An Evaluation of a Generalized Database Subsystem

TAKENORI MAKINO,* MASAYUKI MIZUMA,* SHIGEKI HIYOSHI,*
MASANOBU WATANABE* and KATSUYA HAKOZAKI*

A feasibility study for a database machine is discussed. A generalized database subsystem (GDS), which supports different data models, shares a host main memory with the host processor, and basically manipulates a single record. From the experiment, it is shown that the GDS can perform data manipulation in a database system at high performance level due to the combination of firmware and hardware, also the host processor offload rate to the GDS is an important factor in increasing system throughput. These results will be useful to design practical database systems.

1. Introduction

Database oriented systems have been increasing in number, in accordance with advances in database management techniques. This trend poses a significant database performance problem. Extensive use of database management functions puts a heavy database load onto a system. As a result, the system tends to be database management bounded. Currently used general purpose computers are not designed to effectively manipulate non-numeric data, while database management functions require extensive non-numeric data manipulation power. A database machine is expected to be the solution to establish cost effective database systems.

There are several approaches to database machine problems, as Champine explained in his article [1]. These approaches can be classified into three categories, in view of data search algorithms. One is based on full-scan. Logic-per-track devices such as RAP [2], RARES [3] and, CASSM [4] are included in this category. The second category is an extension of conventional search algorithms, which use indexes and pointers (or links). A classical back-end processor XDMS [5] is in this category. The third category is a combination of the above two approaches. DBC [6] by Hsiao and others is classified into this category. Each has different advantages and disadvantages. They should be evaluated from various viewpoints.

Based on an assumption that a large capacity high-speed memory device, which can replace moving-head disk devices, and a high-speed processing element will be economically available, a full-scan database machine is advantageous. An associative memory can be economically constructed under the above assumption. Application systems will fully utilize the associative memory capability for obtaining rapid response to a variety of queries.

However, there are occasions when the above assumption is not satisfied. Disk devices have been dominantly used for database storage media and for some time in the future will continue to be used. Proposed new memory devices, such as charge-coupled devices or magnetic bubble devices, are not penetrating the market with the rapid growth rate expected, and the cost per bit is not yet sufficiently low. This trend does not support the assumption. Moreover, there are a large number of application systems in which some fixed programs are executed repeatedly for operational use. Traditional database management systems favorably process this application category, whereas associative memory utilization is not so advantageous for this category. For these reasons, it is considered premature to introduce a full-scan database machine into a practical system.

The third approach is characterized a as combination of a small capacity associative memory and a large conventional memory device. The proposed systems, such as DBC, seem attractive for a variety of application classes. However, they are based on several new storage device technologies, which are not proven as yet. Based on the state-of-the-art technology, the approach has to be evaluated for its feasibility from a more practical viewpoint.

From a practical viewpoint, it is of great importance that a database machine be developed based on conventional search mechanisms, but that its database processing power be greatly enhanced by using firmware and hardware technology which, recently, has shown significant improvement. Several non trivial problems exist in this line. Functions to be performed in a database machine, the relationship between implementation methods for the functions and their performance, the interconnection between a host processor and the database machine and storage management are some of the issues to be explored. It is necessary to obtain extensive information about these problems. A previous paper on a conceptual design for Generalized Database Subsystem (GDS) [7], provides a basis for this study.

An experimental system, to be described in this paper, has been developed to realize the GDS design concepts. The major objectives of this experiment are as follows:

1) Obtaining the performance related data which can be used to determine trade-offs for various im-

*Computer System Research Laboratory, C & C Systems Research Laboratories, Nippon Electric Co., Ltd., Japan.

plementations.

2) Examining the functional trade-offs between the host processor and the database machine for typical database management systems.
3) Exploring some problems encountered to realize a practical database machine.

A user microprogrammable minicomputer is used as a vehicle for this experiment in order to evaluate software/firmware/hardware trade-offs. The minicomputer is connected to a host, which is a medium scale general purpose computer (ACOS System 400).

In this paper, the data manipulation in a database system is discussed intensively. The storage system, which is another major part of the database system, is to be evaluated separately for a storage hierarchy management and is explained elsewhere.

Functional aspects of the GDS are briefly reviewed in the following section. The experimental system and the performance data obtained by the experiment are explained. Based on the results, relational database performance figures are estimated. The possible performance enchancement, when a practical database machine is to be realized, is also discussed.

2. Generalized Database Subsystem

Generalized Database Subsystem (GDS) provides fundamental database functions which are efficiently used to construct database management systems based on different data models. A generalized high level functional interface (GDI: Generalized Database Interface) is defined to separate the database functions from a host processor. The GDI is a single record access interface, which can support a CODASYL and a relational database. As the GDS design concepts are reported in reference [7], only brief reviews are made here.

2.1 Generalized Database Subsystem Functions

The GDS completely controls logical and physical storage structures. This means that the host processor is free from executing complex physical and logical input-output functions and also that the GDS is able to keep the physical storage structures at an optimum state without affecting the host processor.

The logical storage space is introduced into the GDS, which is independent of physical devices and data structures. A logical record address consists of an area number, a page number and a line number. Database objects are organized with any of the following three storage structures: (1) Key Sequenced, (2) Key Randomized and (3) Entry Sequenced structures. In addition, two kinds of secondary access paths, Link and Image, can be set up for database records. Link represents the parent-child relationship between records, which is provided by means of a pointer chain. Image is a secondary index in which each entry contains a pair of a key value and a logical record address.

2.2 Generalized Database Interface

The GDI commands are classified into three categories: (1) database definition, (2) data manipulation and (3) control commands. The outline of GDI commands is explained with two examples; a relational and a CODASYL database.

Figure 1 shows how a query language is decomposed into a sequence of GDI commands. A database structure is described in Fig. 1(a). It is assumed that the EMP (employee) relation has the image path IMGLOC for the LOC (location) field and that the link path LDNO is defined to specify a join relationship between the EMP relation and the DEPT (department) one.

The query in Fig. 1(b), which is represented in QUEL of INGRES [8], means to list the name, salary and department for the employees who work in Tokyo and get more than 200,000 yen in salary. The query is decomposed as in Fig. 1(c). At first, the START-TRANS command sets up a communication path between the

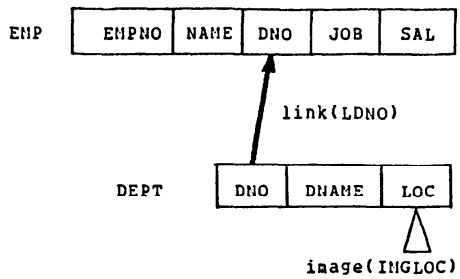

Fig. 1(a) Data structure.

```
RANGE OF E IS EMP
RANGE OF D IS DEPT
RETRIEVE E.NAME, E.SAL, D.DNAME
        WHERE D.LOC="TOKYO"
        AND D.DNO=E.DNO
        AND E.SAL GT "200000"
```

Fig. 1(b) Example of a query.

```
001  START-TRANS return (trans-id (TR1));
002  USE-AREA (area (DBAREA), shared retrieval);
003  INIT-PATH (image (IMGLOC), image-key="TOKYO")
                                  return (cur-id (C1));
     do while S1=exist;
004  GET-IMAGE-NEXT (rec-id (DEPT), cur-id (C1),
     f-list (F1.DNAME), s-cond (LOC="TOKYO"))
                                  return (status (S1));
005  INIT-PATH (link (LDNO), cur-id (C1))
                                  return (cur-id (C2));
     do while S2=exist;
006  GET-LINK-NEXT (rec-id (EMP), cur-id (C2),
     f-list (F2.NAME, F3.AGE), s-cond (SAL GT "200000"))
                                  return (status (S2));
     end;
     end;
007  RELEASE-PATH (cur-id (C1, C2));
008  RELEASE-AREA (area (DBAREA));
009  END-TRANS (trans-id (TR1));
```

Fig. 1(c) GDI command sequence.

Fig. 1 Relationship between a query language and GDI commands.

host processor and the GDS (in statement 001). Then, the INIT-PATH command sets the Image starting point into currency indicator C1 (003). By the GET-IMAGE-NEXT command, the object record is found and the next record address on the Image is set into C1 (004). Then, the INIT-PATH command creates another currency indicator C2 for Link, and the starting point is obtained from the current record address in C1 (005). The GET-LINK-NEXT command selects the next record, on the Link, which satisfies the condition: E. SAL>"200,000" (006). When the data manipulation is completed, the END-TRANS command terminates the communication path (009).

A relationship between the CODASYL data manipulation language (DML) and the GDI command is discussed below. At a DB (subschema-name) statement in a user COBOL program, the logical data management on the host processor makes ready for database processing. That is, it generates a START-TRANS command and also creates access paths for every record class and set class defined in a subschema. DML commands in the user program are translated into GDI commands. In most cases, one DML command corresponds to one GDI command. For example, FIND-ANY, FIND-NEXT and GET record DML commands correspond to GET-PRIM-DIRECT, GET-LINK-NEXT and GET-CURRENT record commands in the GDI, respectively.

As described above, the GDS can be applied to both relational and CODASYL database management systems.

3. Experimental Database Machine System

Figure 2 shows the experimental system structure. At present, two database management systems are implemented on the experimental system. One is a CODASYL database system, which is completely compatible with a commercial database management system. The other is a simplified query system, which is operated either on a stand alone system or with a host processor.

3.1 GDS Processor

A user microprogrammable minicomputer, Varian V-76 is used as a GDS processor. Basic functions and control programs, which are implemented by firmware, are stored in writable control storage (WCS). The developed microprograms are about 2 K steps. Main memory consists of two banks. One is used to store the programs and control tables. The programs are about 20 K steps. Some programs are overlaid. The other is used for database buffers. The disk is applied to programs and database areas. The V-76 computer is a 16-bit machine, with a memory cycle of 660 ns (nano-seconds); a WCS cycle of 195 ns; a one word instruction time of 1320 ns; a double word instruction time of 1980 ns.

3.2 Inter-Processor Communication Mechanism

Figure 3 shows the inter-processor communication

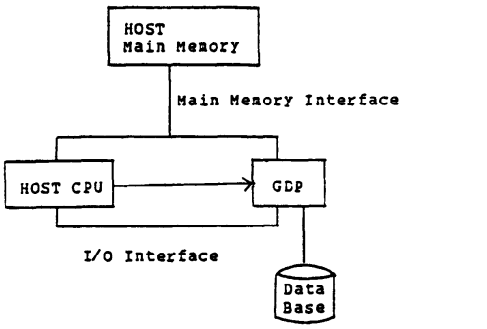

Fig. 2 Experimental system.

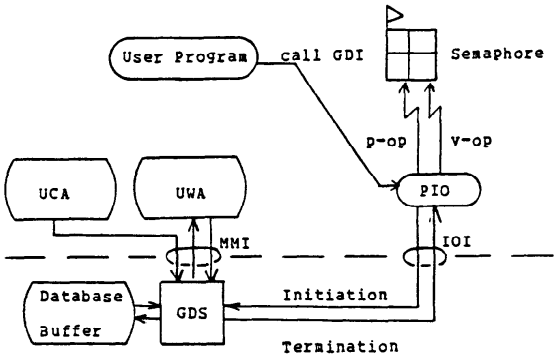

Fig. 3 HOST-GDS interface.

mechanism. A user program specifies a GDI command or a GDI command sequence in the user communication area (UCA) and initiates a CALL GDI operator with the UCA address and transaction identification. The host processor and the GDS processor share the host main memory through the main memory interface (MMI) and are connected through and I/O interface. In order to minimize the inter-processor communication overhead, the data and GDI command parameters are transferred through the main memory interface. The I/O interface is used for process synchronization, that is, for initiating a GDI command or for a termination notification.

3.3 GDS Function Modules

The GDS processor consists of several function modules are classified into four categories; Transaction management, Logical data access, Physical data access and Page management modules.

Transaction management modules schedule transactions, communicate with the host processor, and back

out transactions. Logical data access modules perform data retrieval and maintenance functions on the database. Data access module (DAM), which is a main module, selects and updates data through the specified access path. The information for data access is obtained from a data catalog by Data descriptor access module (DDA). Physical data access modules, which are implemented by firmware, translate a logical page number to a buffer memory address and manipulate the physical record in the specified page. An example of one of them, is the Data transfer module (DXM), which selects the target record in the page using a binary search method and transfers the specificed values to the main memory. Page management modules split, allocate and replace pages.

A GDI command is executed by using the above function modules. The routing among the function modules is specified for each GDI command, and also the recursive operation scanning indexes or link pointers are specified.

Several "primitives" are defined for each function module in order to denote the actions performed in it. When a "primitive" operation is initiated, the primitive parameter values are set in a fixed area of transaction control block and a function call is performed. As the function module is implemented either by software, firmware or hardware, the function call is accomplished by a return-address-stack-and-jump, a WCS jump or a I/O instruction, respectively. Transaction changing, when a page fault occurs or a locked page is referred to, is performed by setting new transaction identification into an index register, which is used to refer to the control block. This flow control scheme tends to reduce transaction transition time among function modules and overhead time for transaction change.

4. GDS Performance Evaluation

The GDS experimental system performance is measured on a simple and small database, whose structure is shown in Fig. 1(a). The conventional database management system performance is also measured on a medium scale conventional computer with 0.25 MIPS under the same environment. The I/O time is omitted in this evaluation, in order to concentrate the discussion on the database processing.

4.1 GDS Performance

GDI command execution times measured in the experimental system are shown in Table 1. It is shown that the GDS performance in the experimental system is much better than a medium scale conventional computer. For example, a GET-PRIME-DIRECT command without field value transfer is performed in 1.04 milli-sec., while a FIND-ANY DML, which corresponds to the GET-PRIME-DIRECT, is performed in 6.56 milli-sec., in the conventional computer. A GET-LINK-NEXT command is performed in 0.49 milli-sec., while a FIND-NEXT DML execution time is 4.26 milli-sec. in the conventional computer. The performance ratio for the GDS and the conventional computer is about one to eight. Therefore, it is considered that the GDS performance is the same order as a conventional computer with 2 MIPS, regarding the database processing. The GDS performance gain is considered to be brought by the following factors;

a) Firmware effects: The firmware effects are examined during module implementation. When a module implemented by software is implemented again by firmware, the execution time is reduced to 1/4~1/10 of the time. It can be said that the performance of a module is increased by a factor of at least four by firmware implementation.
b) Simplified system control: The system control scheme is specialized for database processing. Overhead time required for a transaction change and a function module call is sufficiently small. For example, a function module call can be performed in 2.6 micro-sec., while a call instruction in the conventional computer is performed in about 30 micro-sec. because of the time required for saving registers and processor status.

To explain software/firmware/hardware trade-offs to achieve a high performance GDS, two typical commands

Table 1 GDI command execution time in the experimental system.

Command	Time (msec)	Command	Time (msec)
Get-prime-direct	0.75+0.29n+0.13f	Store (KR)	1.55+0.1n
Get-prime-next	0.28+0.29n+0.13f	Delete (KR)	0.97
Get-image-direct	0.42+0.13y+0.13f	Store (ES)	1.05
Get-image-first	0.41+0.13f	Delete (ES)	0.59
Get-image-next	0.21+0.13f	Update (field)	0.20+0.28f
Get-link-first	0.41+0.13f	Update (record)	0.51
Get-link-next	0.49+0.13f	Connect	0.99
Get-link-owner	0.46+0.09m+0.13f	Disconnect	0.69
Get-current	0.11+0.13f	Get-scan-next	0.12+0.15x+0.13f

Note: f: the number of records
m: record position on a Link
n: record position on a hash collision chain
x: the number of scanning records
y: index level

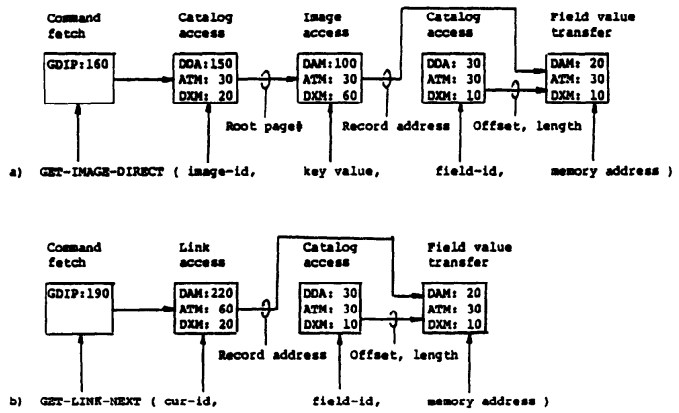

Fig. 4 Function module elapsed time (micro-sec).
ATM: Address translation module (firm).
DXM: Physical data access module (firm).
GDIP: GDI command processor (soft).
DAM: Logical data access module (soft).
DDA: Data descriptor access module (soft).

are discussed as examples. Figures 4 shows the functional module elapsed time in the experimental system. From the results of GET-IMAGE-DIRECT command, in Fig. 4(a), and GET-LINK-NEXT command, in Fig. 4(b), further performance improvement will be accomplished as follows;

a) Firmware implementation of software modules: Although most of basic functions are implemented by firmware, the ratio of software modules elapsed time to firmware modules elapsed time is about 1 to 1.5. To achieve high performance, most of the main function modules need to be implemented by firmware.

b) Hardware implementation of basic modules: The execution time of address translation module is 30 micro-sec. Adopting a hardware mechanism, as described in reference [7], the execution time becomes less than 2 micro-sec., including parameter transfer time. The performance for Data transfer module, which manipulates physical data, is limited by memory access time. Therefore, the execution time is estimated to be reduce to about 1/4 by adopting high speed memory devices and a four byte memory system.

By such improvements, the GDI command execution times, shown in Fig. 4, will become:

a) GET-IMAGE-DIRECT: $(100+25y+20f)$ micro-sec.,

b) GET-LINK-NEXT: $(110+20f)$ micro-sec.

Compared with the execution times shown in Table 1, these GDI command execution times reduce to about 1/5, when index level y is two and the number of fields f is four. Therefore, GDS performance will be increased about five times more than in the experimental system, and will correspond to a large scale conventional computer with 10 MIPS.

4.2 Communication Performance between Host Processor and GDS

The data transfer rate through the main memory interface is limited by the minicomputer data transfer rate, which is about 1 Mbyte/sec. This transfer rate is large enough to transfer commands, parameters and resulting data.

On the other hand, communication through the I/O interface is accompanied with overhead time, which is about 1.5 milli-sec. in the experimental system, because a general I/O package for online adaptors is used. To reduce the overhead, a special I/O package, which initiates and terminates GDS at high speed, or interprocessor communication mechanism should be adopted. By such improvement, the communication overhead, including process synchronization overhead, will be reduced to the order of 100 micro-sec. in medium scale computers. However, this improvement needs an alteration to the existing operating systems.

4.3 CODASYL Database Support

As described in the previous section, GDI commands have equivalent commands in the CODASYL data manipulation language. An object program generated by a conventional database compiler is transformed into GDI command sequences by a simple translator made for GDS. Therefore, most of a data base management system is moved to GDS, while that remaining on the host system is merely the translator. In the experimental system, the running time of the translator is reduced to less than 20% of the whole database management system. That is, the offload rate of the host to GDS is more than 80% in database management system. However, introducing GDS, communication time is needed to initiate and terminate GDS. If it takes 1.5 milli-sec., the

communication time becomes longer than a GDI command execution time. Therefore, the communication time should be reduced to the order of 100 micro-sec. as described in the previous section.

4.4 Relational Database Support

A relational database query is decomposed into a GDI command sequence whose access cost becomes minimum by access path selection strategies [10], [11]. The execution time for a query after the decomposition is discussed in the following with the example query shown in Fig. 1(a). Though I/O time is important, it is neglected.

Let n_r be the number of restricted tuples in the DEPT relation and n_j be the expected number of tuples to be joined in the EMP relation for each tuple. The following three cases, which contain the best and the worst cases for GDS, are examined;

a) When Image for restriction and Link for join exist: The retrieval time is represented as

$(\text{GET-IMAGE-DIRECT})_{\text{DEPT}}+(n_r-1)(\text{GET-IMAGE-NEXT})_{\text{DEPT}}+(n_r)(n_j)(\text{GET-LINK-NEXT})_{\text{EMP}}$,

where $(\text{command name})_{\text{A}}$ means the execution time for the command for relation A.

b) When Image for restriction and Image for join exist: The retrieval time is represented as

$(\text{GET-IMAGE-DIRECT})_{\text{DEPT}}+(n_r-1)(\text{GET-IMAGE-NEXT})_{\text{DEPT}}+(n_r)((\text{GET-IMAGE-DIRECT})_{\text{EMP}}+(n_j-1)(\text{GET-IMAGE-NEXT})_{\text{EMP}})$.

c) When neither Image nor Link for restriction and join exists: In this case, if not sorted, the restriction and the join operations are performed by a scan operation. Let N_{EMP} be the number of tuples in EMP relation and let N_{DEPT} be the number of tuples in DEPT relation. The retrieval time is represented as $N_{\text{DEPT}}(\text{GET-SCAN-NEXT})_{\text{DEPT}}+n_rN_{\text{EMP}}(\text{GET-SCAN-NEXT})_{\text{EMP}}$.

Consider that there are 10^4 tuples in DEPT relation, 10^5 tuples in EMP relation, $n_r=10$ and $n_j=10$. The execution time in case a) becomes 6.04 milli-sec., in case b) 8.44 milli-sec. and in case c) 1.5×10^5 milli-sec.

When there is neither Image nor Link for restriction and join operations, the execution time by scanning search on relations becomes too long to be used in real time, even if I/O time is neglected. Therefore, in order to achieve a high performance relational database system, it is desirable to use a sorting operation in supporting a join and a restriction operation without Image or Link. The sort operation will be effectively performed by a firmware implementation or introducing sorting hardware.

5. System Performance Evaluation

The system performance is dependent on the host processor offload rate to a GDS as well as the GDS performance. As the host processor and the GDS operate in parallel, the system performance, in multiple transactions environment, can be estimated by a closed queueing network [12].

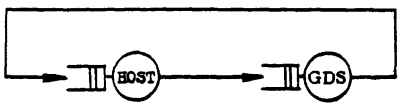

Fig. 5(a) Evaluation model.

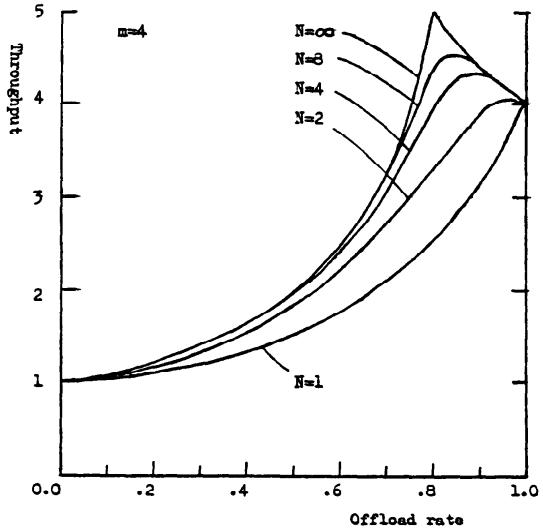

Fig. 5(b) Relationship between system throughput and offload rate; m: performance rate, N: the number of transactions.

Fig. 5 System throughput evaluation.

Let an expected transaction service time by only the host processor be unity. Assume that the transaction service time by the host processor or the GDS is exponentially distributed and that the GDS can execute the transactions offloaded from the host processor m times faster than the host processor. Therefore, when the host processor offload rate to the GDS is α $(0\le\alpha\le1)$, the expected transaction service times by the host processor and the GDS are $(1-\alpha)$ and α/m, respectively. Figure 5(a) shows this evaluation model, where I/O devices are neglected. When there are N transactions in this system, the probability $p(n)$ that there are n transactions in the host processor can be obtained from [12], as follows:

$$p(n)=\left(\frac{m(1-\alpha)}{\alpha}\right)^n\Big/\sum_{i=0}^{N}\left(\frac{m(1-\alpha)}{\alpha}\right)^i.$$

In this case, system throughput T can be derived as:

$$T=[1-p(0)]/(1-\alpha).$$

After some computations, the system throughput can be rewritten as:

$$T=\begin{cases}\dfrac{m\{\alpha^N-[m(1-\alpha)]\}^N}{\alpha^{N+1}-[m(1-\alpha)]^{N+1}}, & \text{for } 0\leqslant\alpha\leqslant1,\ \alpha\neq\dfrac{m+1}{m},\\[2ex] N(m+1)/(N+1), & \text{for } \alpha=\dfrac{m}{m+1}.\end{cases}$$

From this equation, given offload rate α and GDS per-

formance ratio to the host processor m, the system throughput is derived. Figure 5(b) shows the relationship between the system throughput and the offload rate, rate, when that performance rate m is 4. The offload rate at the maximum throughput varies with the number of transactions in the system, and inclines toward a higher number as the number of transactions decreases.

In a CODASYL database, it is said that the database management ratio to the whole system is about 1 to 2, application program rate is 20% and data communication rate is 30%. The offload rate to the GDS becomes 40%, because 80% of the database management is offloaded to the GDS. Therefore, the system throughput increases by 67%. The host processor and GDS system executes databse applications with high performance per cost, because the GDS can be implemented by a small amount of hardware.

On the other hand, the relational database also has a problem with respect to the offload rate. Before this experiment, the time required for a query parsing had been considered to be smaller than the data manipulation time. However, if often takes longer than the data manipulation to perform the query parsing, even in this simplified query system. A paper about the performance evaluation, in INGRES [13], reports that the performance characteristics of two query types: data-intensive queries and overhead-intensive queries, are so different that it may be difficult to design a single architecture that is efficient for both. It is also shown that, in the overhead-intensive queries, the data processing time is much less than the time for setting up the query, while the data processing time becomes greater than the query setting up time, in the data-intensive queries.

These results show that the host processor offload rate to the GDS is dependent on query property. If a query is overhead-intensive, the system throughput does not increase, as shown in Fig. 5(b), even if the GDS has high performance. On the other hand, the system throughput may be effectively increased for a data-intensive query.

From the above discussion, in order to support a relational database effectively in a general case, the offload rate should be raised. Therefore, the command interface between the host processor and the GDS should be set up to a query level, rather than a single record interface, and it is necessary that the GDS perform the query parsing and access path selection. It is expected that high performance query processing can be achieved by implementation of firmware and hardware modules in a similar manner to the GDS, as described in this paper.

6. Conclusion

An experimental database machine is implemented to verify the efficiency of a GDS in database systems. The GDS, in the experimental system, is implemented on a microprogrammable minicomputer, and the basic functions are performed by firmware. The experimental system is designed to obtain the performance information for hardware/firmware/software trade-offs in the database machine implementation.

A logical interface between a host processor and the GDS, called a GDI, is a highly functional single record interface effectively supporting not only a CODASYL database, but also a relational database. In the experimental system, a simplified query system and a CODASYL database interface program are provided on the host system for the experiment.

From results in the experiment, the GDS performance was evaluated as follows:

1) The GDS performance on a minicomputer is the same order as a conventional computer with about 2 MIPS. With further improvement, it is expected to correspond to a conventional computer with 10 MIPS.
2) A command, data and parameter transfer time is sufficiently short because of a memory sharing interface used between the host processor and the GDS.
3) A command or command sequence initiation and termination time is too long to neglect, because an I/O interface is used. It is, however, reduced to a low overhead by inter-processor communication.
4) The offload rate to the GDS, in an online CODASYL database, may be about 40% of the system. Introducing the GDS, the system throughput will increase by about 70%.
5) In the case of a relational database, the offload rate is dependent on a query type. For effective support both data-intensive and overhead-intensive queries, the command interface should be set up to a query level, rather than the GDI.

As described above, data manipulation in a CODASYL and a relational database is effectively performed by the GDS, which is a special processor oriented to database processing. However, some problems exist about the offload rate and the communication overhead. Under this condition, possible alternatives for a practical database machine implementation are considered as follows;

1) A tightly coupled database machine with a host processor: It should be called a database booster. The database booster performs GDI commands to support a common part of a CODASYL and a relational database. It will realize a high cost-performance database system for a CODASYL database and a data-intensive query.
2) A loosely coupled database machine: The database machine is connected with a host processor through an I/O interface, and has a query level command interface. In this case, it is not necessary to use a main memory sharing interface, because the communication overhead time may be small enough compared with query execution time.

These results will be useful to implement a practical database machine, according to application fields.

However, some important problems to implement practical database systems remain, which include the following;

1) Database I/O and memory hierarchy management.
2) Strategies for query parsing and access path selection.
3) Total system architecture including a recovery facility for the system failure.

These problems will be examined for a CODASYL and a relational database.

Acknowledgement

The authors are grateful for continuous encouragement and advices of Mr. K. Nezu of Nippon Electric Co., Ltd. Thanks are also due to Dr. A. Sekino of Nippon Electric Co., Ltd for his appropriate suggestion.

References

1. CHAMPINE, G. A. Current Trends in Data Base Systems, *Computer*, **12**, 5 (May 1979), 27–41.
2. OZKARAHAN, E. A., SCHUSTER, S. A. and SMITH, K. C. RAP-An Associative Processor for Data Base Management, *Proc. NCC* (1975), 379–387.
3. LIN, C. S., SMITH, D. C. P. and SMITH, J. M. The Design of a Rotating Associative Memory for Relational Database Applications, *ACM Trans. Database Systems*, **1**, 1 (1976), 53–65.
4. SU, S. Y. W. and LIPOVSKI, G. J. CASSM: A Cellular System for Very Large Data Bases, *Proc. International Conference on VLDB* (1975), 456–472.
5. CANADAY, R. H. and et al. A Back-End Computer for Data Base Management, *Commun. ACM*, **17**, 10 (Oct. 1974), 575–582.
6. BANERJEE, J., HSIAO, D. K. and KANNAN, K. DBC-Database Computer for Large Databases, *IEEE Trans. Comput.*, **C-28**, 6 (1979), 414–429.
7. HAKOZAKI, K., et al. A Conceptual Design of a Generalized Database Subsystem, *Proc. 3rd VLDB* (1977), 246–253.
8. STONEBRAKER, M. WONG, E., KREPS, P. and HELD, G. The Design and Implementation of INGRES, *ACM Trans. Database Systems*, **1**, 3 (1976), 189–222.
9. BAYER, R. and MACCREIGHT, C. Organization and Maintenance of Large Ordered Indexes, *Acta Inf.*, **1**, 3 (1972), 173–189.
10. YAO, S. B. Optimization of Query Evaluation Algorithms, *ACM Trans. Database Systems*, **4**, 2 (1979), 133–155.
11. BLASGEN, M. W. and ESWARAN, K. P. Storage and Access in Relational Data Bases, *IBM Syst. J.*, 4 (1977), 363–377.
12. Gordon, W. J. and NEWELL, G. F. Closed Queueing Systems with Exponential Servers, *Opns. Res.* 15 (1976), 254–265.
13. HAWTHORN, P. and STONEBRAKER, M. Performance Analysis of a Relational Data Base Management System, *Proc. ACM-SIGMOD* (1979), 1–12.

(Received June 26, 1981: revised September 18, 1981)