work on multiple data model support, we have been studying several theoretical problems:

(1) Given a view V over conceptual schema C, what functional and inclusion dependencies hold on V?

(2) Given a conjunctive query Q over conceptual schema C, find a minimal query equivalent to Q, taking into account the constraints in C.

(3) Given a conjunctive query Q of a particular form ("fan-out free"), find an efficient (i.e., polynomial) algorithm for optimizing Q (ignoring dependencies in the schema).

6. Wisconsin Storage System (Boral, DeWitt, Katz, Klug)

WiSS is a low-level database system designed for easy experimentation by simplifying the inclusion of new access methods and the construction of special purpose front-ends. WiSS supports a basic B-tree access method, with "hooks" for concurrency control and recovery. The interface to WiSS is similar to System-R's RSS -- creation/deletion of files and access paths, and the scanning of files either sequentially or via access paths.

We envision a multitude of uses for our storage system, including using it for the storage of geographic, statistical, and VLSI circuit data.