AN EFFICIENT IMPLEMENTATION OF A RELATIONAL DATA BASE

Marian S. Furman

Institute for Scientific, Technical and Economic Information
00-926 Warszawa, ul. Żurawia 3/5, Poland

Abstract

In the paper an approach to implement efficiently a relational data base is presented. The approach combines a new method of physical representation with a novel database machine (DBM) architecture. The dispersed representation assumes a relation consisting of three parts: the identification, value, and link (optionally). The parts are separated and the identification and link parts are converted to be expressed with pointers, and then the parts are dispersed among three memories: the IDENTIFICATION, VALUE, and LINK. The DBM architecture attempts to implement the dispersed representation efficiently and uses modified cellular associative memories (CAM) to store and process the value and link parts. The modification consists in providing a CAM with a respond register called a global mark register (GMR) which can be readily loaded and unloaded externally.

Introduction

Database machines (DBM) [Bane79, Bora82, Cope73, DeWi79, Hsia83, Ozka75, Schu79, Su79a] optimise the data base processing by means of hardware implementation of the basic database operations. However, the achieved performance is not fully satisfactory [Hawt82], especially in the case of the join operation. It can be caused by unsufficiently addressing the sematic meaning of data kept in a data base when developing the hardware algorithms.

When developing a data base to create a model of a ceratain real world situation two things are usually conceptualised: the objects and the relationships among them. However, the distinction tends to vanish in the data base structure and processing. The hardware algorithms of the join [Babb79, Hsia83, Kits83, Meno81, Ozka83], which is predestinated to compute the relationships, are in majority designed for any kind of join and any type of attributes. In normalised relational data bases the natural join on key attributes is most frequently used so this type of joins should be optimised as much as possible.

In the paper we propose an approach to implement a data base which assumes that to store and process a data base efficiently, the information concerning objects should be separated from the information concerning relationships.

In Section 2 the dispersed physical representation of a relational data base is presented. A relation is assumed to consist of three parts: the identification, value, and link (optionally). The parts are separated and the identification and link parts are converted to be expressed with pointers, and then the parts are dispersed among three memories: the IDENTIFICATION, VALUE, and LINK. In Section 3 the cellular associative memory (CAM) being emulated by such DBM's like CASSM [Cope73, Su79a], and RAP [Ozka83, Ozka75, Schu79] is presented. The modification consisting in providing the CAM with a respond register is described. The global mark register (GMR) can be readily loaded and unloaded externally. The GMR enables to mark quickly any set of tuples stored in the CAM as well as to transform a set of marked tuples into a set of pointers to them. In Section 4 a new DBM architecture is presented, which attempts to implement the dispersed representation efficiently. Two modified CAM's are used to implement the VALUE and LINK memories. The memories are bridged by the Pointer Processing Unit which also implement the IDENTIFICATION memory (in RAM). The GMR's allowing fast pointer processing serve as a means of communication between the DBM components. In Section 5 we describe the execution of the set operations: the union, difference, and intersection as well as the join operation. Joins are classified by dividing it into three types: the identifying, linking, and associating according to the aim of the join. In Section 6 the execution of a typical query in the DBM is described. Finally, in Section 7 conclusion remarks are given. The dispersed representation along with the proposed DBM architecture promise significant improvement in the preformance due to the simultaneous processing of the value, link, and identification data base parts and a higher specialisation of the processing devices. However, the most spectacular benefit is that execution costs of the identifying joins are practically negligible.

2. The dispersed physical representation

The dispersed representation bases on the perception that attributes in a tuple are used for three purposes:

a) to hold a value (every attribute)
b) to identify the tuple (key)
c) to provide a link with a tuple of other relation (joinable attributes)

The (a) and (c) attribute functions are well understood while the (b) one is more of the operational nature. The identifying function consists in a transformation of a set of keys into a set of marked tuples or vice versa. Hence, to identify a tuple when its key is known means to mark the tuple, in opposite, to identify a marked tuple means to send its key value to the controller.

Some attributes fulfil more then one function and that is convenient on the conceptual level but causes problems on the physical level since all the functions are performed in one place and have to be accomplished in a sequential manner.

For example, suppose we have a relation EMP with the attributes NAME (value and key), SALARY (only value), and DEP # (link, i.e. a foreign key to the relation DEP). Assume the relation EMP is stored as a file of records. The following tasks corresponding to the three mentioned functions are to be performed: (a) to find employees with SALARY > 10000, (b) to identify (mark) the tuple with the key NAME = Codd, and (c) to provide a link with the relation DEP, i.e. to mark the tuples in the relation DEP matching the value DEP # of a certain (previously marked) tuple in the relation EMP.

The tasks (a) and (b) are realised by proper selections in EMP, while to perform the task (c), first the value of the marked tuple in EMP is sent to the controller, next it is used as a argument of a selection in the relation DEP. In most systems the tasks (a), (b), and (c) have to be performed sequentially. Futhermore, if the tasks (b) and (c) concern many tuples it must be repeated for each tuple.

The proposed dispersed representation attempts to eliminate the drawbacks and enable simultaneous performance of the three functions. This is accomplished, firstly, by relieving the key attributes from performing the identifying function and passing the duty to a tuple address, secondly, by splitting up a relation to separate the value and link attributes.

We now present the method with some simplifying assumptions concerning the data base. Suppose that a data base consists of a set of normalised relations $R_1, R_2, \ldots, R_p$. A relation scheme R^* denotes a set of attributes $A_1, A_2, \ldots, A_r$ composing the relation R. Let A denotes the set of all attributes. From the set A we arbitrary chose a subset J consisting of joinable attributes. We want such a physical data base organisation that natural join and equijoin operations involving a comparison of the attributes from the set J will be performed especially quickly.

Let consider the case when the set J consists of all key attributes. We distinct two types of relations: a simple one and a compound one. The simple relations have no foreign keys while the compound ones do. We also assume that each foreign key in the compound relations point to only one simple relation. The compound relations can be futher divided into two subtypes, i.e. "purely" compound relations which all attributes are foreign keys and "mixed" compound relations which besides foreign keys have also other "value" attributes.

The dispersed representation requires a relation to be either simple or "purely" compound. Therefore, whenever R is the "mixed" compound relation it is decomposed onto two relations R-S and R-C. The relation R-S is a projection of R onto its all non-foreign key attributes while the relation R-C is a projection of R onto all foreign key attributes and the attributes which form a key of the relation R-S. Note that the relation R-S is simple and the relation R-C is "purely" compound. After decomposing all "mixed" compund relations in the above way we obtain a data base scheme consisting of simple and "purely" compond relations.

Then we replace foreign key attributes composing the compound relations by pointer attributes, e.g. if X is a foreign key in the relation R and originates from the simple relation R_1 then X is deleted from R and a new attribute named $R_1\uparrow$ is added which values are the addresses of proper tuples in the relation R_1. To keep this substitution valid we have to assume that the tuple addresses are constant.

Now, in terms of the dispersed representation, there is an ideal situation since the "value" part of a data base (simple relations) is separated from the "link" part (compound relations) so the parts may be stored and processed separately.

Assume that the mass memory consists of three parts: the VALUE, LINK, and IDENTIFICATION memories. We place the simple relations in the VALUE memory and the compound relations in the LINK memory. For a formal reason we assume that to each placed relation R a new address attribute $R\uparrow$ is added. A value of the attribute $R\uparrow$ is equal to an address of the tuple in the VALUE or LINK memory and matches a pointer to the tuple. We denote a simple relation R augmented with the attribute $R\uparrow$ and stored in the VALUE memory by R[V] while a compound relation R stored in the LINK memory by R[L].

Finally in the IDENTIFICATION memory we establish an identifying part consisting of two unary relations R[ID] and R^* [ID] for each simple and compound relation. The relation R[ID] consists of the attribute $R\uparrow$ projected from either R[V] or R[L] and it is just a set of pointers to the tuples of the relation. The relation R^* [ID] consists of the attribute ATTR and contains names of the attributes composing the relation R on the conceptual level.

Let us illustrate the dispersed representation with an example of the data base shown in Fig. 1 consisting of three relations: S (suppliers), P (parts), and SP (shipments). There are foreign keys in the relation SP, i.e. S# and P#. Therefore the relation SP is classified as compound one while the relations S and P as simple ones. Precisely, the relation SP is a "mixed" compound one since the attributes QTY and DATE are not foreign keys. Hence the relation SP is decomposed onto the simple ralation SP-S (QTY, DATE) and the "purely"

S /suppliers/

S#	NAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris

P /parts/

P#	NAME	COLOR	WEIGHT
P1	Nut	Red	12
P2	Bolt	Green	17
P3	Screw	Blue	17
P4	Screw	Red	14

SP /shipments/

S#	P#	QTY	DATE
S1	P1	300	85-05-11
S1	P2	200	85-05-12
S1	P3	400	85-05-13
S2	P1	300	85-05-11
S2	P2	400	85-05-15
S3	P2	200	85-05-15

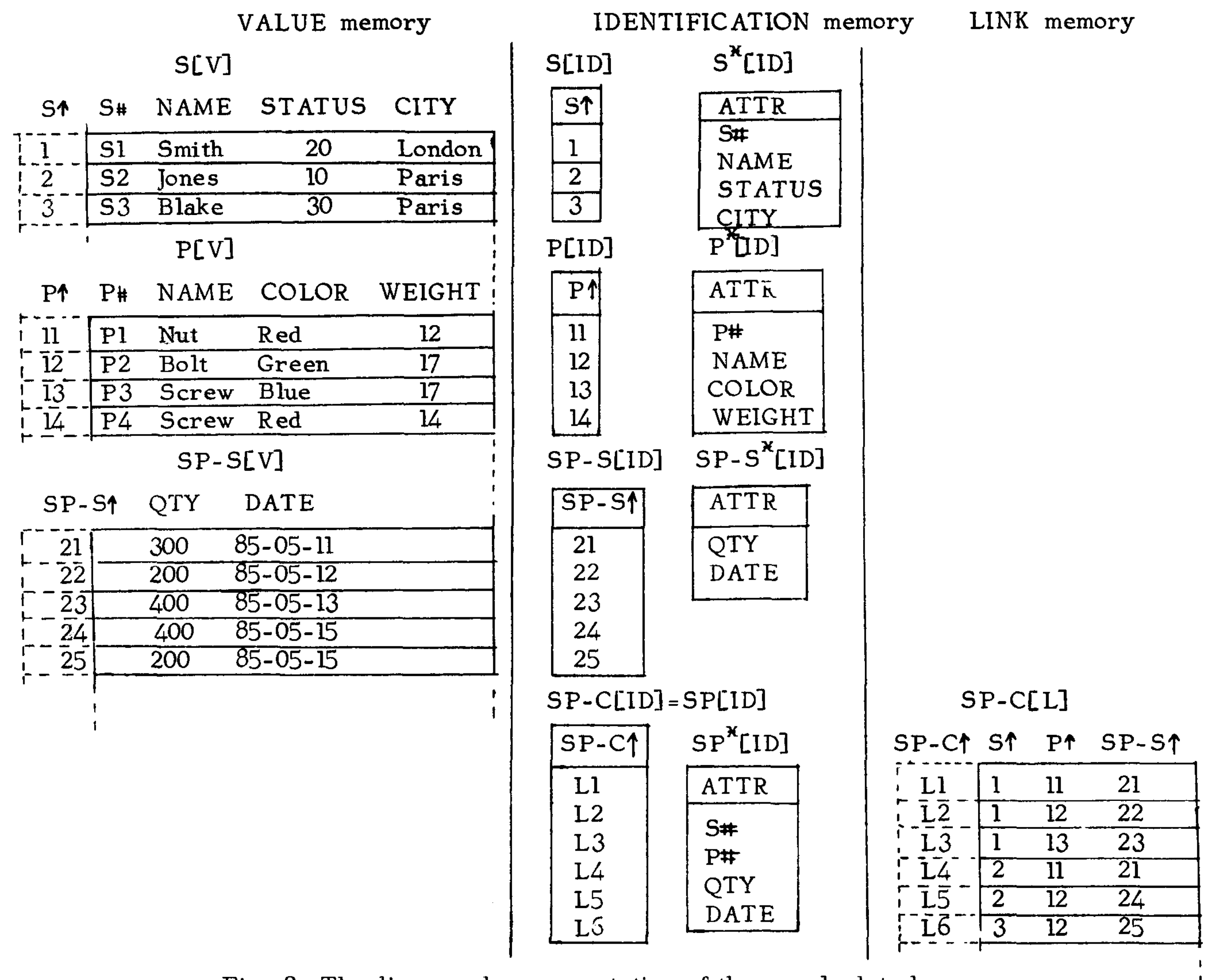
Fig. 1. The sample relational data base.

compound relation SP-C(S#, P#, QTY, DATE). Next the simple relations S, P, and SP-S are placed in the VALUE memory while the compound relation SP-C with its attributes converted into pointers in the LINK memory. After that for each relation a pair of the identifying relations (R[ID] and R^* [ID] is created in the IDENTIFICATION memory. The final arrangment of the data base in the three mass memory components is shown in Fig. 2 (the addresses in the LINK memory are preceded by letter L). Note that values of the added attributes ($R\uparrow$) need not be stored explicitly since they are determined by the tuple position (constant) in the VALUE or LINK memory.

To summarise, the dispersed representation of a simple relation consists of relations R[ID] and R^* [ID] stored in the IDENTIFICATION memory and the relation R[V] stored in the VALUE memory. The original relation R can be created by a natural join of the relations R[ID] and R[V] and then projection onto the set of attributes stored in R^*[ID].

$$R = \pi_{R^*[ID]} (R[ID] \bowtie R[V])$$

Hence a simple relation R can be viewed as an extract from a certain relation stored in the



VALUE memory

S[V]

S↑	S#	NAME	STATUS	CITY
1	S1	Smith	20	London
2	S2	Jones	10	Paris
3	S3	Blake	30	Paris

P[V]

P↑	P#	NAME	COLOR	WEIGHT
11	P1	Nut	Red	12
12	P2	Bolt	Green	17
13	P3	Screw	Blue	17
14	P4	Screw	Red	14

SP-S[V]

SP-S↑	QTY	DATE
21	300	85-05-11
22	200	85-05-12
23	400	85-05-13
24	400	85-05-15
25	200	85-05-15

IDENTIFICATION memory

S[ID]

S↑
1
2
3

S*[ID]

ATTR
S#
NAME
STATUS
CITY

P[ID]

P↑
11
12
13
14

P*[ID]

ATTR
P#
NAME
COLOR
WEIGHT

SP-S[ID]

SP-S↑
21
22
23
24
25

SP-S*[ID]

ATTR
QTY
DATE

SP-C[ID]=SP[ID]

SP-C↑
L1
L2
L3
L4
L5
L6

SP*[ID]

ATTR
S#
P#
QTY
DATE

LINK memory

SP-C[L]

SP-C↑	S↑	P↑	SP-S↑
L1	1	11	21
L2	1	12	22
L3	1	13	23
L4	2	11	21
L5	2	12	24
L6	3	12	25

Fig. 2. The dispersed representation of the sample data base

VALUE memory. The extract is defined by two relations stored in the IDENTIFICATION memory, i.e. the relation R[ID] determines it horizontally while the relation R* [ID] vertically.

In the case R is a compound relation and consists of "p" foreign keys to R_1, R_2, ..., R_p simple relations, its representation consists of the relations R[ID] and R* [ID] stored in the IDENTIFICATION memory, the relation R[L] stored in the LINK memory, and the relations R_1[V], R_2[V], ..., R_p[V] stored in the VALUE memory. The original relation R can be expressed as follows.

$$R=\pi_{R^*[ID]}(R[ID]\bowtie(R[L]\bowtie(R_1[V]\times..\times R_p[V])))$$

A compound relation R can be viewed as an extract from a certain implicit relation being a Cartesian product of the "p" simple relations stored in the VALUE memory. Horizontally the extract is cut out, firstly, by the relation R[L] and, secondly, by the relation R[ID], while vertically it is cut out by the relation R* [ID].

From the above expressions one can notice that to represent any compound relation no relations in the VALUE memory have to be created and to represent any derived relation Q being a "horizontal" or "vertical" extract of an already existing relation R only relations Q[ID] and Q* [ID] have to be created in the IDENTIFICATION memory.

The physical structure of a data base imposed by the dispersed representation strongly depends on the chosen subset J of the joinable attributes since a value of each attribute belonging to J have to be tied to one place in the VALUE memory to enable the substitution of the value by its address. For example the set J chosen in the paper consisting of keys creates a structure corresponding somehow to the entity-relationship approach [Chen76] with the simple relations as the entity sets and the compound relations as the relationship.

On the other hand, comparing the classification of relations imposed by the dispersed representation with the one given in [Codd79] one can notice that the simple and compound relations correspond to the kernel entity and kernel associative relations respectively, while the identifying relations R[ID] to the E (surrogate) - relations. We want to stress that our classficiation of relations, as opposed to Codd's which delt with the conceptual modelling, deals with the physical implementation and attempts to provide tools for high efficiency. Therefore we suggest that classifications established on the conceptual level should be reshaped into the simple, compound, and identifying relations.

The dispersed representation makes an extensive use of pointers therefore to implement it efficiently a system with a powerful pointer mechanism is required. We will describe such a DBM in the next sections.

3. The modified cellular associative memory (CAM)

A number of DBM's use a cellular associative memory (content-addressable memory) built up with cells of comparatively inexpensive memories and processors. An outline of the memory is shown in Fig. 3. It consists of a chain of cells and a controller. Each cell is composed of a processor P_i and a cell memory CM_i. The set of CM's serves as a mass memory to store a data base. The CM_i can be implemented on a track of the head-per-track disk, CCD memory, bubble memory, or even RAM memory. The associativity of the memory is achieved by parallel and synchronised processing of all processors.

The CAM's shows many advantages (fast search) but have some limitations. For example, they are not provided with an efficient mechanism for performing the identifying tasks. Some of the limitations can be overcome by a minor modification of the CAM architecture.

The proposed modification consists in providing every cell with a cell mark register CMR_i.

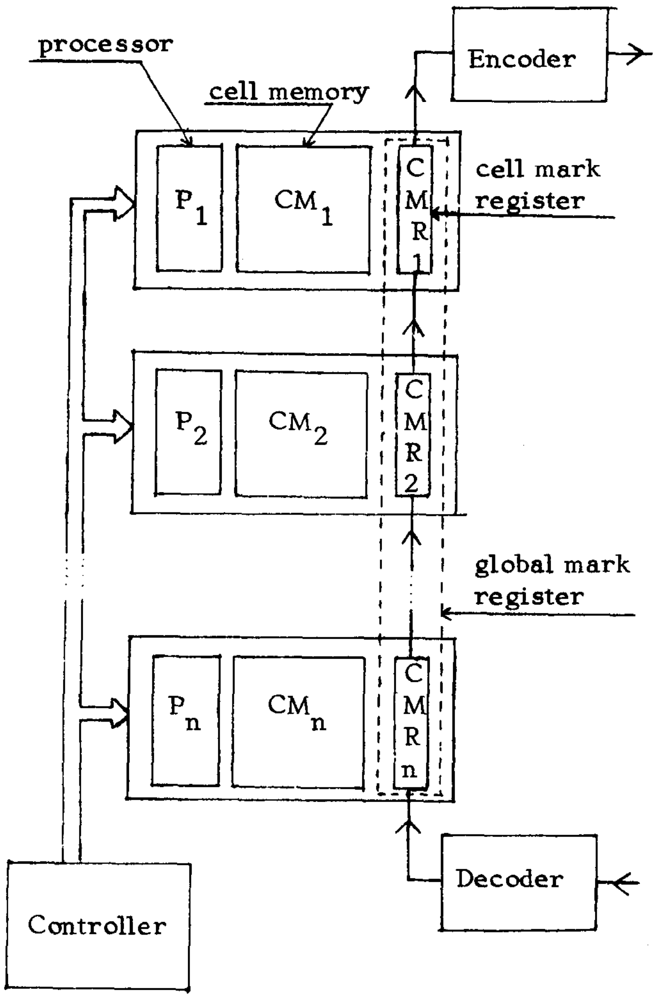


Fig. 3. The modified cellular associative memory.

Each CMR_i has as many bits as tuples stored in CM_i and there is a one-to-one correspondence between the bits and tuples. The CMR_i can be set by P_i or the controller to indicate the tuples relevant for the executed operation.

All CMR's are connected to form the global mark register GMR. Assuming that all CM's are also connected and a tuple occupies one "word" of the memory we obtain the classical model of an associative memory with every word containing one tuple and the GMR as the respond register as shown in Fig. 4.

In the modified cellular associative memory; the GMR is used as a means of communication between the CAM and the external enviroment. To inform about the results of the completed operation the GMR state is sent outside. On the

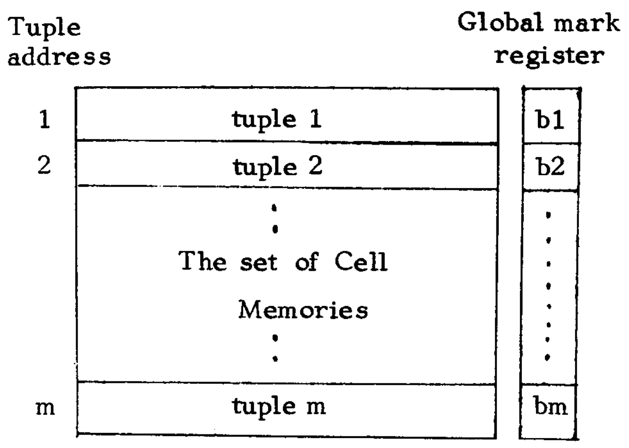


Fig.4. The model of the associative memory.

contrary, to indicate which tuples are to be considered in the next operation the GMR is loaded externally.

The load/unload process is controlled by the Encoder and Decoder units (Fig. 3). Before being sent outside a GMR state is encoded in the Encoder unit. We assume that the encoded GMR state is a list of the bit positions equal 1. The list is just a set of pointers to the marked tuples. Note that physically the list can be stored in a compressed form. On loading, the Decoder converts the set of pointers into the proper GMR state.

Thus the GMR together with the Encoder and Decoder units can be used as an efficient mechanism for pointer processing. It allows to convert a set of marked tuples into a set of pointers and viece versa. Therefore it is a good hardware facility to implement the identifying function of attributes.

Effectively, the GMR mechanism allows to eliminate a great number of selections as well as to decrease the I/O traffic. In the previous CAM's "m" selections have to be generated to mark (identify) a set of tuples knowing their key values (or pointers); where "m" is the number of identifiers. Therefore one operation of loading the GMR saves "m" selections. On the contrary, to inform the controller which tuples were marked their key values or other identifiers have to be sent through the I/O channel. Hence one operation of unloading the GMR saves "m" I/O operations.

It is important to develop a fast load/unload mechanism as the GMR can consists of a great number of bits and is intended to be fequently used. Fortunately, the work of the mechanism can overlap the work of cell processors; if the cell memory is built up with a loop of rotating memory then the desired time for servicing the GMR is as long as one rotation of the loop. We assume that the CAM is provided with at least three copies of the GMR being loaded/unloaded by the same mechanism. During the excution of the current operation, the GMR state set in the previous operation can be sent outside, and the GMR state to be used in the next operation can be loaded. If the loading/unloading process completes before the beginning of the next operation it will cause no delays in the operating of the cell processors, so practically the process costs no time.

There are several methods to develop the GMR mechanism; a simple one is to organise the GMR as a shift register. The shift register can be divided into parts working in parallel in order to make it quicker. More sophisticated techinques are also possible.

4. The proposed DBM architecture

The architecture of the proposed DBM is is shown in Fig. 5. The mass memory consists of the VALUE, IDENTIFICATION, and LINK memories. The VALUE memory is the modified CAM described in Section 3 provided with the global mark register of VALUE memory (GMRV). The LINK memory is also the modified CAM provided with the GMRL. The IDENTIFICATION memory is supposed to be realised in RAM.

The value part of a data base, i.e. the relations R[V], is stored and processed by the cell processors in the VALUE memory, while the link part, i.e. the relations R[L], is stored and processed in the LINK memory. The VALUE memory performs the operations corresponding to the "value" function of attributes, i.e. "value" selections, arithmetic and logical operations, aggregate functions etc. The LINK memory deals with pointers (to the VALUE memory) and its main operation is a selection with an equality comparison being a component of join operation.

Each tuple stored in the VALUE and LINK memory is associated constantly with one bit of the GMRV and GMRL respectively. This assumption allows the global mark register to be the additional and powerful facility to perform the identifying functions.

The identification part of a data base, i.e. the relations R[ID] (set of pointers) and

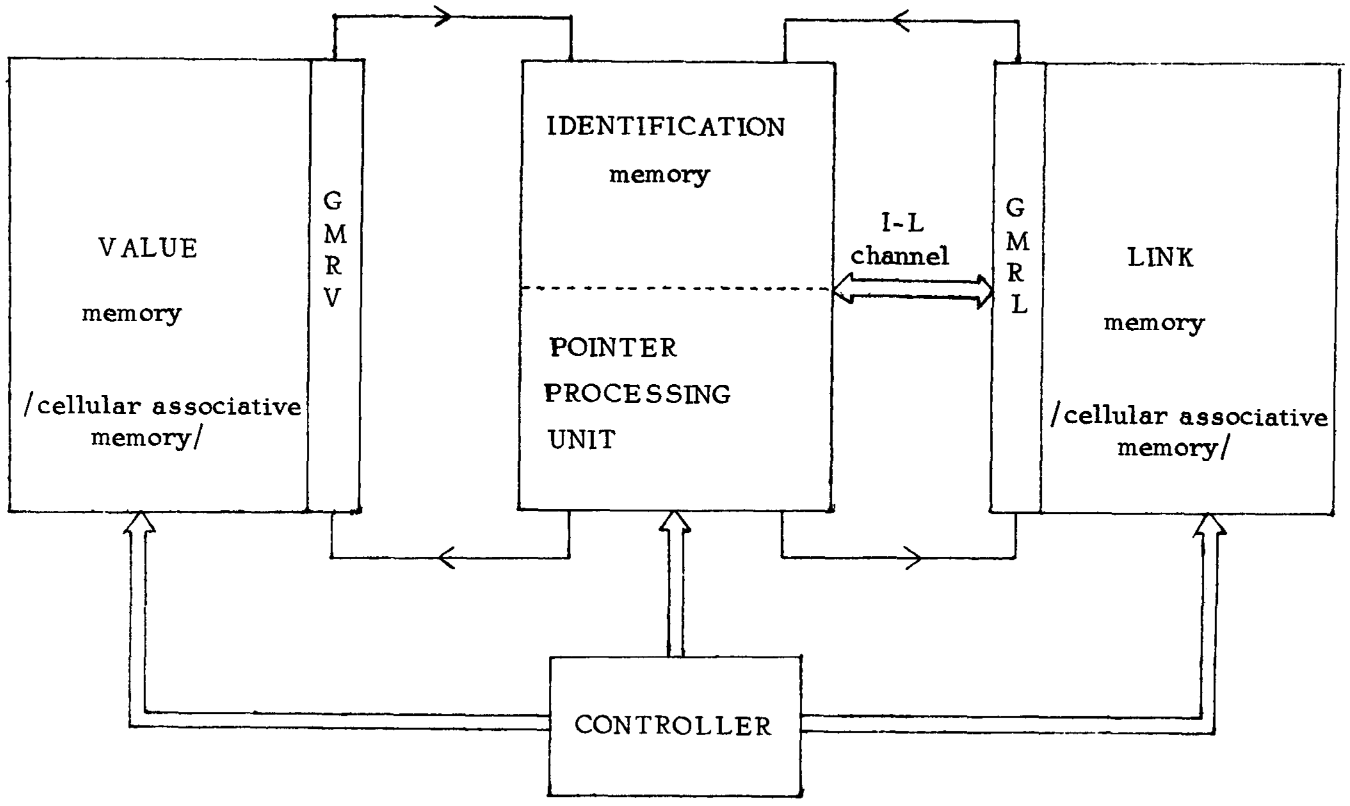


Fig. 5. The proposed database machine architecture.

R^*[ID] (set of attribute names), is stored in the IDENTIFICATION memory and processed by the Pointer Processing Unit (PPU). The PPU is connected with the VALUE and LINK memories so that the flow of information between them is possible. The connections are realised by the global mark register (GMRV and GMRL) loading/unloading mechanisms; precisely, outputs of the Encoders and inputs of the Decoders are adjoned to the PPU. The PPU having the IDENTIFICATION memory as their main memory performs the identifying tasks concerning the relations stored in the VALUE and LINK memory. To identify (mark) a set of tuples a relation R[ID] is sent from the IDENTIFICATION memory to the Decoder of the proper mark register and converted into a state of the register. On the contrary, a state of the GMRV or GMRL can be fatched, coded by the Encoder into a set of pointers and saved as a relation R[ID] in the IDENTIFICATION memory.

The second duty of the PPU is a cooperation with the LINK memory in performing the linking tasks. Therefore the units are additionally connected with the I-L channel. The I-L channel enables the PPU to send arguments of selections to the LINK memory cell processors as well as to transmit data between the IDENTIFICATION and the LINK memories. For example, an attribute of a relation R[L] can be sent to the IDENTIFICATION memory, and next used to identify tuples in the VALUE memory or returned as selection arguments. An example of information flow between the DBM components will be given in Section 6.

Let us show two main advantages of the presented architecture over the previous cellular DBM's.

Simultaneity. Hitherto cellular DBM's consisted of one CAM in which all kinds of tasks were performed sequentially. In our DBM the main components operate simultaneously; at the same time, the VALUE, LINK memory, and PPU independently realise the value, link, and identifying tasks repectively.

Specialization. Hitherto experience with the cellular DBM's proved that the cell processors tend to be complex since they are expected to fulfil many requirments. In the proposed architecture the complexity can be decreased, especially in the case of the LINK memory processors. The LINK memory processors deal with very uniform type of data, i.e. pointers and only the VALUE memory processors handle complex types of data (strings, integers, booleans etc.). The only required operation performed in the LINK memory besides the I/O operations is a selection with an equality test being a component of the join operation. The very short list of performed operations and the uniform data type simplify the LINK memory cell processor and make easier its specialisation. For example, the processor can be adopted to perform selections on a compressed list of arguments rather than on the single one. The complexity of the VALUE memory processors can be decreased because they are relieved from performing a great number of joins, which are performed by the LINK memory, and need not be provided with hardware equipment used to speed up the join.

5. Execution of relational operations in the DBM

We consider execution of a relational operation as a process of creating the dispersed representation of a releation Q being the result of the operation. In proposed DBM the same operation can be executed in a different way depending on its type. The DBM architecture mostly affects the execution of the set operations: the union, difference, and intersection as well as the join operation. We shortly analyse the operations.

Union. $Q = R \cup S$

Difference. $Q = R - S$

Intersection. $Q = R \cap S$

The set operations are expecially easy to perform due to the existence of the global mark registers. All the operations are performed similarly. If the relations R and S are simple then relation R[ID] is sent to the GMRV Decoder to set the register. Next, the GMRV is copied and the relation S[ID] is converted into the GMRV state. After that, the proper logical operation is performed on bits of the current GMRV state and the saved one. After that, the result indicated in the GMRV is sent back to the PPU and converted into the relation Q[ID]. To complete the representation the equivalence $Q^*[ID] = R^*[ID] = S^*[ID]$ is established. In the case the relations R and S are compound the algorithm differs in that the working register is the GMRL.

Join. $Q = R \underset{A\theta B}{\bowtie} S$

The join operation is a bottleneck in relational systems. Generally, execution of join consists of two processes: the derivation and selection. In the derivation process the compared attribute values of one relation are transported from the mass memory to the controller. In the selection process, for each derived value a selection is generated against the second relation. The existing hardware algorithms attempt to optimise the join (mainly the selection process) using such methods as sorting, hashing, and projection. Usually, the algorithms do not classify joins sematicly and execute any kind of joins in the same way.

Our approach to optimise the join execution consists in dividing joins into classes and a specific execution of each class in order to eliminate one or both of the derivation and selection processes.

From the semantic point of view execution of a join creates a new association in the data base (such a join will be called an associating one) or does not. In terms of the dispersed representation the associating join creates a new relation Q[L] in the LINK memory, while the non-associating join creates a relation Q[L] (provided the relation Q is compound) which is a subset of an already existing relation (R[L] or S[L]). This subset can be determined by the relation Q[ID]. The non-associating joins are divided futher into identifying and linking ones. Hence, there are following types of joins.

identifying - natural joins satisfying the conditions $KEYS(R) = KEYS(S)$; where $KEYS(X)$ denotes the set of all foreign keys plus the main key of the relation X.

linking - natural joins such that $KEYS(R) \subset KEYS(S)$. In other words each of the simple relations composing the relation R composes also the relation S.

associating - the rest of joins.

We analyse shortly the realization of the joins. As a part of each realization, a relation $Q^*[ID] = R^*[ID] \cup S^*[ID]$ is created in the IDENTIFICATION memory.

All identifying joins are performed by the PPU together with the GMRV or GMRL. The joins can be divided into two classes.

A - identifying joins such that $R[ID] \subseteq S[ID]$. A join of the class A corresponds to the identifying transformation of the set of pointers (R[ID]) into a set of marked tuples. From the representation formalism, all needed to execute the class A of identifying joins is to establish the equivalence Q[ID] = R[ID].

B - identifying joins such that $R[ID] \notin S[ID]$. The class B is executed similarly to the intersection using the GMRV or GMRL. In the IDENTIFICATION memory, the relations $Q[ID] = R[ID] \cap S[ID]$ is created.

Note that identifying joins require no the derivation process (the arguments are at hand in the IDENTIFICATION memory) nor the selection process which is substituted by the operations of the global mark registers.

The linking joins require the selection process and are performed in the LINK memory with the cooperation of the PPU. We will not describe the algorithm used to perform the join in the LINK memory, which can be one of the existing so far, but, as we mentioned in Section 4, the unified type of data stored and processed in the LINK memory gives an opportunity for futher optimisation. Linking joins are divided into classes C or D according to a type of the relation R.

C - R is simple (S has to be compound). The class C of joins does not require the derivation process. To execute a join of class C, the relation R[ID] is sent from the IDENTIFICATION memory to the LINK memory through the I-L channel. In the LINK memory, the cell processors compare the received values with the attribute $R\uparrow$ of the relation S[L] to mark the matching tuples in the GMRL. Next, the GMRL state is transmitted to the PPU and converted into the relation Q[ID].

D - R is compound. Joins of class D require the derivation process. To execute (in a straightforward way) a join of the class D each attribute of the relation R[L] is sent from the LINK memory to the PPU through the I-L channel. In the PPU the attribute values are sorted to remove duplications and sent back to the LINK memory as arguments of selections in the relation S[ID]. At the end, the tuples of the relation S[L] matching all selections are identified by the relation Q[ID].

The main characteristic of the associating joins is that they create a new relation Q[L] in the LINK memory. The joins can be divided into two classes.

E - equijoins on keys and natural joins on keys such that $KEYS(R) \not\subseteq KEYS(S)$; the relation R is compound.

The class E is performed in the LINK memory similarly to the class D respectively, but the new relation Q[L] is created out of the joinable tuples of the relations R[L] and S[L].

F - all joins which imply other comparison than equality or other attributes than keys. This class has to be performed in the VALUE memory and cause a new relation Q[L] to be created in the LINK memory. The joins require both the derivation (from the VALUE memory) and selections processes.

To summarise, a join is executed by different components of the DBM according to its type and class. The execution of the identifying joins is practically eliminated. The linking joins and the class E of the associating joins are performed in the LINK memory. It is preferable since the VALUE memory is relieved from executing the joins and the LINK memory is considered more efficient. Only the class F of the associating joins is performed in the VALUE memory. The overall improvement of time and data transmission depends on the number of joins performed in each class and is bigger if the number decreases with the advancing of the class's letter. For example, if the percentage of identifying joins is equal 50 then the reduction is at least double.

6. An example of a query execution in the DBM

Let us illustrate the cooperation and flow of information between DBM components with an example of the execution of a typical query. Recall the data base shown in Fig. 1 and its dispersed representation shown in Fig. 2. Let the query be GET INFORMATION ABOUT RED PARTS SUPPLIED BY SUPPLIERS FROM PARIS. The query can be decomposed into the following five operations.

1. $Q1 = \sigma_{CITY="Paris"}(S)$

The selection is performed in the VALUE memory and the GMRV bits in positions 2 and 3 are set to 1. Then the GMRV is transmitted to the PPU and converted into the relation $Q1[ID] = \{2,3\}$; this is the identifying transformation from a set of marked tuples into a set of pointers. In addition, the equivalence $Q1^*[ID] = S^*[ID] =$ {S#, S.NAME, STATUS, CITY} is established.

2. $Q2 = Q1 \bowtie SP$

This is a linking join of the class C. The relation Q1[ID] is sent to the LINK memory where its values are compared with the attribute $S\uparrow$ of the relation SP-C. As the result, the 4, 5, and 6 bits of the GMRL are set. Next the GMRL state is converted into the relation $Q2[ID] = \{L4, L5, L6\}$. Finally, the relation $Q2^*[ID] = Q1^*[ID] \cup SP^*[ID] =$ {S#, S.NAME, STATUS, CITY, P#, QTY, DATE} is created.

3. $Q3 = \pi_{P\#}(Q2)$

This projection is executed in the following way. First, the relation Q2[L] is identified, i.e. the relation Q2[ID] is sent to the GMRL Decoder to mark the proper tuples. If the operation follows immediately after the operation 2 this step is unnecessary since the GMRL is properly set. Next, values of the attribute P↑ are sent from the marked tuples to the PPU throuht the I-L channel. In the PPU the values $\{11, 12, 12\}$ are sorted to remove duplications and form the relation Q3[ID] = $\{11, 12\}$. Finally, the relation $Q3^{*}$[ID] = $\{$ P# $\}$ is created.

4. $Q4 = Q3 \bowtie P$

This is the identifying join of the class A and its execution is reduced to establish the equivalence Q4[ID] = Q3[ID] and create the relation $Q4^{*}$[ID] = $Q3^{*}$[ID] $\cup$ P^{*}[ID] = $\{$P#, P.NAME, COLOR, WEIGHT$\}$.

5. $Q5 = \sigma_{COLOR="Red"}(Q4)$

This selection is executed in the VALUE memory. First, the relation Q4 is identified by loading the GMRV. Next, the selection is performed in the relation Q4[V] and a new GMRV state is determined. After that, the GMRV state is converted into the relation Q5[ID] = $\{11\}$. Finally, the equibalence $Q5^{*}$[ID] = $Q4^{*}$[ID] is established.

To summarise, the execution of the query began in the VALUE memory where the first selection was performed and its results were sent to the PPU using the GMRV. Next, the results (Q1[ID]) were sent to the LINK memory (through the I-L channel) and used as the argument of the linking join. The projection in turn used the I-L channel to send the values of the attribute P↑ of the relation Q2[L] from the LINK memory to the PPU. After properly formatting, the PPU sent the relation Q4[ID] to the VALUE memory, using the GMRV, to assist in the final selection.

7. Conclusion

An approach to implement efficiently a relational data base is presented in the paper. We assume that three kinds of task are performed on data kept in a data base, i.e. identification, link, and value. According to the distinction the dispersed physical representation divides relations into the identification, link, and value parts, and then disperses the parts among three separated devices specialised to store and process their part. The division of a data base depends on the set J of attributes, which is determined by the user and contains the attributes destined to express the associations between relations.

Next, the DBM architecture attempting to implement optimally the dispersed representation is proposed. The DBM uses cellular associative memories to store and process the value and link parts. The CAM's are modified by providing a CAM with the global mark register, which can be quickly loaded and unloaded externally. Tuples stored in the CAM are tied to bits of the GMR allowing the GMR to be a powerful mechanism to perform the identifying tasks. The dispersed representation along with the novel DBM offers the following advantages.

(1) A simple and economic method of storing the derived relations. Due to the efficiency of the marking mechanism the relations can be stored as a set of pointers to the composing tuples.

(2) The simultaneous execution of the three data base tasks and the specialisation of the DBM components. This is illustrated by analysing the execution of join operations. Joins are classified into identifying, linking, and associating types. The cost of the identifying joins is practically negligible.

(3) In addition to the aspects discussed in the paper, the DBM provides some new facilities. The GMR along with its loading mechanism may be used as a hardware tool for realisation such functions as data security and integrity. For example, security can be protected by establishing the relation R′ [ID] defining an accessable part of relation R for a specific access kind. Before accessing relation R, the proper R′ [ID] is converted into a GMR state to mark the allowed tuples.

At the end we want to make the following remark. We assume that an entire data base is kept in the cellular associative memories (the VALUE and LINK memories) which not always can be cost-effective. There are suggestions to use the CAM as a cache memory in the DBM systems. With some modification in controlling the GMRV and GRML the proposed architecture could also serve as the cache.

References

[Baab79] E.Babb, "Implementing a Relational Database by Means of Specialized Hardware", ACM Transactions on Database Systems, Vol. 4, No. 1, March 1979, pp.1-29.

[Bane79] J.Banerjee, D.K.Hsiao, K.Kannan, "DBC - A Database Computer for Very Large Databases", IEEE Transactions on Computers, Vol. C-28, June 1979, pp. 414-429.

[Berr79] P.B.Berra, E.Oliver, "The Role of Associative Array Processors in Data Base Machine Architecture", IEEE Computer, Vol.12, Mach 1979, pp.53-61.

[Bora82] H.Boral, D.J.DeWitt, D.Friedland, N.F.Jarrell, W.K.Wilkinson, "Implementation of the Database Machine DIRECT", IEEE Transactions on Software Engineering, Vol. SE-8, November 1982, pp.533-543.

[Chen76] P.P.Chen, "The Entity-Relationship Model: Toward a Unified View of Data", ACM Transactions on Data Base Systems, Vol,1, March 1976, pp.9-36.

[Codd79] E.F.Codd, "Extending the Database Relational Model to Capture Move Meaning", ACM Transactions on Database Systems, Vol.4, December 1979, pp.397-434.

[Cope73] G.P.Copeland, G.J.Lipovski, S.Y. Su, "The Architecture of CASSM: A Cellular System for Non-numeric Processing", Proceedings 1st Annual Symposium on Computer Architecture, December 1973, pp.121-128.

[DeWi79] D.J.DeWitt, "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems", IEEE Transaction on Computers, Vol.C-28, June 1979, pp. 395-406.

[Hawt82] P.Hawthorn, D.J.DeWitt, "Performance Evaluation of Database Machines", IEEE Transactions on Software Engineering, Vol.SE-8, January 1982, pp.61-75.

[Hsia83] D.K.Hsiao (Editor), "Advanced Database Machine Architecture", Prentice-Hall Inc, 1983.

[Kits83] M.Kitsuregawa, H.Tanaka, T.Motooka, "Application of Hash to Data Base Machine and Its Architecture", New Generation Computing, 1, 1983, pp.63-74.

[Meno81] M.J.Menon, D.K.Hsiao, "Design and Analysis of a Relational Join Operation for VLSI", Proceedings 7th Conference on VLDB, Cannes 1981, pp.44-55.

[Ozka83] E.A.Ozkarahan, "Implementations of the Relational Associative Processor (RAP) and Its System Configurations", Arizona State University, Dep.of Computer Science, TR-82-005.

[Ozka75] E.A.Ozkarahan, S.A.Schuster, K.C.Smith, "RAP - An Associative Processor for Data Base Management", Proceedings of AFIPS NCC, Vol. 44, 1975, pp.379-387.

[Smit79] D.C.P.Smith, J.M.Smith, "Relational Database Machines", IEEE Computer, Vol.12, March 1979, pp.28-38.

[Slot70] D.L.Slotnick, "Logic per Track Devices", Advances in Computers, Academoc Press, 1970, pp.291-296.

[Schu79] S.A.Schuster, H.B.Nguyen, E.A. Ozkarahan, K.C.Smith, "RAP.2 - An Associative Processor for Databases and Its Applications", IEEE Transactions on Computers, Vol.C-28, June 1979, pp.446-458.

[Su79] S.Y.W.Su, "Cellular-Logic Devices: Concepts and its Applications", IEEE Computer, Vol.12, March 1979, pp. 11-25.

[Su79a] S.Y.W.Su, L.H.Nguyen, A.Emam, G.J.Lipovski, "The Architectural Features and Implementation of Multicell CASSM", IEEE Transactions on Computers. Vol.C-28, June 1979, pp.430-445.

[Ullm82] J.D.Ullman, "Principles of Database Systems", Computer Science Press, Inc., Second Edition, 1982.