The Knowledge Engineering Review, Vol. 4: 1, 1989, 1–29

Computer architectures for logic-oriented data/knowledge bases[1]

DONGHOON SHIN* AND P. BRUCE BERRA**

*School of Computer and Information Science, Syracuse University, Syracuse, NY 13244, USA
**Dept of Computer and Electrical Engineering, Syracuse University, Syracuse, NY 13244, USA

Abstract

Knowledge base management systems (KBMS) are designed to efficiently retrieve and manipulate large shared knowledge bases. A significant subclass of KBMS consisting of a combination of logic programming and database is often called a logic oriented knowledge base system (LOKBS). These systems must possess considerable processing and I/O capabilities so many approaches have been taken to the improvement of their performance. In this paper we review the current performance enhancing hardware approaches for LOKBS. We include parallelism, both in processing and I/O, algorithms, caching, and physical data organizations.

1 Introduction

In recent years there has been considerable effort directed toward the integration of many aspects of the artificial intelligence field with the database field. An outcome of this effort is the knowledge base management system (KBMS) which can be described simply as an advanced database system augmented with an inferencing mechanism. Since logic programming is a useful tool in many knowledge-oriented applications, and can be well integrated with existing database management systems, it is the inferencing mechanism in many knowledge base systems. Thus, in this paper, we consider knowledge base systems consisting of Horn clauses called *logic-oriented knowledge base systems.*

One of the distinctive features of the KBMS is the ability to efficiently retrieve and manipulate large, shared knowledge bases. As the management of both general rules and large persistent objects requires an enormous processing load, diverse approaches have been taken in improving the performance of these systems. Exploiting parallelism plays an important role in this context since many knowledge base processing operations are parallel in nature. Also, advances in hardware technology make it feasible to construct specialized computer architectures for time-consuming operations.

The objective of this paper is to investigate various approaches to enhancing the performance of logic-oriented knowledge base systems by means of specialized computer architectures. We begin by introducing four logic oriented knowledge base models: *main memory resident; loosely coupled; tightly coupled;* and *integrated.* In the context of each KB model, we describe basic tasks in order to identify functional requirements, and investigate techniques that have been used to improve the performance of the knowledge processing operations. Then we present a taxonomy for existing and proposed knowledge base machines. We first classify the machines into two large classes: combined KB machines (C-KBM) and integrated KB machines (I-KBM). The machines in the C-KBM class are based on the loosely and tightly coupled KB models, and consist of two components; an inference machine performing resolution and unification in main memory, and a database machine

[1] This work was supported by the Air Force Systems Command, Rome Air Development Center at Griffiss Air Force Base in New York, and the Air Force Office of Scientific Research at Bolling AFB in Washington DC under Contract No. F30602-85-C-008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC)

for managing large amounts of data stored in secondary storage. The machines in the I-KBM class are based on the integrated KB model, and are designed to provide the functionalities required for managing very large rule and fact bases in a uniform way. In the subsequent sections, we describe the architectures of some representative knowledge base machines according to the taxonomy. In the last section, some conclusions are drawn from the survey, and isues of future knowledge base systems are addressed.

2 Computational requirements of knowledge base systems

2.a Processing vs I/O

Performance can be measured in terms of the total time required to complete a user's task. We classify tasks into three types according to the major computational requirements: processing bound, balanced, and I/O bound tasks.

Processing bound tasks consist of operations to repeatedly manipulate a relatively small amount of data residing in the main memory. Examples of processing bound tasks include most numerical processing problems found in mathematics, chemistry, and physics.

In I/O bound tasks, the operations to be performed tend to be simple, but involve large amounts of data in secondary storage. There exists a five to six order of magnitude difference in the time required to access main memory technology versus moving head disk. This access time gap is at the heart of the problem of data intensive applications. Furthermore, as main memory technology is expected to develop faster than that of secondary storage, this gap will become even greater. Thus, increasing the power of processing elements does not help enhance the performance of data intensive applications unless the access time and data rate of secondary storage can be improved.

Knowledge base applications are both processing bound and I/O bound depending on the size of the knowledge bases and the underlying knowledge base system model. For example, finding candidate clauses by unification can be considered a processing bound task if all clauses are resident in main memory. On the other hand, a task can be I/O bound when a large number of clauses are stored in secondary storage.

2.b Four models of logic-oriented knowledge bases

In this section, we introduce models that have been used in realizing logic-oriented knowledge base systems to provide users with functionalities originating from both database systems and logic programming systems. We classify logic-oriented knowledge base systems into four models: main memory resident; loosely coupled; tightly coupled; and integrated.

Most logic programming systems currently available, provide limited functionality for handling large data objects stored in secondary storage, and all clauses must be in main memory before execution. We classify such systems as *memory resident* KB systems. Here tasks are primarily processing bound.

Relational database systems can serve as repositories of information for logic programming systems. More importantly, existing database systems can be exploited through a well defined interface. When the two different systems operate independently, the resulting system has been identified as a *loosely coupled* KB system. This system has the I/O bound tasks inherited from the nature of database processing as well as computation intensive ones from AI processing.

In *tightly coupled* KB systems, each system has extensive information about the other. Thus, there exits more opportunities for optimization by reducing the overhead involved in communication between two independent systems.

In the *integrated* KB model, the redundancy between the two different systems is removed in favour of a total integration of their respective capabilities. In this case, operations lead to a combination of processing bound and I/O bound tasks depending on the size of knowledge bases.

For each of the four knowledge base system models, we discuss basic operations, and present common techniques used to improve the performance of the basic tasks.

3 Main memory resident KB model

3.a Basic tasks

The basic structure that logic programs deal with is a constant, variable or compound *term* defined by a functor with arity n and a sequence of n terms as arguments (i.e. $f(t_1, t_2 \ldots t_n)$). A *program clause* or *rule* is a universally quantified logical formula of the form $A \leftarrow B_1, B_2 \ldots B_n$, where A is a term called the head of the clause, and $B_1, B_2 \ldots B_n$ is the body of the clause. A *unit clause* or a *fact* is a clause without body (i.e. A $\leftarrow$). A *goal clause* or a *query* is a clause of the form $\leftarrow G_1, G_2 \ldots G_n$. Logic programs consist of program clauses, and goal clauses.

Clauses in logic programming are searched and manipulated through *unification* and *resolution*. A *substitution* θ is a finite set of pairs of the form X_i/t_i, where X_i is a variable and t_i is a term, with constraints that $X_i \neq X_j$ if $i \neq j$, and t_i's must not include any X_j's. For any substitution $\theta = \{X_1/t_1, X_2/t_2 \ldots X_k/t_k\}$ and a term s, $s\theta$ represents a term obtained by simultaneously replacing each occurrence of X_i with t_i in s. The main purpose of unification is to find a substitution called the most general unifier σ for a set of terms $\{t_1, t_2 \ldots t_m\}$ so that $t_1\sigma = t_2\sigma = \cdots = t_n\sigma$. The first step of unification is to find a substitution for a *disagreement set*. The disagreement set D of a set of terms S is the set of leftmost subterms consisting of different symbols. For example, the disagreement set D of S = {p(X,b), p(r(d),Y)} is {X, r(d)}, and the substitution for D is {X/r(d)}. If no such substitution exists, a unification failure is recognized. In the above example, the substitution θ = {X/r(d)} is added to the unifier σ, and is applied to S, resulting in a new set of terms $S\sigma$ = {p(r(d), b), p(r(d), Y)}. By repeating this procedure, we can obtain a unifier σ = {X/r(d), Y/b}.

Let $\leftarrow G_1, G_2 \ldots G_n$ be the given goal clause, and assume that the resolution rule selects the leftmost subgoal for generating a new goal. Let $A \leftarrow A_1, A_2 \ldots A_k$ be a *candidate clause* for the subgoal G_1 where A can be unified with G_1. The successful unification operation on A and G_1 generates a unifier σ. Then, G_1 is replaced by $A_1, A_2 \ldots A_k$, the body literals of the candidate clause, and the unifier σ is applied to the new goal $\leftarrow A_1, A_2 \ldots A_k, G_2 \ldots G_n$. When the candidate clause is a unit clause of the form A $\leftarrow$, the new goal becomes $(G_2 \ldots G_n)\sigma$. A result can be obtained when an empty goal is finally reached, which is expressed in terms of a unifier for the variables in the original goal. If a candidate clause cannot lead to an empty goal, a backtracking is required to try a next candidate clause. In this case, the status of computation before the execution of the failed candidate clause should be restored. For problems of significant size the above operations require considerable processing power and memory management capability.

3.b Techniques for improving performance

3.b.1 Compilation and pipelining

The performance of logic programming systems can be considerably improved by using logic programming compilers as opposed to interpreters. Dorby, Despain and Patt (1985) indicated that an efficient compiler can gain a factor of 20 over interpreters. Warren proposed a Prolog instruction set called the *Warren abstract machine* (WAM) to provide the basis for constructing Prolog compilers (1983). Many Prolog machines are designed based on WAM, exploring the pipelined execution of compiled code with special registers for passing variable binding information among goals.

Another important technique used in WAM is the clause indexing scheme to find candidate clauses without exhaustive searching. Clauses with the same predicate name are divided into several groups according to the types of the first arguments (i.e. variable, constant, list, structured term). Then, the selection of a candidate clause in a group is performed according to the hashed value of the

first argument. This type of indexing has proven to be very useful for small logic programs residing in main memory.

3.b.2 Parallelism

Parallel evaluation of logic programming has been an important research issue in the context of the memory resident KB model. Conery (1987) classified the inherent parallelism in logic programming into three major categories; low-level parallelism, OR-parallelism and AND-parallelism. Low-level parallelism can be obtained by overlapping primitive operations so that the execution time can be reduced; OR-parallelism concerns trying all candidate clauses in parallel; AND-parallelism, on the other hand, concerns parallel evaluation of subgoals by replacing the strict left to right evaluation order of Prolog. Exploiting parallelism in unification has also been studied (Harland and Jaffar, 1987; Vitter and Simons, 1986; Yasuura, 1984).

However, parallel execution in the memory resident KB model does not gain much speed up (Fagin and Despain, 1987), since generally only a small number of candidate clauses exist, and the indexing scheme used in WAM considerably reduces the time required to identify candidate clauses. Dwork, Kanellakis and Mitchell (1984) also indicated that, since unification is inherently sequential, parallel evaluation of a unification algorithm may not offer much speed-up over sequential evaluation.

4 Loosely coupled and tightly coupled KB models

4.a Basic tasks

A loosely coupled or a tightly coupled KB system has an inferencing system in the front end which communicates with an independent database system in the back end. Thus, the tasks of the front end are those of the memory resident KB model, which are supported by either general purpose or specialized computer systems. One of the issues involved in coupling a logic programming system and a database system is to resolve the mismatch between the tuple oriented access of the logic programming system and the set oriented retrieval of the database system. From the performance point of view, supporting partial match queries, join and other operations has been one of the major concerns in the back end (Gonzalez-Rubio et al., 1987; Sakai et al., 1986) since these operations are expected to occur more frequently than in conventional database systems. Consider, for example, the query ← p(a,X),q(c,b,c), and suppose that p and q are base relations (facts). Processing the query involves searching relation p based on the first attribute, q on the second and third attributes followed by an equi-join and a projection.

Recursively defined rules require a more complicated operation called the *least fixed point* (LFP) operation (Aho and Ullman, 1979; Sakai et al., 1986). When processing a query involves recursively defined subgoals, the query cannot be translated into an explicit sequence of relational operations as in the above example, since the termination point cannot be recognized in advance. The LFP operation is basically performed by repeated relational operations such as selection, projection, join and a set union operation with the previous results to detect the termination point. Consider, for example, the following recursive rules and facts

```
r(X,Y) ← p(X,Y).
r(X,Y) ← p(X,Z), r(Z,Y).

p(a, a1). p(a, a2).
p(a1,b1). p(a1, b2). p(a1, b3). p(a2, b1).
p(b1,c1). p(b2, c2).
p(c1,d1). p(c2, d2).
```

Figure 1 shows the sequence of intermediate results of the LFP operation for the goal ← r(a,X). In

Figure 1 Example of LFP Operation

i	R_Δ	R_i	$R_\Delta \cup R_i$	t_i
0	ϕ	ϕ	ϕ	F
1	a1,a2	ϕ	a1,a2	F
2	b1,b2,b3	a1,a2	a1,a2,b1,b2,b3	F
3	c1,c2	a1,a2,b1,b2,b3	a1,a2,b1,b2,b3,c1,c2	F
4	d1,d2	a1,a2,b1,b2,b3,c1,c2	a1,a2,b1,b2,b3,c1,c2,d1,d2	F
5	ϕ	a1,a2,b1,b2,b3,c1,c2,d1,d2	a1,a2,b1,b2,b3,c1,c2,d1,d2	T

Figure 1, R_Δ and R_i represent the newly generated results and the results obtained by all previous steps respectively. If $R_{i+1} = R_\Delta \cup R_i$, then t_i is set to terminate the LFP operation.

4.b Techniques for improving performance

4.b.1 Parallelism

Parallelism can greatly improve the performance of I/O bound tasks. Current magnetic disk technology is such that it requires milliseconds of access time and the rate from an individual disk is limited to about 3 Mb/s. The use of multiple independent disk systems and logic per track devices have been proposed for many database machine architectures to increase the data rate from secondary storage. Some recent commercial computer systems exploit parallel disk interleaving techniques where related data are stored across different disk units (Kim, 1986; Thinking Machines Technical Report, 1987). As small disks become more reliable, cheaper, and faster, it becomes technically feasible to incorporate a large number of disks in a system thus obtaining a much higher data transfer rate.

Parallel database machines are in need of new algorithms that optimally map the problem into the architecture. In this context, a great deal of effort has also been placed on developing efficient parallel algorithms for relational operations. For example, DeWitt and Gerber have observed that hash join algorithms can be more effective than nested loop or sort-merge algorithms in a parallel processing environment (DeWitt and Gerber, 1985).

Parallel algorithms for relational operations generally require a larger data cache in order to effectively provide a large number of processors with the required data (Berra and Oliver, 1979) or to store intermediate results for further processing. For simple operations, each processor can take advantage of two buffers to overlap processing and I/O. That is, data in one buffer can be processed while the other is being filled with the next block. Thus, large data caches are important components of many database machines.

4.b.2 Physical data organization

An effective indexing scheme can outperform the use of brute force parallelism (Stone, 1987) since only a small portion of the database generally needs to be accessed for a given query. Some database machines have adopted new physical data organization schemes to improve the performance of partial match retrieval and join operations. For partial match queries, data should be organized in such a way that any combination of attribute values specified in queries does not lead to an exhaustive scan of the entire database. On the other hand, join operations require that data be clustered based on the join attributes. In the remainder of this section, we consider a fully decomposed storage organization and a grid file scheme that have been used in database machines for efficient partial match retrieval and join operations.

The physical data organization scheme of DELTA (Sakai et al., 1986) is a fully decomposed storage oganization, where a relation with N arguments is divided into N binary relations with a

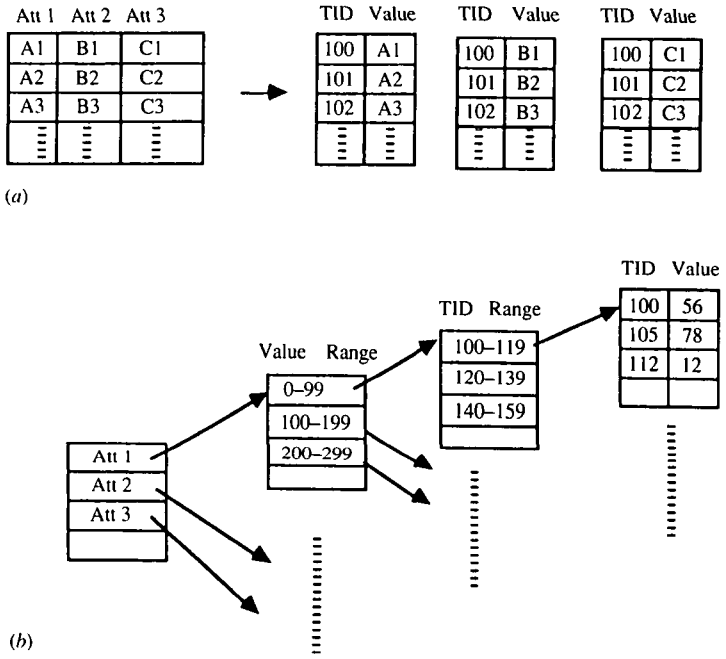


Figure 2 Physical Data Base Organization of DELTA (a) Attribute-Based Configuration. (b) Two-Level Clustering Method (Sakai et al., 1986)

tuple identifier (TID) as the first attribute for all N binary relations (see Figure 2(a)). This organization is well suited to join operations and partial match retrieval queries, especially for those queries having a small number of arguments (attributes) specified. To reduce the search space further, the data is clustered according to the TID numbers as well as the values. This two-level clustering method is shown in Figure 2(b).

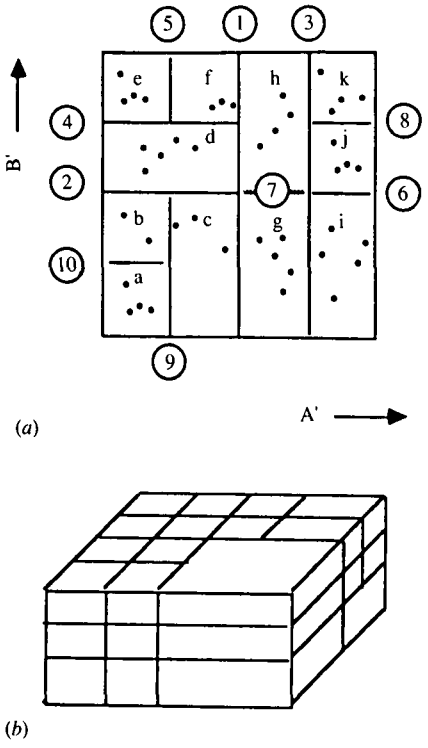


Figure 3 Grid File Based on Data Space Partitioning. (a) Two Dimensional Data Space. (b) Three Dimensional Data Space.

The Massively Parallel Database Computer (MPDC) (Tanaka, 1986) uses the grid file (Nievergelt et al., 1984) as the basic physical storage structure to partition a large data space into manageable small buckets. Actually, a page in the page buffer corresponds to a bucket in the grid file. In the grid file, tuples with n attributes are represented as points of n dimensional hyper space $(D_1, D_2, \ldots, D_n)$ where D_i is the domain of the ith attribute. The hyper space is partitioned according to the data distribution. Figure 3 shows the partition of the data space for two dimensional data (a) and three dimensional data (b).

Suppose, for example, that we have a relation R(A,B) and the corresponding encoded relation R'(A',B'). Possible pairs of A' and B' values are plotted in the two dimensional space as shown in Figure 3(a). The numbers in the circles represent the order of data space partitioning, and each bucket can hold 5 records. Consider a query R'(010010, 110011). For this query, only bucket f needs to be accessed. Since each node can be represented by a small amount of data, the directory itself can be resident within the main memory in most existing computer systems. In general, for queries specifying a large portion of the attribute values, a small number of disk accesses are required. It has also been reported that this organization can provide high performance in join operations as well (Ozkarahan and Bozsahin, 1988).

5 Integrated KB model

5.a Basic tasks

The Integration of an inferencing system with a database system can be done in many ways. One way is to introduce extensive indexing capabilities to a logic programming system to allow efficient accesses to persistent data objects stored in secondary storage. Another is to extend database models so that general clauses can be stored and managed in a uniform way.

The processing requirements of integrated KB systems depend on assumptions made on the structures of underlying database models and the query evaluation strategy on the databases. The knowledge base can have very complex structures allowing general terms or clauses without any constraints in structure. The structure of first order terms provides the basis for modeling real world information beyond the functionalities of traditional database models such as the relational, hierarchical, and network models. The data model of logic oriented knowledge bases actually subsumes more recent models such as the non-first normal form (NFNF) model (Dadam et al., 1986) which has been proposed for the efficient management of complex objects. Both top-down and bottom-up query evaluation strategies can be used to obtain one solution at a time and all possible solutions at once respectively. When the knowledge-based system is based on complex, persistent objects, extended relational algebra and retrieval by unification have been considered.

5.a.1 Extended relational algebra

Zaniolo (Zaniolo, 1985) proposed a set of relational algebra operations for logic oriented knowledge bases called *Extended Relational Algebra* (ERA). ERA includes such operations as *extended select*, *extended project*, and *combine* as well as conventional relational operations such as set union and join. Only complex facts without logical variables are considered in ERA. Therefore, unification-based operations are not supported.

An addressing scheme is required for ERA operators to specify the attributes participating in evaluation. ERA uses Dewey's decimal notation as the addressing scheme for arbitrary complex facts. Consider, for instance, the term p(f(a, l(a,r(c))),h(a)). Here, addresses 1.0 denotes f(a, l(a, r(c))), 1.1 denotes a, and 1.2 denotes l(a,r(c)). The arity of a functor can also be specified by the special notation *count*.

An extended select operation is of the form

$$\sigma[adr_1\langle op_1\rangle value_1, \ldots, adr_k\langle op_k\rangle value_k],$$

where adr_i is an address, $value_i$ is an address or a ground term, and $\langle op_i\rangle$ is one of the relational

operators, $=$, $\neq$, $>$, $<$, $\leqslant$, $\geqslant$. Similarly, an extended project operation is specified by

$$\pi[adr_1, \ldots, adr_k].$$

The two operations can be combined by an Extended Select/Project (ESP) operation defined as

$$\rho[adr_1\langle op_1\rangle term_1, \ldots, adr_k\langle op_k\rangle term_k/adr_{k+1}, \ldots, adr_n]$$

denoting

$$\sigma[adr_1\langle op_1\rangle term_1, \ldots, adr_k\langle op_k\rangle term_k]$$

with

$$\pi[adr_{k+1}, \ldots, adr_n].$$

For example, the query ← p(q(a), r(Y,d)) can be defined as an ESP operation on base relation p by

$$\rho[1.0 = q,q.count = 1,1.1 = a,2.0 = r,2.count = 2,2.2 = d/2.1].$$

The combine operator combines N objects of the relation into one object and denotes it by γ.

5.a.2 Retrieval by unification

Yokota and Itoh (1986) proposed a *Relational Knowledge Base Model* based on *term relations* which can accommodate unrestricted Horn clauses including both general rules and facts. The basic operation proposed by Yokota and Itoh is called *retrieval by unification* (RBU). It consists of unification-restriction which is based on unification on the data stream, and unification-join which is actually a parallelized, top-down resolution procedure. The first step of retrieval by unification is to find candidate clauses in a term relation for a given goal. This is done by the unification-restriction operation. Consider, for example, the term relation T given in Figure 4(a), and the goal ← ancestor(smith, A). The unification-restriction operation is invoked for the term relation T by the condition

$$T(Head) \diamond [ancestor(smith,A)]$$

and makes a new term relation T_1 (Figure 4(b)). Here the symbol $\diamond$ denotes unification. The second attribute of T_1 contains the new goals generated by the unification restriction operation, that is, ← parent(smith,A) and ← parent(smith,Z),ancestor(Z,A). Then, all the leftmost subgoals, ← parent(smith,A) and ← parent(smith,Z), are evaluated in parallel against the original term relation T (unification join), producing the new term relation T_2 shown in Figure 4(c).

The tuples whose second attributes are empty ([ ]) represent the results of the query. This procedure continues for the remaining tuples having non-empty bodies until no more new results can be generated.

The relational knowledge base model provides the functionalities required for managing very large rules and facts in a uniform way. However, the operations incur expensive I/O bound tasks, especially when all possible solutions must be generated.

5.b Techniques for improving performance

5.b.1 Parallelism and memory management

Efficiency in staging qualified clauses from secondary storage into a processing level is one of the major concerns in integrated knowledge base system. Some specialized hardware components such as data filters perform preliminary unification steps on the data stream from secondary storage (Sabbatel and Dang, 1987) thus reducing the amount of data to be placed in the main memory for further processing. But a more general approach to providing the necessary data rate is based on the use of multiple disc systems with a page oriented large shared buffer memory between processing

T

Head	Body
[ancestor(X,Y)]	[parent(X,Y)]
[ancestor(X,Y)]	[parent(X,Z), ancestor(Z,Y)]
[parent(smith, clark)]	[]
[parent(clark, tuner)]	[]
:	:

T_1

Head	Body
[ancestor(smith,A)]	[parent(smith,A)]
[ancestor(smith,A)]	[parent(smith,Z), ancestor(Z,A)]

T_2

Head (T_1)	Body (T)
[ancestor(smith, clark)]	[]
[ancestor(smith,A)]	[ancestor(clark, A)]

Figure 4 An example of retrieval-by-unification. (a) An example of a term relation. (b) The result of unification-restriction. (c) The result of unification join (Yokota and Itoh, 1986)

elements and the disc. A contention free multi-port page memory (MPPM) has been explored as a central component of various knowledge-based machines developed in the Fifth Generation Computer System project (Itoh et al., 1988).

Another important issue involved in parallel processing is the management of a large number of subprocesses generated by parallel, top-down evaluation. In parallel logic programming evaluation, some processes are independent of each other (i.e. OR processes), while some must closely interact with others (i.e. AND processes). The efficient management of memory space is one of the most critical factors in this context. In addition, these subprocesses frequently need to access complex objects stored in secondary storage. MANIP (Wah et al., 1989) supports a virtual memory scheme to control the memory space needed for managing subprocesses in parallel.

However, even with parallel disk systems and very large buffer memories, knowledge base machines should be provided with efficient access methods since reading the entire knowledge base searching for qualified clauses will incur an enormous overhead. In the next section we describe some physical data organization schemes proposed for complex knowledge bases.

5.b.2 Physical data organization

Data structures of general terms and clauses can be arbitrarily complex. Indexing schemes for these data objects must provide flexible access paths required for operations such as ERA and RBU.

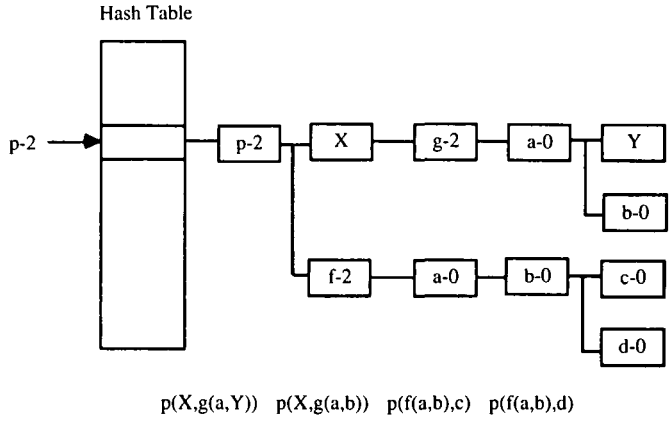


Figure 5 A Physical KB Organization Scheme Based on Hashing and Trie (Itoh et al., 1988)

Physical data organization schemes based on transformed hash coding such as superimposed code words (SCW) or concatenated code words (CCW) have been widely investigated with the objective of enhancing retrieval from very large knowledge bases (Berra et al., 1987; Ramamohanarao and Shepherd, 1986; Shin and Berra, 1989; Wada et al., 1987). In these schemes, compact images of the knowledge bases are first generated by transformed binary codes. The transform is performed by well chosen hashing functions which produce codewords of a fixed length. Then, structural information and unique identifiers are attached to the codewords. Due to the compact and uniform structure of the codewords, parallelism can be effectively utilized, and I/O time can be reduced.

A knowledge base indexing scheme based on tries and hashing has been proposed for the retrieval by unification operation as shown in Figure 5 (Itoh et al., 1988). Complex terms are stored in such a way that terms with similar structures share the common left-most subterms with each other so that the number of comparisons required for unification can be reduced.

6 Taxonomy of logic-oriented D/KB machines

The major objectives of logic-oriented knowledge base machines are to achieve high performance in two important tasks which can significantly influence overall system performance; inferencing on a large number of rules and management of a very large data space consisting of facts stored on secondary storage. In this section, we classify logic-oriented data/knowledge base machines into two large classes depending on how "rules" and "facts" are managed in main memory and secondary storage:[1]

1 *"combined" knowledge base machines* (C-KBM) consisting of a number of inference machines to efficiently manage the clauses stored in the main memory performing resolution and unification, and a number of database machines for managing very large facts stored in secondary storage.
2 *"integrated" knowledge base machines* (I-KBM) with the capability of uniform management of very large amounts of general clauses stored in secondary storage.

The inference machine component of the C-KBM class can be viewed as a front-end of a database machine component, which is mainly responsible for the unification operation or for the efficient management of memory space to maintain information about variable bindings for unification. The inference machines can be further classified into two subtypes depending on the major task.

[1] Different classification schemes for broader surveys on AI machines have been suggested; in Hwang et al. (1987), AI machines are classified into three major categories, namely language-based machines, knowledge-based machines, and intelligent interface machines. On the other hand, Treleaven et al. (1986) classifies computer architectures into 6 major categories including two AI-related categories; logic computers and knowledge-based computers. The scope of this paper includes those AI-machines based on logic and database machines in the logic paradigm. The taxonomies of database machines are presented in (Boral and Redfield, 1985; Qadah, 1985).

1 *unification machines* designed to speed up unification for logic programming interpreters. Most of them are designed as *unification co-processors* which interact with a host by sharing memories or by interconnection networks.
2. *resolution machines* designed for performing resolution as well as unification. Fast searching for candidate clauses based on unification is the major task of the machines in this class. Some machines perform operations on clauses, while others are based on compiled codes tailored to logic programming. Since the compiler performance is far greater than that of interpreters, an order ot magnitude increase in speed can be achieved by exploiting computer architectures tailored to the instruction set.

The database machine component of the C-KBM class provides the knowledge base system with efficient access paths to objects stored on secondary storage, and supports diverse database operations such as join. In the context of knowledge-based systems, we can consider two types of database machines.

1 *deductive database machines* mainly concerned with how to deal with rules in the relational database environment. Queries are translated by a sequence of relational operations by a front end, and then a deductive database machine performs the operations to answer the queries bottom-up. The Least Fixed Point (LFP) operation is considered in most deductive database machines to support recursive rules.
2 *database machines for unnormalized relations* designed to support non-relational models such as the hierarchical database model. As general terms and Horn clauses cannot be represented by the relational model, these machines can be used in dealing with general terms.

The knowledge-based machines in the second class, I-KBM, integrate data manipulation and inferencing into a single system. From the software point of view, this approach includes augmenting logic programming systems with a secondary storage access capability, and extending underlying data models to accommodate rules and general terms. We classified the I-KBM's into three categories:

1 *unification filters* which function similarly to database filters. They can perform unification on data streams from a large file store.
2 *massively parallel machines* which can offer the high performance required for integrated knowledge-based machines. In order to enhance the performance of I/O bound tasks, I/O systems with high data rates are important components of these systems.
3 *machines with intelligent storage management* especially designed for logic-oriented knowledge bases. The principal enhancement of this approach over the combined approach stems from the capability of managing objects residing in secondary storage by intelligent storage management such as virtual memory and/or shared page buffers. Complex objects (terms) can be retrieved based on unification/term matching, and top-down evaluation (backward-chaining) of clauses is used.

In the following sections, we present some representative knowledge base machines for each class.

7 Inference machines

7.a Unification machines

Unification machines directly implement various unification algorithms by using hardware or microprogramming to enhance the performance of logic programming interpreters. The unification machines generally work with general purpose hosts which send terms to be unified to the unification machine. We can further classify them into two types: unification co-processors and unification machines based on stream processing. This section describes the architectures of various unification machines.

7.1.a Unification co-processors

Having found that most execution time of the University of New South Wales (UNSW) Prolog interpreter is taken by unification operations, Woo proposed a Hardware Unification Unit (HUU) as a back-end server to the VAX11/780 to speed up the unification operations (Woo, 1985a,b).

Although the HUU can improve the performance of unification up to 100 times faster than the software "unify" function of the UNSW Prolog, the HUU may not outperform a Prolog compiler in terms of the overall performance. Even if we assume that the "unify" function takes 70% of the total execution time and the HUU can improve the performance of unification operations by a factor of 100, the HUU can only achieve 3.3, while compilation can enhance the performance by a factor of 20. The performance analysis of the Syracuse Unification Machine (SUM) showed a similar result (improvement by a factor of 2) (Robinson, 1985).

The SUM co-processor was originally envisioned as a single chip (Figure 6). One of the major differences between SUM and HUU is that SUM needs to access the memory of the host for processing data such as lists, while HUU deals with such structure intputs by itself. Therefore, the SUM co-processor was designed to interact with the host in a very loosely coupled fashion. The host sends terms to be unified to SUM without doing any analysis on the types of the terms, and SUM requests input data stored in the host when it is ready. It has been observed that a great deal of time is involved in communications between SUM and the host.

The associative processor (AP) designed by Stormon (Stormon, 1986) provides various enhancements for improving performance over the SUM. In AP, to decrease the communication overhead, the host analyzes the terms to be unified into a number of subtasks, handles some obvious cases such as the comparison of two constants, and then sends the only remaining cases (i.e. subtasks involving variables) to the co-processor. While SUM exploits two associative memories for binding agents primarily to prevent common variables from being bound to different values, the AP unification machine exploits content addressability in maintaining the stack of substitutions for variables which can be used for the case of backtracking as well. The maximum speed that can be achieved by the unification processor is predicted up to 500K LIPS (Logical Inferences Per Second).

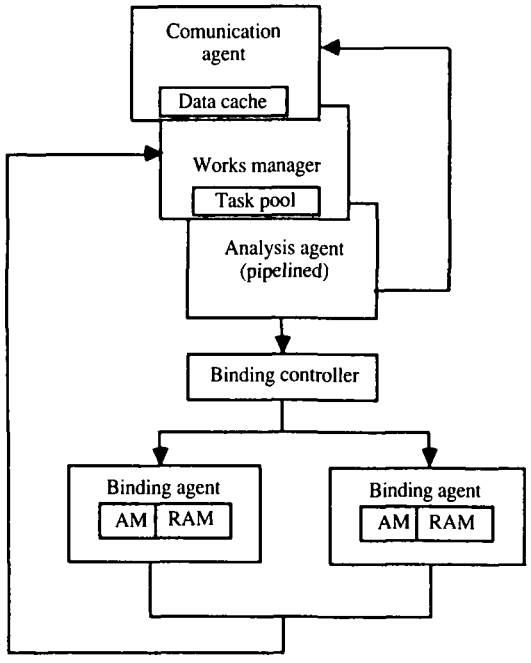


Figure 6 Syracuse Unification Machine (SUM) (Robinson, 1985)

7.a.2 Stream unification machines

Shobatake and Aiso (1986) proposed a systolic-like method for implementing a VLSI-oriented unification processor by using a linear organization of processing elements (cells) aimed at eliminating frequent data transfers between the host and the unification processor. A cell consists of two identical pairs of registers; a bound variable register (BVR) and a symbol register (SYR). The SYR contains a functor with its arity, and the BVR is designed to keep the variable after unification is complete. It plays a similar role to that of the trail stack in Prolog interpreters. Shobatake and Aiso's machine performs unification on two input terms represented as streams of tokens. An input term to the cell array is composed of a beginning mark (TS), the functor name with arity followed by arguments also represented by functor name with arity number, and finally, an ending mark (TE). This representation allows for the storage of arbitrarily complex terms. The other term to be unified with the given term is represented in reverse order. The arities of variables and constants are assigned the value zero. For example, a term f(X, a(b, Y)) can be represented as

(TS)(f,2)(X)(a,2)(b,0)(Y)(TE)

and the other term f(c,Z) to be unified can be represented as

(TE)(Z)(c,0)(f,2)(TS)

Tanabe and Aiso (1987) proposed the use of a structure sharing scheme, and designed a pipelined unification processor (PUP) for stream processing to solve this problem. The input terms to the PUP are also represented as the linearized strings as shown above. One of the major improvements of the PUP over the previous machine is that consistency checks on variable bindings are separated from comparisons among constants by using a pipelining technique. The PUP consists of 4 major components (Figure 7):

1 *STBV (Select Term Binding Variable) processor* which performs a comparison among constants to be unified (a preunification step). In this step, all the variables appearing in the terms are regarded as don't care match indicators.
2 *AVTC (Assign Variable To CTBV) processor* which finds binding conflicts for shared variables.
3 *CTBV (Charge Term Binding Variable) processors* which manage the actual bindings for variables. There are a number of CTBV's each of which contains a variable binding. After a successful unification, the most general unifiers (mgu's) will be stored in the CTBV's.
4 *CCTBV (Control CTBV) processor* which controls the CTBV and AVTC.

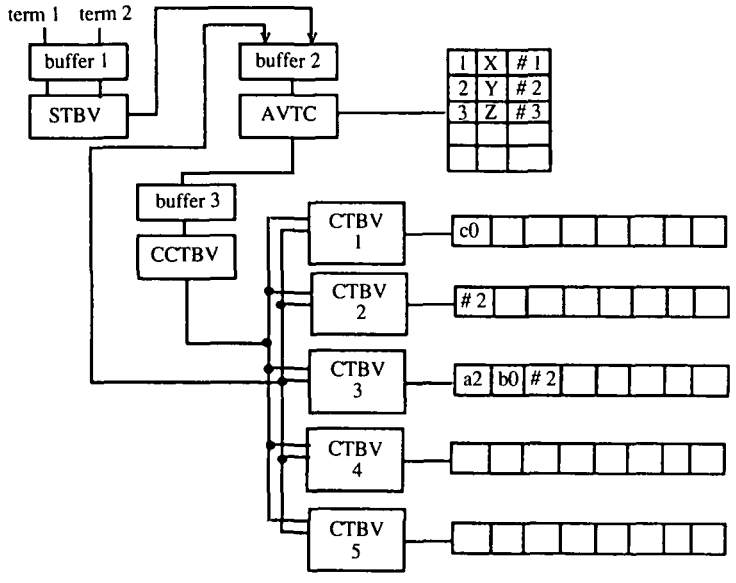


Figure 7 Pipelined Unification Processor (PUP) (Tanabe and Aiso, 1987)

Figure 7 shows the contents of the STBV processor and CTBV processors after the unification of two input terms, f(X, a(b,Y)) and f(c,Z). The first CTBV processor contains binding information for the variable X (i.e. {X/c}), while the second and the third CTBV processors manage variable bindings for Y and Z respectively.

7.b Resolution machines

We consider two types of resolution machines in the context of the main memory resident KB model; associative processors and Prolog machines. The associative processors exploit content addressability in some resolution steps such as finding candidate clauses or maintaining variable bindings. Prolog machines are designed to optimize the performance in executing compiled logic programs by exploiting fast registers and pipelining techniques. In this section, we consider associative processors HAS (Hahne et al., 1985) and ASSIP-T (Schneider and Dilger, 1986), and high performance Prolog machines PLM (Dorby et al., 1984, 1985) and IPP (Abe et al., 1987).

7.b.1 Associative processors for resolution

The Hybrid Associative Store (HAS) consists of 64 processing elements each of which is associated with a large number of memory cells (Hahne et al., 1985). The resulting architecture can be viewed as a quasi-associative processor, since all the PE's are synchronously controlled, that is, all the PE's simultaneously access the same locations in memory. Each PE has an ALU, a number of general purpose registers, and a mask register.

To illustrate the management of rules in HAS, consider, for example, the following clauses:

1 p(X, Y) ← q(X, Y), r(Y).
2 p(a, b).
3 q(c, d).
4 q(d, f).
5 r(X) ← s(X, X).
6 r(b).

Figure 8 shows the contents of memory for the above logic program. Each PE stores a term and an index associated with the term. An index is of the form < clause number, level number > where the clause number uniquely identifies a clause, and the level number indicates the position of the term within the clause (i.e. head = 0, first subgoal = 1, second subgoal = 2, ...). Suppose that a goal ← p(a, Z) is given. The evaluation process is as follows:

1 Find terms whose index is of the form $< *, 0 >$ (i.e. heads of clauses). Then, unify the terms with the given goal ← p(a,Z). In our example, the terms associated with $\langle 1, 0\rangle$ and $\langle 2, 0\rangle$ are found. Substitutions $\theta_1 = X/a, Y/Z$ and $\theta_2 = X/a, Y/b$ are obtained.
2 Find terms starting with $\langle 1, 1\rangle$ and $\langle 2, 1\rangle$. In our example, there does not exist any term

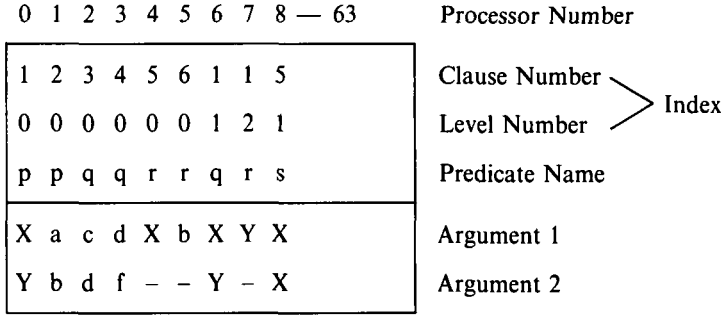


Figure 8 Data Organization of HAS (Hahne et al., 1985).

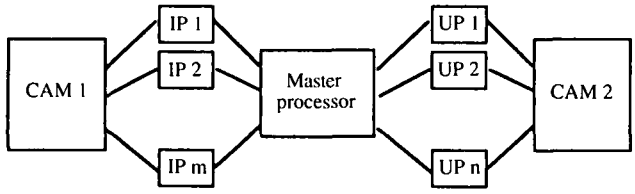


Figure 9 ASSIP-T (Schneider and Dilger, 1986)

corresponding to ⟨1, 1⟩, and thus θ_1 is considered as a result. The term q(X, Y) is found for ⟨2, 1⟩, and the substitution θ_1 is applied to the term producing the subgoal ← q(a,Z).

3 The above procedures continue until all the subgoals are resolved.

Although associative searching can be very effective for parallel evaluation of logic programs, the size limitation of associative memories makes it difficult to use them for arbitrarily complex objects.

Another computer, ASSIP-T, consists of a master processor, two associative memories (CAM1 and CAM2), and multiple inference and unification processors (Schneider and Dilger, 1986) (Figure 9). The main principle of ASSIP-T is to separate unification operations from resolution by maintaining two graphs called a *deduction plan* and *a unification graph with constraints* (UwC). Nodes of a deduction plan denote goals, and edges are classified into two types, SUB and RED, according to the type of inference performed on the resolution step. Associated with each edge of the deduction plan is a set of constraints of the form $\{(t_1, s_1, \{n\}), \ldots, (t_m, s_m, \{n\})\}$, assuming that the starting edge is of the form $p(t_1, \ldots, t_m)$, the ending edge is $p(s_1, \ldots, s_m)$, and the edge number (i.e. resolution step) is n. The corresponding UwC is constructed by forming a labeled undirected graph from the constraint. That is, for the constraint (t, s, d), the nodes t and s are connected by an edge labeled d. Unification failure is detected when a variable node in the UwC leads to different constant values. When a unification failure occurs, a minimial conflict set of edges are found and by removing one of the edges in the minimal conflict set, a successful unification can be obtained.

The master processor gets the constraints from the inference processors and distributes them to the unification processors where the conflict sets are calculated. The CAM2 stores each term for the UwC with term names, labels, and the adjacent node in the UwC.

7.b.2 Prolog machines based on WAM

Prolog machines are designed to enhance the performance of both sequential and parallel logic programming languages. Parallel Prolog machines also use parallelized WAM instructions to incorporate AND/OR parallelism. Table 1 shows some Prolog machines proposed so far.

A representative Prolog machine based on the WAM is the Programmed Logic Machine (PLM) (Dorby et al., 1984, 1985). PLM exploits small scale parallelism and instruction pipelining with

Table 1 Prolog Machines

Machine	*Institution*	*KLIPS*	*Characteristics*
PLM (Dorby et al., 1984, 1985)	UC Berkley	425	Sequential, WAM
PPP (Fagin and Despain, 1987)	UC Berkley	N/A	Parallel extension of PLM and WAM
HPM (Nakazaki et al., 1985)	NEC	280	Sequential, WAM
IPP (Abe et al., 1987)	Hitachi	1000	Sequential, WAM Special clause indexing
PSI (Fuchi and Furukawa, 1986)	ICOT	30	Sequential Interpreter (Structure Sharing)
PSI-II (Fuchi and Furukawa, 1986)	ICOT	100	Sequential, WAM
ICM3 (Noye and Syre, 1987)	ECRC	530	WAM, Co-processor

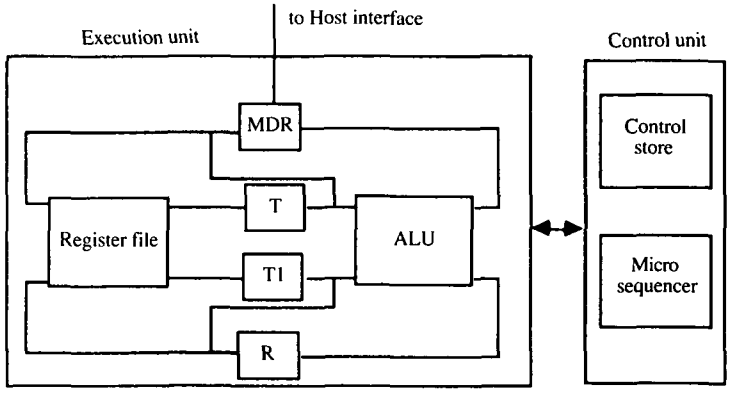


Figure 10 Programmed Logic Machine (PLM) (Dorby et al., 1985)

special registers and a microcoded instruction set. PLM consists of two main units as seen in Figure 10; the control unit and the execution unit. The control unit consists of a control store containing microcodes and a microsequencer for executing the microcodes, and is primarily responsible for directing the actions of the execution unit as well as for communicating with the host interface using buffers. The execution unit consists of two major components; a register file and an ALU. The register file consists of nine general registers to support the PLM instruction set. These registers are mainly used to store pointers to indicate the current state of computation and the previous state for backtracking. The other special registers—T1, T, MDR, and R separate an operation into three stages for pipelining. In the first stage (C stage), the T and T1 registers are loaded with data. Then, at the next stage (E stage) the ALU performs corresponding commands and intermediate results are placed in the R and MDR registers.

In the final stage (P stage), the results are transferred to the general registers in the register file. A typical microroutine involves one C stage operation, several E stage operations and one P stage operation. By pipelining these three stages, system performance can be improved. The Parallel Prolog Processor (PPP) is a parallel extension of the PLM to exploit AND/OR parallelism (Fagin and Despain, 1987). A commercial version of PLM called X1 is also being developed by Xenologic (Dorby, 1987; Ribler, 1987).

The Integrated Prolog Processor (IPP) (Abe et al., 1987), which is one of the highest performance Prolog machines proposed so far, uses optimal arguments (up to 2) instead of the first argument for indexing. The optimal argument is the one on which clauses can be discriminated. Consider, for example, the following clauses:

$$p(X, r(Y,Z), [H \mid T]) \leftarrow \cdots$$
$$p(X, s(Y,Z), [H \mid T]) \leftarrow \cdots$$

The second arguments are optimal for indexing since the clauses differ from each other only in the second arguments. Alternatively, an argument having the maximum number of constants is selected as the optimal argument. If there are more than two arguments that satisfy this condition, the left-most two optimal arguments are selected for indexing. If there are no such arguments satisfying the above conditions, then the first argument is selected for indexing as in WAM. This indexing scheme contributes to the high performance of IPP.

8 Data base machines

Research and development on database machines has drawn great attention in the past two decades as high-level data models such as the relational model becomes more popular. Modern database systems consist of multiple layers of software, which map high-level commands into low-level storage

representations. This is a major cause of inefficiency under conventional computer systems. The main purpose of building a database machine is to reduce the gap between the semantic model and the actual internal representation of data (Su, 1988). In the context of logic-oriented knowledge base systems, we can consider two types of database machines; deductive database machines and non-relational database machines. The former are designed as back-ends to inference machines and mainly support the relational model by providing the means for efficient relational operations for general rules. The machines in the latter category provide the basic mechanism for handling complex objects. We consider the physical data organizations of these machines since efficient management of a large data space is one of the most critical requirements of a database machine (Boral and Dewitt, 1983; Stone, 1987).

8.a Deductive database machines

Deductive databases use logic programming rules to define virtual relations and the relational operations required to obtain the virtual relations. Queries are expressed by goals in logic programming, and through recursion, the least fixed point queries can also be handled. Deductive database machines are back end computers designed to manage very large fact bases by efficiently supporting the relational operations. In this section, we describe the architectures of three representative deductive database machines including ILEX (Li, 1984) DELTA (Sakai et al., 1986), and MPDC (Tanaka, 1986).

8.a.1 ILEX

The ILEX machine proposed by Li is a typical example of knowledge base systems based on the combination of existing systems (Li, 1984). In order to avoid modifications of an existing logic programming system residing in the host and to provide an interface between a data manipulation language and a logic programming language, a meta-level procedure transforming a query written in the data manipulation language to the corresponding canonical logic programming form is used.

The hardware of ILEX consists of three major components (Figure 11); a PDP-11 host, Relational Associative Processor (RAP), and an information retrieval system called MEMEX. The host is mainly responsible for user interface operations and performs compilation and optimization of user queries. RAP sends transformed queries to MEMEX for searching, and then performs database operations on the qualified data by using a content addressable mechanism. The MEMEX is a specialized information retrieval system for searching text by using inverted lists.

The KM-1 proposed by Kellog (1986) takes a similar approach to that of ILEX in the sense that it uses existing systems rather than developing specialized hardwares (the Xerox 1100 Lisp machine as the front-end and the Britton-Lee IDM-500 relational database machine as the back-end).

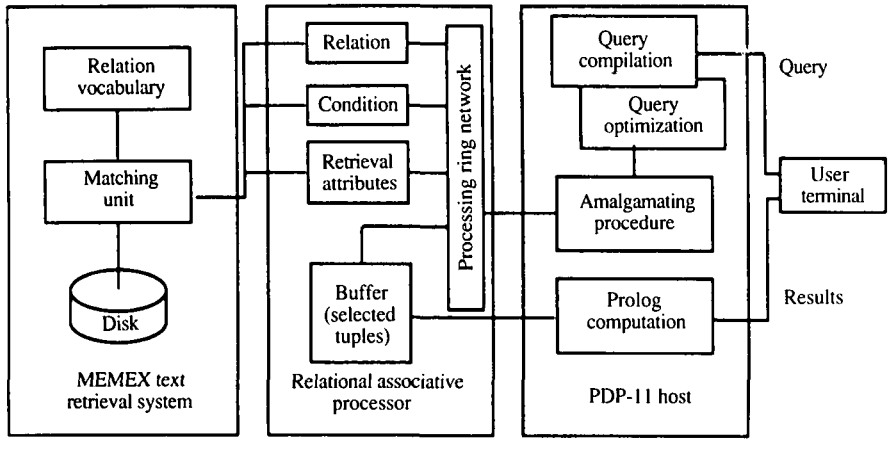


Figure 11 ILEX (Li, 1984)

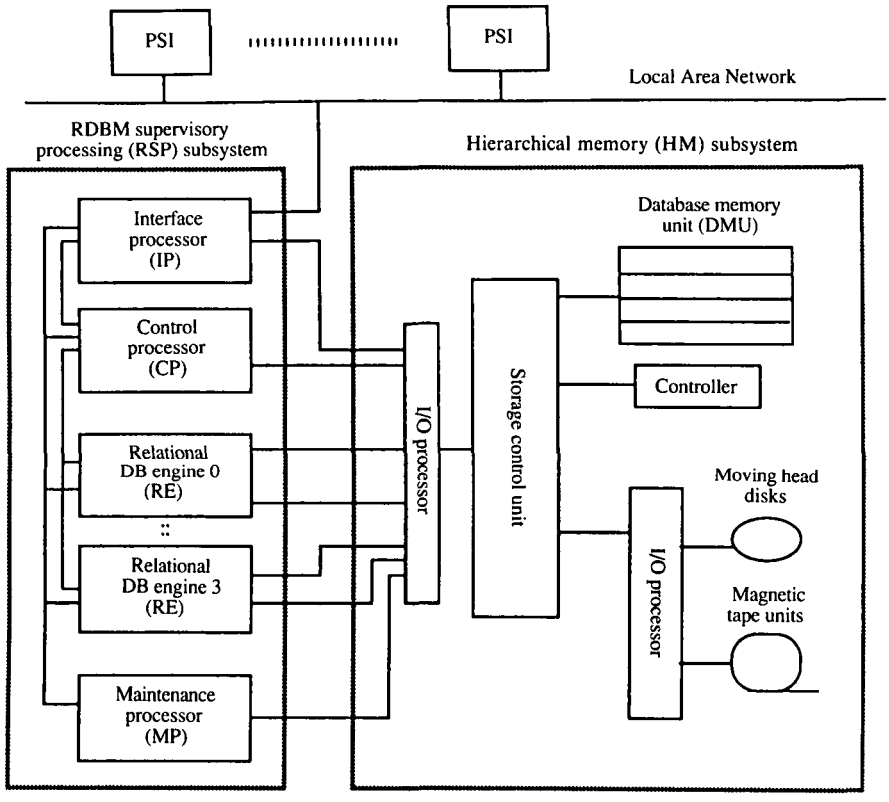


Figure 12 DELTA (Sakai et al., 1986)

8.a.2 DELTA

DELTA is a functionally distributed MIMD database machine supporting inference machines (PSI's) as a back-end (Sakai et al., 1986). It consists of two subsystems: the *relational database management supervisory/processing* (RSP) subsystem and the *hierarchical memory* (HM) subsystem. Figure 12 shows the overall organizations of DELTA. The RSP subsystem consists of four components each of which functions as follows:

- *interface processor* (IP) which is responsible for communications between the hosts and the CP, and between the hosts and the HM subsystem.
- *control processor* (CP) which transforms queries into subcommands.
- *relational database engines* (RE's) which performs relational operations. Join operations can be performed in O(N) by using a pipelined merge-sort algorithm and specialized hardware.
- *maintenance processor* (MP) which monitors the status of the system.

The HM subsystem is used to store large databases, and provides access paths for efficient retrieval of qualified tuples. The HM has a two-layer structure; the lower level consists of moving head disks, and the upper level is a fast RAM-cache memory called the database memory unit (DMU) which can be further divided into a buffer area, a disk cache area, a HM control program area and a working data area for the HM control program.

8.a.3 MPDC

The Massive Parallel Database Computer (MPDC) (Tanaka, 1986) consists of two subsystems; the *control subsystem* and the *data subsystem* (Figure 13).

The data subsystem consists of a number of segment processors, a number of disk systems, and a shared page buffer memory for fast data transfer between the two sets of systems. The multi-port

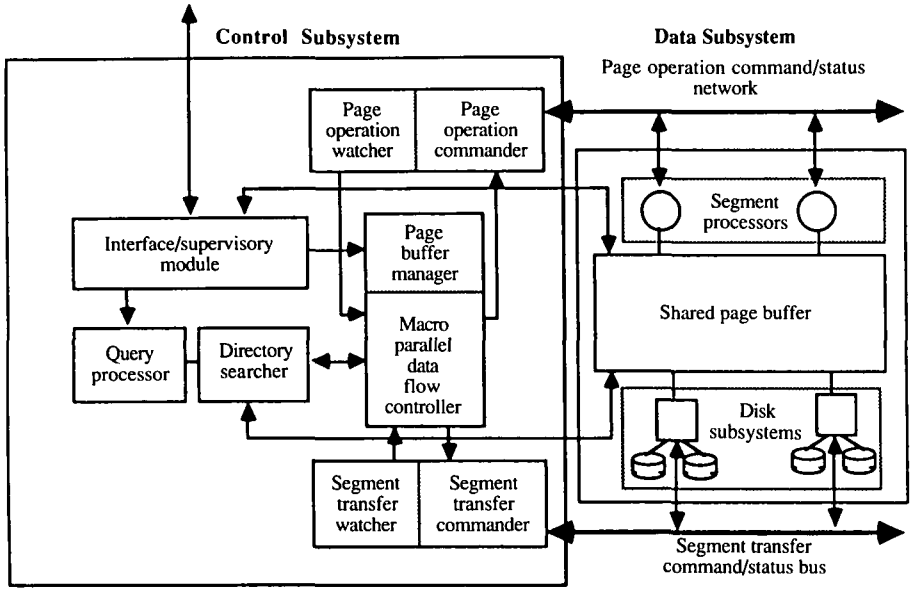


Figure 13 Massively Parallel Database Computer (MPDC) (Tanaka, 1986)

page memory (MPPM) (Tanaka, 1984) is used as the page buffer, which allows a segment processor to access a page of the MPPM without contention. This memory device is also used in an integrated knowledge base machine as described later. A segment processor consists of a high performance microprocessor, a main memory with capacity for more than four buffer pages, a search engine, and a sort engine, which is mainly responsible for relational operations on one or two pages of data. The segment processors are connected by a ring network called the *page operation command/status network* where the *page operation commander* of the control subsystem transmits relational operation commands. Monitoring the completion of the commands is the responsibility of the *page operation watcher*. The *segment transfer commander* issues commands for data transfer between disk systems and the shared page buffer through the *segment transfer command/status bus*.

The control subsystem is mainly concerned with the optimization of concurrently executable relational operation commands by searching the directory for partitioned data, and providing the host interface through the *interface/supervisory module*. The *query processor* in the control subsystem transforms the given query into an optimized program where high-level concurrency control commands are added. The transformed program is then processed by the *directory searcher* which generates a series of segment processing commands by using the grid file directory stored in the memory of the directory searcher. The *macroparallel data flow controller* issues commands to the data subsystem based on status information and the data flow analysis of the segment processing commands received from the directory searcher. The *page buffer manager* dynamically allocates pages to save the results requiring more than one page.

8.b Database machines for unnormalized relation

Most of the database machines that have been developed so far have been designed for the relational model. The data in the relational model strictly follows the first normal form restriction where each attribute value must not be decomposable. Many researchers became aware of this limitation in the context of knowledge-based systems and in semantic modeling, and thus have investigated the use of non-first normal form (NFNF) relations (Dadam et al., 1986; Roth et al., 1987). In this section, we consider a database machine CASSM (Su et al., 1979) which is designed for the hierarchical model, and the B-LOG machine (Lipovski and Hermenegildo, 1985) for data/knowledge bases having network-like structures.

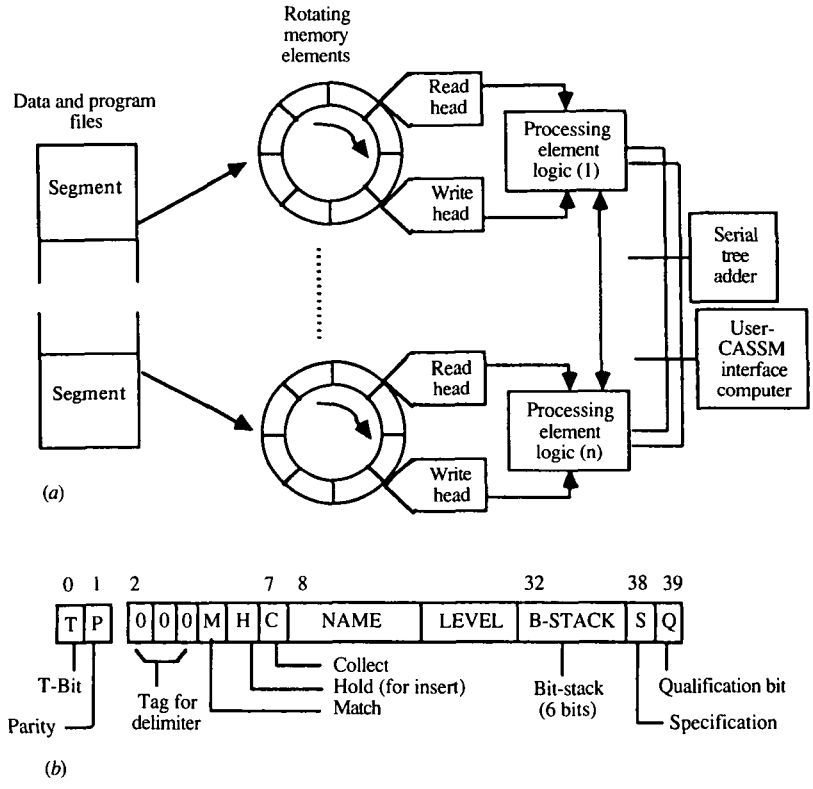


Figure 14 Context Addressed Segment Sequential Memory (CASSM). (a) The Architecture of CASSM. (b) The Word Organization of CASSM. (Su et al., 1979).

8.b.1 CASSM

Context Addressed Segment Sequential Memory (CASSM) is one of the earliest database machines (Su et al., 1979), which supports hierarchically structured relations (Su and Emam, 1978). The CASSM has facilities for searching complex structured data types such as sets, trees, and directed graphs. Another distinctive feature of the CASSM is that it can fetch compiled programs from disks. This concept is similar to that of logic programming systems, that is, program and data are uniformly managed.

The CASSM contains a linear array of cells each of which consists of a processing element and a rotating memory device. The overall organization of CASSM is shown in Figure 14(a). A hierarchical record is linearized in a top-down and left-right order (level-order), and is physically stored in a linear vector consisting of 40-bits words. A word is composed of an 8-bit tag and 32-bits of data. According to the tag bits, eight data types can be identified, including delimiters, name-values, pointers, strings, instructions, operands, protect-locks, and erased data. Figure 14(b) shows the data structure of the delimiter type. Since CASSM contains data and program represented by hierarchies, each node of a hierarchy has a unique level number. As seen in the figure, the tag consists of a bit stack (6 bits), S bit (Specification Bit), and Q bit (Qualification Bit). The bit stack is used for storing intermediate results. The Q bit is used to mark a context to search a specific subtree, while the S bit is used to find nodes satisfying the specification given in the query. Due to its associative processing nature, processing nodes at lower levels is as easy as processing higher level nodes.

8.b.2 B-LOG machine

Lipovski and Hermenegildo (1985) proposed a parallel "branch and bound" algorithm called B-LOG to efficiently evaluate logic programs stored in secondary storage. A parallel computer exploiting a special secondary storage named a Semantic Paging Disk (SPD) (Lipovski, 1978) is proposed to implement the B-LOG scheme (Figure 15). The SPD is designed to support pointer-

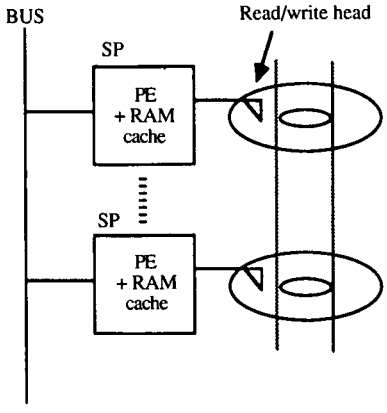


Figure 15 Semantic Paging Disk in the B-LOG Machine (Lipovski and Hermenegildo, 1985)

based searches for databases with a network structure, and consists of one or more search processors (SP). Each SP has a read–write head corresponding to one or more tracks, a random access memory able to hold a track's data, and special logic. The logic is designed to perform search and mark operations by traversing pointers. The basic task of the SPD is thus to provide processors with appropriate subsets of data needed for further processing.

Since the SPD works effectively on complexly structured data linked by pointers and stored on secondary storage, rules and facts are represented by a network-like model. A block representing a rule or a fact contains pointers to other blocks of subgoals, and weights (bounds) to guide the search procedures for finding the optimal path. The search strategy adopted by the B-LOG machine is the best-first search combined with a branch-and-bound algorithm. Although the best-first search has advantages over depth-first or breadth-first search in the context of parallel processing, it is not an easy task in the evaluation of logic programs to assign a weight to each arc. In the B-LOG approach, weights are added to each branch of the OR-tree. The AND nodes, the conjunction of subgoals, are assumed to be executed sequentially. A heuristic success probability is introduced as a basis for the branch-and-bound algorithm.

9 Integrated knowledge base machines

In this section, we consider integrated knowledge base machines which can manage both facts and rules in a uniform way. For very large knowledge bases, these machines should support unification-based retrieval of clauses from secondary storage, and thus require enormous processing loads. They generally exploit the top-down strategy on which most logic programming languages are based, rather than the databases' conventional bottom-up strategy.

9.a Unification filter

The unification filter proposed by Sabbatel and Dang (1987) is the first unification machine that can find candidate clauses or unifiable terms as the data come from secondary storage. This machine exploits "on the fly" unification and set-oriented retrieval by using a pipelined method. The proposed hardware component functions similarly to the database filters (Figure 16).

The unification filter performs the preunification step, and the rest of unification is performed on the binding variables supplied from the filter by using the software executed on the processing elements. In the preunification step, two terms are compared for equality, regarding all variables as "don't care" matches. The preunification step can be further decomposed into four steps which can be executed in sequence by the following four operators:

1 the *structure operator* manages the terms in the input buffer, and analyzes the structure of the goals to classify the types by using a stack. Then, the positions of the subterms are encoded for

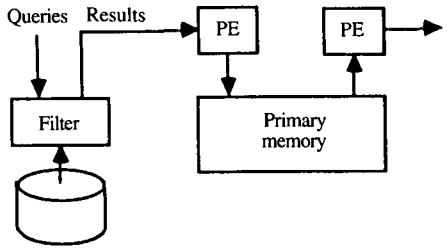


Figure 16 Unification Filter (Sabbatel and Dang, 1987)

further processing. Some binding information can be sent to the output buffer for the subterms corresponding to the variable positions of the given goal.

2 the *search operator* compares pairs of constant values.

3 the *set operator* manages set of unifiable terms with the given goal.

4 the *selection operator* manages the output buffer. It removes some substitutions transmitted by the structure operator for the ununifiable terms.

The concept of the unification filter is very similar to some stream-based unification machines described previously.

9.b Massively parallel machines

Many massively parallel machines aimed at high performance AI systems have been designed. But when large amounts of data are involved, transferring data from secondary storage to the processing level can be a serious bottleneck in these machines. We present three massively parallel machines which can be used for logic-oriented knowledge bases, and assess the issue of the I/O bottleneck.

9.b.1 DADO

DADO consists of a large number of identical processing elements (PEs) connected by using a binary tree topology, and was originally designed to provide high performance in the execution of large production rule systems (Stolfo, 1987). A Control Processor (CP) is attached to the root of the DADO tree, which is mainly responsible for broadcasting data to the PEs.

Taylor et al. (1983, 1984) proposed an implementation technique for OR-parallel execution of logic programs by exploiting the parallel pattern matching capability of DADO. For logic program evaluation, clauses are initially distributed across DADO as follows (Figure 17):

- the bodies of rules are stored in the CP,
- clause heads are stored across each PE, and
- an index is associated with each clause head and the corresponding body.

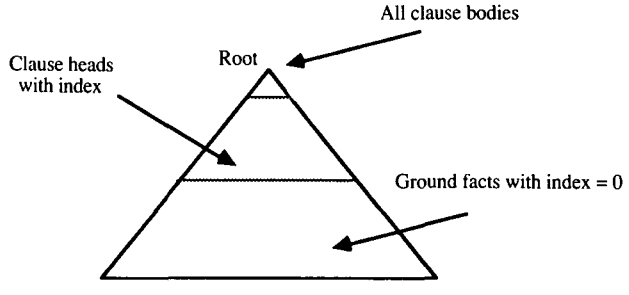


Figure 17 Logic Programming Evaluation in DADO (Taylor et al., 1983)

The proposed parallel evaluation procedure is as follows:

1 The CP broadcasts the given goal to all PEs.
2 Each PE performs unification between the broadcast goal and stored heads.
3 The CP polls each PE to collect bindings first from unit clauses, and then from rule heads.
4 The collected binding sets are merged. For example, {X/a(Y), Y/b} is reduced to {X/a(b)}.
5 The conjunction is first solved independently in a left-to-right manner. Then join is performed for shared variables.

The major problem of the DADO architecture for data intensive applications lies in slow data transfer time; the data is broadcast from the root, resulting in high overhead for large knowledge bases.

9.b.2 NON-VON

Shaw proposed a massively parallel database machine called NON-VON (Shaw, 1985). Although NON-VON was originally designed for efficient relational operations, it can also support rule-based systems (Hillyer and Shaw, 1986). As shown in Figure 18, NON-VON consists of two major components; a primary processing subsystem and a secondary processing subsystem. The primary processing subsystem is composed of many small processing elements (SPE)(S in the figure) and large processing elements (LPE). The SPE's are interconnected by a tree network as in DADO, but leaf nodes are also connected via a mesh connection. Each SPE has a very small amount of local memory where data are stored. A LPE is a general purpose microcomputer having a large random access memory to hold programs, and connected to a SPE located near the root of the tree. The operation of LPEs is primarily in a MIMD mode, while SPEs perform instructions broadcast by LPEs in a SIMD mode. Thus, NON-VON can operate in a multiple-SIMD mode. The secondary processing subsystem consists of a large number of disk systems each of which is connected to a LPE. Associated with the secondary processing subsystem is the intelligent head unit capable of on-the-fly data filtering. Thus, NON-VON can offer higher I/O bandwidths from secondary storage than DADO.

9.b.3 Connection machine

The Connection Machine Model CM-2 (Thinking Machines Technical Report, 1987) is a massively parallel back-end computer consisting of a very large number of simple data processors (16K, 32K,

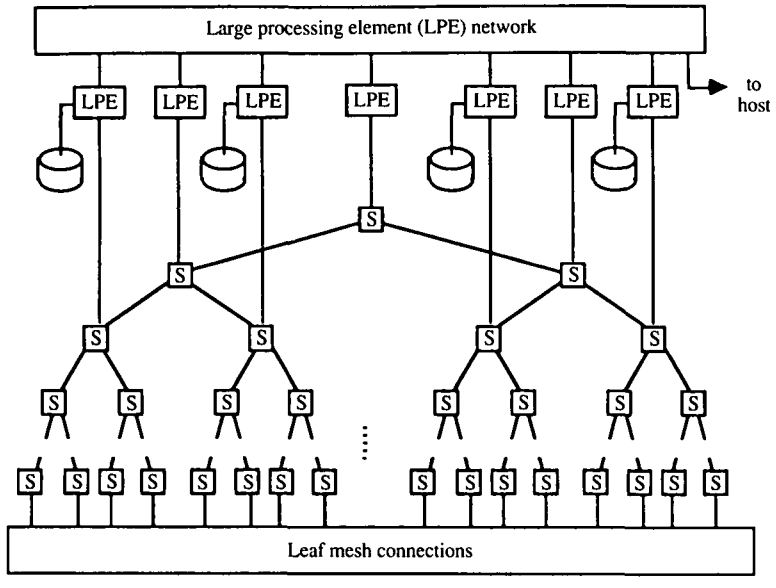


Figure 18 NON-VON (Shaw, 1985)

or 64K), a number of sequencers, a Nexus switch, a number of floating-point accelerator chips, and I/O systems. By using system software, a user can create more than four million virtual processors devoted solely to a user's program. Each data processor has a very simple structure consisting of 8 Kb of bit-addressable local memory and an arithmetic logic unit. The most important feature of the CM stems from its communication ability among data processors. Sixteen data processors are fabricated in a chip, each of which is connected by a cube topology. Associated with a chip is a router which allows many simultaneous accesses to the local memory of other processors in a very general way. A less general, but efficient, communication mechanism is provided by organizing the virtual processors as a two dimensional grid, which allows all the virtual processors to communicate to their neighboring processors simultaneously. A data processor does not directly execute parallel instructions. Instead, they are processed by a sequencer. Processors can be divided into up to four independent sections each of which can have its own sequencer, router, and grid. Up to four front-ends can be connected by the Nexus which is a programmable, bidirectional switch.

The CM-2's I/O structure allows data to be moved into or out of the parallel processing unit at peak rate as high as 320 Mb/s. I/O is done in parallel by allowing as much as 2 K processors to simultaneously perform I/O related operations when the maximum number of I/O channels are available. The Data Vault, CM-2's mass storage system, can be connected to each I/O channel, which can transfer data at the rate of 40 Mb/s and can hold data up to 10 Gb. A Data Vault unit consists of 42 winchester disks and 6 SCSI controllers working in parallel.

9.c Intelligent virtual memory support/disk cache

One of the approaches in realizing a logic-oriented very large knowledge base is to provide a logic programming system with facilities to access secondary storage. Intelligent virtual memories and special disk caches have been proposed for this purpose.

9.c.1 MANIP-2

MANIP (Wah et al., 1984) is designed to solve combinatorial searching problems in parallel by a branch-and-bound algorithm with the best-first search strategy. The branch-and-bound algorithm partitions a large problem into many smaller subproblems until a solution is obtained or infeasiblity is found. The best-first search strategy selects the most feasible subproblem based on heuristic values assigned to subproblems. As the best-first search requires more memory space, especially in a parallel processing environment, special care must be taken for efficient management of the memory space. MANIP exploits a special virtual memory for this purpose (Yu and Wah, 1983).

After the problem is decomposed into many small subproblems, the subproblems stored in secondary storage are organized as a B^+ tree (Comer, 1979) whose leaf nodes represent pages containing one or more subproblems. Maintained in main memory are a heap of pointers to a partial list of subproblems also residing in main memory. When a subproblem is generated, it is added to the list until the main memory space runs out. Then, the subproblems are inserted to the B^+ tree.

MANIP-2 (Li and Wah, 1985) is designed for parallel heuristic evaluation of logic programming with a similar architecture to MANIP (Figure 19). The logic programming evaluation procedure can be represented by an OR-tree as in B-LOG, where a node denoting a subproblem can be a goal, or by an AND/OR tree where each node can be either a goal or a subgoal. Due to the AND/OR tree representation, the calculation of heuristic values of MANIP-2 is more complex than that of B-LOG since weights should be added to both AND and OR nodes.

9.c.2 Mu-X

One of the main research areas in Japan's FGCS project has been directed towards the design of parallel computer architectures for efficient processing of very large knowledge bases. A variety of

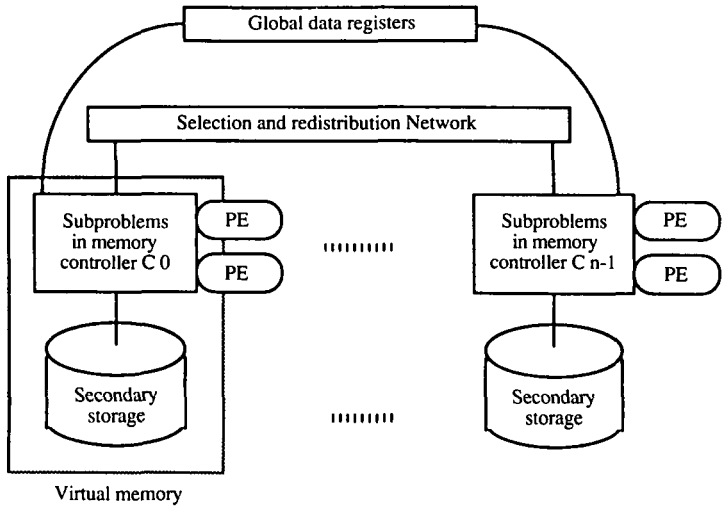
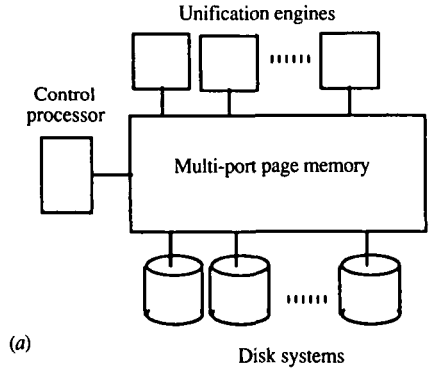


Figure 19 MANIP-2 (Li and Wah, 1985)

computer architectures and underlying knowledge base models has been explored in the past several years. The first proposed integrated knowledge base machine was designed with a number of unification engines (UE), a control processor, a multi port page memory (MPPM), and a number of disk systems attached to the MPPM. Figure 20 shows the basic architecture of the parallel knowledge base machine (Yokota and Itoh, 1986).

However, it was soon recognized that the hardware unification engine could provide only limited functionality and small performance enhancement. In a more recent proposal (Itoh et al., 1988), the knowledge base machine called Mu-X uses eight general purpose microprocessors each of which has 2 Mb of main memory, instead of the unification engines. Mu-X still uses the MPPM designed by Tanaka (1984) which serves as a large shared buffer for storing pages as a basic access unit. The MPPM consists of a set of I/O ports, a set of memory banks, and a switching network (Omega network) connecting the ports with the memory banks.



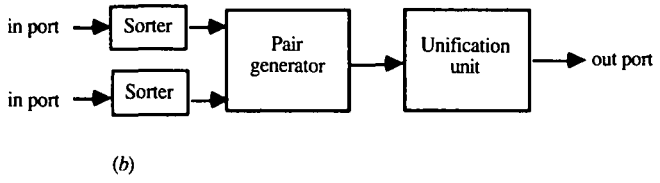


Figure 20 MPPM with Multiple Unification Engines. (a) Systems Configuration of a KBM. (b) Unification Engine (UE) (Yokota and Itoh, 1986)

10 Conclusions

The integration of logic programming and database places new demands on the performance of these two systems. In this paper, we have investigated knowledge base models, operations, and specialized computer architectures for logic-oriented knowledge base systems which are summarized in Table 2. A variety of approaches has been taken to improve the performance of time-consuming tasks to managing large data objects in both logic programming and database processing environments.

We have described four knowledge base models used in designing logic-oriented knowledge bases. Each model uses a different approach to integrating the two types of systems, and thus requires different operations for knowledge processing. Main memory resident KB systems are not concerned with any operations inherited from database systems. In loosely or tightly coupled KB systems, on the other hand, database systems at the back end should support Least Fixed Point (LFP) operations for queries based on recursively defined rules as well as conventional database operations. Integrated KB systems also require operations such as Extended Relational Algebra (ERA) or Retrieval By Unification (RBU) for managing complex objects.

Since knowledge processing requires enormous processing loads, a number of specialized computer architectures have been proposed to enhance the performance of knowledge base processing. We described the architectures of some machines based on the taxonomy presented in Section 6. Parallelism certainly plays the most important role in designing these machines. For data intensive operations, there exists opportunities for performance improvements that can be achieved

Table 2 Summary

KBM Classes	*KBM Machines*	*KB Models*	*Basic Tasks*	*Techniques Used*
C-KBM	• Inference Machines Unification machines – Unification coprocessors (Robinson, 1985; Stormon, 1986; Woo, 1985a) – Stream Unification Machines (Shobatake and Aiso, 1986; Tanabe and Aiso, 1987)	Main memory Resident KB model	Unification Resolution	Compilation Clause Indexing AND/OR/Low Level Parallelism Pipelining
	Resolution Machines – Associative Processors (Hahne et al., 1985; Schneider and Dilger, 1986) – Prolog Machines (Abe et a., 1987; Dorby et al., 1984, 1985; Fagin and Despain, 1987)	Loosely and Tightly Coupled KB Model	Relational Operations LFP	Parallel I/O Parallel Algorithms for Relational Operations Data Cache
	• Database Machines Deductive Database Machines (Li, 1984; Sakai et al., 1986; Tanaka, 1986) Database Machines for Unnormalized Relations (Lipovski, 1978; Su et al., 1979)			Physical Data Ortganization for Join and Partial Match
I-KBM	• Integrated KB Machines Unification Filter (Sabbatel and Dang, 1987) Massively Parallel Machines (Shaw, 1985; Stolfo, 1987; Thinking Machines Technical Report, 1987) Intelligent Memory machines (Wah et al., 1984; Yokota and Itoh, 1986)	Integrated KB Model	RBU ERA	Data Filter Parallel Process Management Term/Clause Indexing Contention Free Shared Buffer

by exploiting parallelism since the data rate from slow secondary storage can be increased proportionally to the number of disk systems working in parallel. However, for inferencing, parallel processing does not gain much speed up.

Another important issue lies in the design of physical data organization schemes for very large knowledge bases, since brute force parallelism does not offer much advantage over sequential processing. We have investigated a number of physical data organization schemes and indexing schemes used in the context of logic-oriented knowledge base processing. For knowledge bases consisting of complex objects, the design of effective data organizations can be one of the most critical factors influencing overall system performance. Effective utilization of specialized hardware components and inherent parallelism must be considered for enhanced physical data organizations.

High performance front ends for the inferencing mechanism are technically feasible, and can enhance the performance of knowledge base processing by using the compilation technique. However, when large volumes of data are supported in knowledge bases, most operations involve extensive I/O bound tasks. Furthermore, the construction of hardware components dedicated to special functions require great complexity and tend to be inflexible. Due to advances in hardware technology, general purpose state-of-the-art microprocessors are powerful and cost effective. The future research on knowledge base machines will be directed toward parallel computer architectures consisting of multiple general purpose microprocessors and high performance secondary storage systems.

References

Abe, S, Bandoh, T, Yamaguchi, S, Kurosawa, K and Kiriyama, K, 1987. "High performance Integrated Prolog Processor IPP" In: *Proceedings of the 14th Annual Symposium on Computer Architecture*, pp 100–107.

Aho, AV and Ullman, JD, 1979. "Universality of data retrieval languages" In: *ACM Symposium on Principles of Programming Languages*, pp 110–117

Berra, PB, Chung, SM and Hachem, N, 1987. "Computer architecture for a surrogate file to a very large data/knowledge base", *IEEE Computer* **20**(3) pp 25–32

Berra, PB and Oliver, E, 1979. "The role of associative array processors in database machine architecture", *IEEE Computer* **12**(3) pp 53–61

Boral, H and DeWitt, DJ, 1983. "Database machines: An idea whose time has passed? A critique of the future of database machines" In: *Proceedings of 3rd International Workshop on Database Machines*, pp 166–187

Boral, H and Redfield, S, 1985. "Database machine morphology" In: *Proceedings of 11th International Conference on VLDB*, pp 59–71

Comer, D, 1979. "The ubiquitous B-tree", *ACM Computing Surveys* **11**(2) pp 121–137

Conery, JS, 1987. *Parallel Execution of Logic Programs*, Kluwer Academic Publishers: Boston, Massachusetts

Dadam, P, et al., 1986. "A DBMS prototype to support extended NF^2 relations: an integrated view on flat tables and hierarchies" In: *Proceedings of SIGMOD '86*, pp 356–367

DeWitt, DJ and Gerber, R, 1985. "Multiprocessor hash-based join algorithms" In: *Proceedings of the 11th International Conference on Very Large Data Bases*, pp 151–164

Dorby, T, 1987. "A coprocessor for AI; LISP, Prolog and data bases" In: *COMPCON Spring '87*, pp 396–402

Dorby, TP, Despain, AM and Patt, YN, 1985. "Performance studies of a Prolog machine architecture" In: *Proceedings of the 12th Symposium on Computer Architectures*, pp 180–190

Dorby, TP, Patt, YN and Despain, AM, 1984. "Design decisions influencing the microarchitecture for a Prolog machine" In: *Micro 17 Proceedings*, pp 217–231

Dwork, C, Kanellakis, P and Mitchell, J, 1984. "On the sequential nature of unification" *Journal of Logic Programming* **1** pp 35–50

Fagin, BS and Despain, AM, 1987. "Performance studies of a parallel Prolog architecture" In: *Proceedings of the 14th Annual Symposium on Computer Architecture*, pp 108–116

Fuchi, K and Furukawa, K, 1986. "The role of logic programming in the fifth generation computer project" In: *Proceedings of the Third International Conference on Logic Programming*, pp 1–24

Gonzalez-Rubio, R, Rohmer, J, Bradier, A and Bergsten, B, 1987. "DDC: a deductive database machine" In: *Proceedings of the Fifth International Workshop on Database Machines*, pp 116–129

Hahne, K, Pilgram, P, Schuett, D, Schweppe, H and Wolf, G, 1985. "Associative processing in standard and deductive databases" In: DeWitt, DJ and Boral, H, eds, *Database Machines—Fourth International Workshop*, New York: Springer-Verlag, pp 1–12

Harland, J and Jaffar, J, 1987. "On parallel unification for Prolog" *New Generation Computing* **5** pp 259–279

Hillyer, BK and Shaw, DE, 1986. "Execution of OPS5 production systems on a massively parallel machine" *Journal of Parallel and Distributed Computing* **3** pp 236–268

Hwang, K, Ghosh, J and Chowkwanyun, R, 1987. "Computer architectures for artificial intelligence processing" *IEEE Computer* **20**(1) pp 19–27

Itoh, H et al., 1988. "Knowledge base system in logic programming paradigm" In: *Proceedings of the 1988 International Conference on Fifth Generation Computer Systems*

Kellog, C, 1986. "From data management to knowledge management" *IEEE Computer* **19**(1) pp 75–84

Kim, MY, 1986. "Synchronized disk interleaving" *IEEE Transactions on Computers* **C-35**(11) pp 978–988

Li, D, 1984 *Prolog Database System*, London: Research Studies Press

Li, G and Wah, BW, 1985. "MANIP-2: a multicomputer architecture for evaluating logic programs" In: *Proceedings of the International Conference on Parallel Processing*, pp 123–130

Lipovski, GJ, 1978. "Semantic paging on intelligent disks" In: *Proceedings of the Fourth Workshop on Computer Architecture for Non-Numeric Processing*, pp 30–34

Lipovski, GJ and Hermenegildo, MV, 1985. "B-LOG: a branch and bound methodology for the parallel execution of logic programs" In: *Proceedings of International Conference on Parallel Processing*, pp 123–130

Nakazaki, R et al., 1985. "Design of a high-speed Prolog machine (HPM)" In: *Proceedings of the 12th International Symposium on Computer Architectures*, pp 191–197

Nievergelt, J, Hinterberger, H and Sevcik, KC, 1984. "The grid file: an adaptable, symmetric multikey file structure" *ACM Transactions on Database Systems* **9**(1) pp 38–71

Noye, J and Syre, J-C, 1987. "ICM3: design and evaluation of an inference crunching machine" In: *Proceedings of the 5th International Workshop on Database Machines*, pp 1–14

Ozkarahan, EA and Bozsahin, CH, 1988. "Join strategies using data space partitioning" *New Generation Computing* **6** pp 19–39

Qadah, GZ, 1985. "Database machines: a survey" In: *National Computer Conference*, pp 212–223

Ramamohanarao, K and Shepherd, J, 1986. "A superimposed codeword indexing scheme for very large Prolog databases" In: *Proceedings of the 3rd International Conference on Logic Programming*, pp 569–576

Ribler, RL, 1987. "The integration of the Xenologic X-1 artificial intelligence coprocessor with general purpose computers" In: *COMPCON Spring '87*, pp 403–407

Robinson, P, 1985. "The SUM: an AI coprocessor" *Byte* **10**(6) pp 169–180

Roth, MA, Korth, HF and Batory, DS, 1987. "SQL/NF: a query language for $\neg 1NF$ relational databases" *Information Systems* **132**(1) pp 99–114

Sabbatel, GB and Dang, W, "Search strategy for Prolog data bases" In: *Proceedings of the 5th International Workshop on Database Machines*, pp 654–667

Sakai, H et al., 1986. "Development of Delta as a first step to a knowledge base machine" In: Sood, AK and Qureshi, AH, eds, *Database Machines* New York: Springer-Verlag, pp 159–181

Schneider, H-A and Dilger, W, 1986. "Information processing with associative processors" In: *Proceedings of the Conference on Algorithms and Hardware for Parallel Processing*, pp 222–229

Shaw, D, 1985. "Relational query processing on the Non-Von supercomputer" In: Kim, W, Reiner, DS and Batory, DS, eds, *Query Processing in Database Systems*, New York: Springer-Verlag, pp 248–258

Shin, D and Berra, PB, 1989. "Surrogate file approach to managing first order terms in secondary storage" In: *Proceedings of the SPIE Conference on Applications of AI VII*, pp 1051–1062

Shobatake, Y and Aiso, H, 1986. "A unification processor based on a uniformly structured cellular hardware" In: *Proceedings of the 13th International Symposium on Computer Architectures*, pp 140–148

Stolfo, SJ, 1987. "Initial performance of the DADO 2 prototype" *IEEE Computer* **20**(1) pp 75–83

Stone, HS, 1987. "Parallel querying of large databases: a case study" *IEEE Computer* **20**(10) pp 11–21

Stormon, CD, 1986. *An Associative Processor and Its Application to Logic Programming Computation* Technical Report 8611, Syracuse University-CASE Center

Su, SYW, 1988. *Database Computers—Principles, Architectures, and Techniques*, New York: McGraw-Hill

Su, SYW and Emam, A, 1978. "CASDAL: CASSM's DAta Language" *ACM Transations on Database Systems* **3**(1) pp 57–91

Su, SYW, Nguyen, LH, Emam, A and Lipovski, GJ, 1979. "The architectural features and implementation techniques of the multicell CASSM" *IEEE Transactions on Computers* **C-28**(6) pp 430–445

Tanabe, M and Aiso, H, 1987. "The unification processor by pipeline method" In: *Proceedings of the 5th International Workshop on Database Machines*, pp 668–680

Tanaka, Y, 1986. "Massive parallel database computer MPDC and its control schemes for massively parallel processing" In: Sood, AK and Qureshi, AH, eds, *Database Machines*, New York: Springer-Verlag, pp 127–158

Tanaka, Y, 1984. "A multiport page-memory architecture and a multiport disk-cache system" *New Generation Computing* **2** pp 241–260

Taylor, S et al., 1984. "Logic programming using parallel associative operations" In: *Proceedings of the 1984 International Symposium on Logic Programming*, pp 58–68

Taylor, S, Maio, C, Stolfo, SJ and Shaw, DE, 1983. *Prolog on the DADO Machine: A Parallel System for High-Speed Logic Programming* Technical Report CUCS-46-83, Columbia University

Thinking Machines Technical Report HA87-4, 1987. *Connection Machine Model CM-2 Technical Summary* Thinking Machines Co

Treleaven, PC and Refenes, AN, 1986. "Computer architecture for artificial intelligence" In: *Proceedings of the Advance Courses on Future Parallel Computers*, pp 416–492

Vitter, JS and Simons, RA, 1986. "New classes for parallel complexity: a study of unification and other complete problems for *P*" *IEEE Transactions on Computers* **C-35**(5) pp 403–418

Wada, M, Morita, Y, Yamazaki, H, Yamashita, S, Miyazaki, N and Itoh, H, 1987. 'A superimposed code scheme for deductive databases" In: *Proceedings of the 5th International Workshop on Database Machines*, pp 569–582

Wah, BW, Li, G and Yu, CF, 1989. "The status of MANIP—a multicomputer architecture for solving combinatorial extremum-search problems" In: *Proceedings of the 11th Annual Symposium on Computer Architecture*, pp 56–63

Warren, DHD, 1983. *An Abstract Prolog Instruction Set* Technical Report 306, SRI International

Woo, NS, 1985a. "The architecture of the hardware unification unit and an implementation" In: *Micro 18 Proceedings*, pp 89–98

Woo, NS, 1985b. "A hardware unification unit: design and analysis" In: *Proceedings of the 12th International Symposium on Computer Architectures*, pp 198–205

Yasuura, H, 1989. "On parallel computational complexity of unification" In: *Proceedings of the International Conference on Fifth Generation Computer Systems*, pp 235–243

Yokota, H and Itoh, H, 1986. "A model and an architecture for a relational knowledge base" In: *Proceedings of the 13th International Symposium on Computer Architectures*, pp 2–9

Yu, CF and Wah, BW, 1983."Virtual memory support for branch-and-bound algorithms" In: *COMPSAC*, pp 618–626

Zaniolo, C, 1985. "The representation and deductive retrieval of complex objects" In: *Proceedings of the 11th International Conference on VLDB*, pp 21–23