Parallel Management of Large Deductive Data Bases in a Multi-Processor Environment

Christophe Maciazek[2], Nick Bassiliades[1], *Student Member, IEEE*,
and Ioannis Vlahavas[1], *Member, IEEE*

[1]Dept. of Informatics,
Aristotle University of Thessaloniki
54006 Thessaloniki,
Greece

[2]IRESTE
University of Nantes
La Chantrerie - CP 3003
44087 Nantes Cedex 03, France

Tel: +3031-998145 Fax: +3031-206138
e-mail: vlahavas@olymp.ccf.auth.gr

Abstract - This paper describes a parallel deductive database system, built on top of Prolog. The system is based on the TOP-DOWN evaluation of logic programs. Parallelism is provided at the rule level, by transforming the query AND/OR tree into Disjunctive Normal Form. The clauses of the transformed formula are executed independently in parallel, on a transputer multi-processor machine, using the processor-farm algorithm. Both main-memory consultation and direct disk access have been implemented and tested. The measurement of the system performance shows speed improvement over the sequential Prolog interpreter, for large rule bases, but also exhibits implementation-dependent drawbacks that cause under-linear speed-up.

INTRODUCTION

Deductive databases have been identified [4] as a necessary amalgamation of logic and relational databases to provide an efficient management of knowledge-bases with large amounts of data and/or knowledge expressed in the form of rules. The management of large databases is constrained [9] by the size of information which can be stored in the main computer memory and by the search time. In this paper, we attempt to implement a deductive database system, relaxing the above constraints, by:

1) developing a Database Management System (DBMS) in PROLOG,
2) using a multi-processor system to execute queries in parallel,
3) consulting medium-to-large databases in main memory, and
4) accessing very large databases directly on disk instead of consulting them in main memory.

The system has been developed [3, 14] on a transputer-based machine and has been tested on a large sample database. Performance measurements allowed conclusions to be drawn about the efficiency of each algorithm and type of database access used.

LOGIC AND DATABASES

PROLOG blurs the distinction between databases and programs [13] and therefore a PROLOG program can be viewed as a database where the relations are described both by facts and rules. Rules are general axioms that compute data at query time, while facts are special rules without bodies and act as the actual data.

We may assume that the number of facts in a large database can be enormous while on the other hand, the number of rules is usually limited [2]. The former are stored in the Extensional Data Base (EDB) while the latter are stored in the Intentional Data Base (IDB). This allows rules and facts to be treated differently, for the shake of efficiency.

Queries to the database consist of PROLOG-like goals to be resolved. Resolution consists mainly of searching in the database for the facts and/or the rules that match with the query. Resolution can be done in several distinct ways, providing different database access plans that can result in quite different query evaluation times. In this work we explore the efficiency of query answering by exploiting the OR-parallelism of the execution of a goal, at the rule level only.

Related Work

There have been several attempts to integrate Prolog and Databases [2, 5, 15], based on the sequential Prolog interpreter. Prolog is based on a top-down evaluation of logic programs (backward chaining). Forward chaining or bottom-up evaluation has also been studied extensively for deductive databases [17], since it is argued that it is most suited to database applications [18]. The pros and cons of both approaches are discussed in [17]. In our work we follow the top-down approach.

Parallelisation of logic programs has been studied in both approaches, like [8, 18] for bottom-up evaluation in the deductive databases framework or [7, 16] for top-down evaluation in the framework of logic programming.

In respect to the terminology used in [7], our approach is top-down, using a kind of pure OR-parallel execution scheme. However, since we deal with databases, where all solutions to a problem are needed, our OR-parallel execution scheme keeps all alternative branches of the search tree and merges the answers produced by each. Moreover, our approach “compiles” the AND/OR execution tree into a DNF formula that can be executed in parallel with no run-time overheads.

IMPLEMENTATION ISSUES

The tasks of the DBMS can be outlined as:

1) Input of a user's query,
2) Query decomposition,

3) Parallel query execution, and
4) Output of the results to the user.

Query Decomposition

The first task of the DBMS is to read-in the user's query, therefore it does not require more explanation. The second task requires the decomposition of the initial query into ground calls. Ground calls are considered the following:

1) EDB relations (i.e. facts),
2) built-in PROLOG predicates, and
3) user-defined non-database predicates.

Ground calls are directly executable by the system meta-interpreter, since they either require a call to the built-in PROLOG interpreter, or the retrieval of a single EDB fact.

The transformation is done using Boolean algebra laws. The result of the transformation is the AND/OR tree, where terminal nodes are ground calls, while the internal nodes combine ground calls and internal nodes with any of the logical connectives AND, OR and NOT.

The above AND/OR tree is used for obtaining:

1) The list of the external relations that are needed to resolve the query, and
2) The Disjunctive Normal Form (DNF) of the query.

List of External Relations: The AND/OR tree is traversed in a depth-first manner and terminal nodes are collected in a bag. This bag is then turned into a set that contains the names of all the necessary EDB relations. The latter need to be consulted into main memory (or opened on the disk), to answer the original query.

```
?-  a(X), b(X,Y).

b(X,Y) :- d(X,Z), e(Z,Y).
b(X,Y) :- f(X,Y), not(c(Y)).
```

Figure 1. Example query

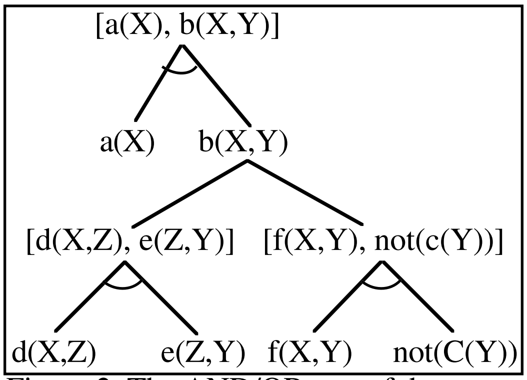

Figure 2. The AND/OR tree of the query

Disjunctive Normal Form: The AND/OR tree is not adequate for parallel processing, therefore it is normalised into DNF, in order to obtain a list of independent propositions that can be executed in parallel. The resulting tree is called the OR tree.

A formula is in DNF if it consists only of a disjunction of propositions which are either literals or conjuncted literals (disjunction-free formulas). A literal is an atomic proposition or its negation.

An example of a goal is shown in fig. 1, along with the rules of the IDB. The rest of the relations a/1, d/2, e/2, f/2 and c/1 are all EDB predicates. The AND/OR tree of the original query is shown in fig. 2 and the corresponding DNF (OR tree) in fig. 3.

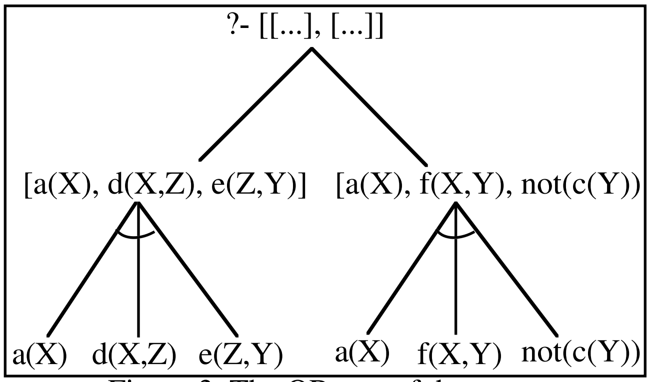

Figure 3. The OR tree of the query

Parallel Execution

Assuming the independence among the disjuncted "disjunction-free" propositions, they can be executed independently, in parallel, using OR-parallelism at the rule level. The execution of each proposition is assigned to a distinct concurrent process in a multi-processing system. Each process solves individually a part of the problem in a sequential manner and the union of the results is returned to the user. The concurrent processes are executed in parallel using the processor-farm algorithm with and without management.

Processor-Farm: The processor-farm algorithm is a very simple and efficient way to execute a set of independent tasks in parallel. According to this algorithm (fig. 4), there is a master process and N worker processes in the system.

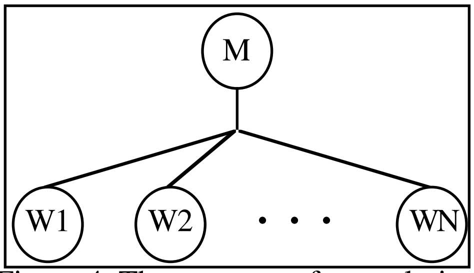

Figure 4. The processor-farm solution

The tasks of the master process are:

1) Initialising the worker processes.
2) Distributing the EDB data packets corresponding to the terms of the DNF among the worker processes. This task is valid only for the in-memory consultation of the EDB.
3) Distributing the disjunction-free clauses to be processed among the workers.
4) Collecting the results from the workers.

5) Displaying the results to the user when all the workers have finished and after removing duplicate answers.
6) Terminating the system.

The worker processes are all identical and consist of the following tasks:

1) Receiving a data packet with EDB facts from the master, during the initialisation phase. This task is valid only for the in-memory consultation of the EDB.
2) Receiving clauses to be processed from the master.
3) Processing the clauses sequentially, using either the internal, local EDB (for in-memory consultation) or the external common EDB (for direct disk access).
4) Sending the results to the master.

Processor-Farm with Process Management: The un-supervised processor-farm algorithm is adequately efficient for a small number of processes. When the number of processes gets larger than the number of available processors, then we need an allocation strategy. We have chosen the round-robin algorithm for the shake of simplicity. That means the master process distributes the clauses to be processed in a circular fashion among the workers.

Each process, though, does not have the same processing time with the rest. Therefore, the almost uniform distribution of tasks among the workers does not guarantee an equal workload on each. Therefore a process manager is needed to distribute the tasks to the workers according to the workload and not the number of the tasks. The objective of this manager is to balance the workload of each processor in the framework of the processor-farm algorithm.

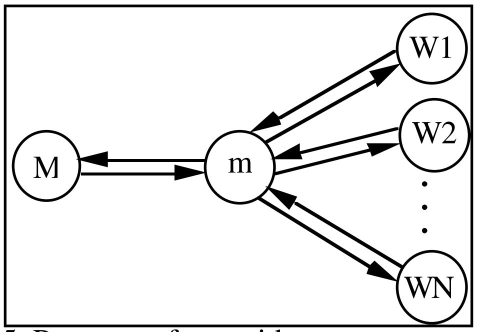

Figure 5. Processor-farm with process management

The process-manager process runs on the master processor (fig. 5). The tasks of the process-manager are the following:

1) Find a free worker process, i.e. a worker that has finished the previous task and returned the results to the master.
2) Choose a task and send it to the free processor found.
3) If a free worker cannot be found then the manager waits until a message from a worker is received.
4) If the message is a result, then it means that a worker has been “freed”, therefore a new task can be assigned to it. The results are stored until all workers finish all tasks.

5) If there are no more tasks then the manager waits for all the workers to finish, collecting the results.

The manager keeps track of the "free" workers by storing information on the master local memory. When a worker returns results, then it means that it is "free". When the manager sends a task to a worker, then this worker is no more "free".

Disk Access

The main memory consultation of the database has the advantage of the fast data retrieval at query-time, because memory access is fast. On the other hand, main memory is limited and thus cannot be loaded with an arbitrarily large database. Furthermore, direct memory consultation has the disadvantage of large initialisation time for the transputers, because each worker process has to copy the entire EDB relation, when this is needed.

To overcome this problem, we have developed direct disk access for the EDB, without altering the processor-farm algorithm. In order to achieve this, we had to alter the meta-interpreter, to transparently include disk access calls for the execution of an EDB predicate call.

The disadvantages of direct disk access in our system can be focused on the concurrent requests of the workers on different relations, that are randomly distributed on the disk. The result is a constant and random disk-head movement, that severely deteriorates the speed-up obtained from the parallel query evaluation.

PERFORMANCE MEASUREMENTS

The system has been developed on a transputer board with five transputers, using CS-Prolog [12] as the implementation language. The transputer configuration involves one master transputer that is attached directly to the host system (a PC) and four slave ones that can communicate with the host system only through the master. Therefore, our platform allowed the existence of only one disk.

We have developed two different kinds of EDB retrievals. The first one pre-loads the relative EDB relations into the main (local) memory of all processors, while the second one directly accesses the EDB partitions on the disk.

The performance measurements were performed on a 8.3 MByte external database, using various sizes for the rule base (maximum 52 clauses). The schema of the sample database is the same as in [6]. We have measured the real time for the completion of the query evaluation for various number of processors, for both algorithms and both types of EDB access.

Main Memory Consultation

Conventional database systems are disk-resident. This is dictated by the volume of data to be stored in the database. Recently, however, the decreasing price for memory chips revealed the possibility for fast, vast and reliable main memory database systems [1, 10]. Since transputers [11] are amenable to direct memory access up to 4 Giga Bytes, we explore here the possibility for an in-memory version of our parallel deductive database system.

The performance of the system when the EDB is consulted in main memory is shown fig. 6. The first case (on the right) is for the sequential algorithm. The second and the third cases are the processor-farm algorithm with 3 and 5 processors, respectively. Finally, the last case is the controlled processor-farm algorithm with a process manager and 4 processors.

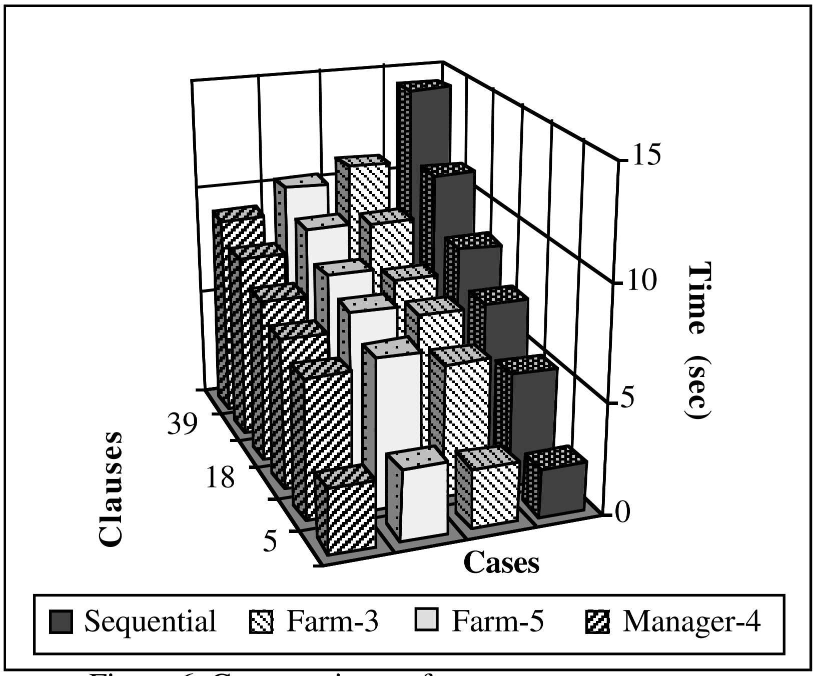

Figure 6. Comparative performance measurements

The results exhibit the following behaviour. For a small number of resolved clauses, the sequential algorithm is the fastest. This happens because the initialisation phase of the worker processes takes a long time. Hence, the speed-up obtained by the parallel execution does not compensate for the initialisation time.

As the number of clauses increases the parallel cases (i.e. cases 2, 3 and 4) perform better than the sequential. The use of a manager in case 4 is faster than the simple processor-farm cases with either 3 or 5 processors, because it preserves a balanced workload among the workers.

Direct Disk Access

The database performance with direct disk access is generally worse than in-memory access (fig. 7). This is due to the different response time of the two media. However, for a small number of clauses, the initialisation time for the in-memory consultation is a large fraction of the overall query completion time, therefore, the direct disk access performs better. The test cases of fig. 7 have been conducted using the processor-farm algorithm without process management.

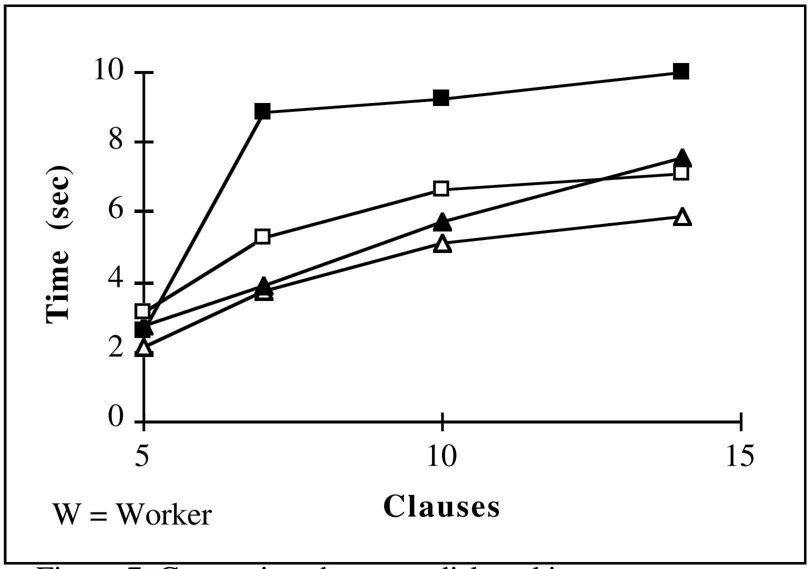

Figure 7. Comparison between disk and in-memory access

The main advantage of disk access is that there are no limits to the size of the database. Since in our hardware configuration each worker transputer has only 1 MByte of memory, the system with main-memory consultation cannot be scaled-up to a very large database.

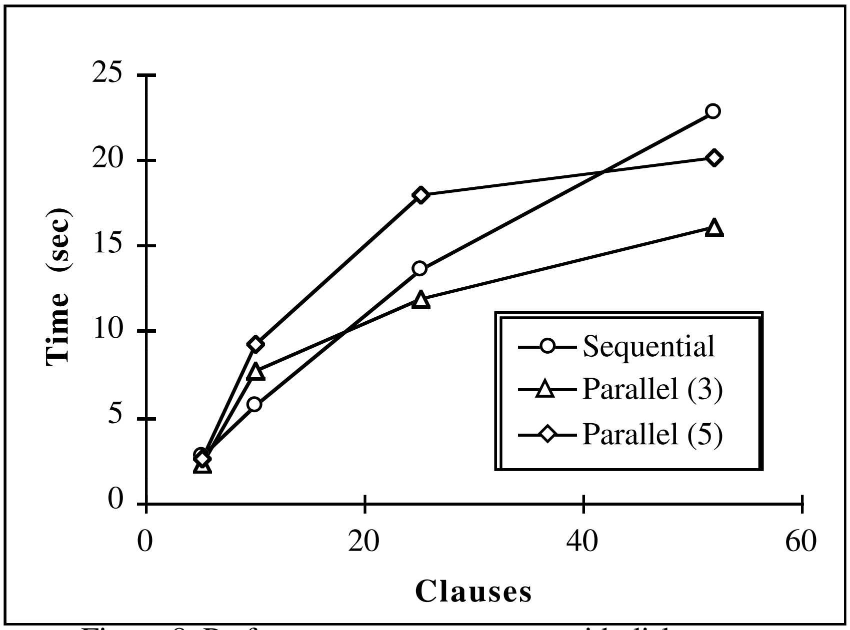

Figure 8. Performance measurements with disk access

A noticeable finding on the behaviour of disk access (fig. 8) is that the increase on the number of processors that execute in parallel does not always decrease the query evaluation time. This is due to the fact that only one disk is used, therefore the concurrent (and randomly distributed) requests from many processors to only one disk, clash the disk head. The optimum performance has been measured for about 3 processors.

A better solution would be to increase the number of disks in the system and the best would be to attach one disk to each worker. Still the system could operate sufficiently well with one disk, if the master processor played the role of a disk caching device. Then requests of the workers would be directed to master's local memory instead of disk. A background process running on master would cache EDB relation clusters from time to time.

Speed-up Issues

So far, we did not examine carefully the term “speed-up”. Speed-up is defined [9], as the fraction:

$$S = \frac{T_{par}}{T_{seq}}$$

where T_{par} is the query completion time of the parallel algorithm on the multi-processor system and T_{seq} is the equivalent time of the sequential algorithm on the same system.

Speed-up is usually defined as a function of the number of processors used in the multi-processor system. In order to plot this function, we measure the query completion time of the same query for a database of a constant size, but with an increasing number of processors. The ideal speed-up is linear with the number of processors, e.g. when the processors are doubled, the query completion time should be halved.

The speed-up for the main-memory access system and using the processor-farm algorithm without process management is plotted in fig. 9. Our system has only five transputers, therefore speed-up curves are measured for only up to five processors used. Measurements for various number of clauses (but for the same size database) have been conducted; fig. 9 contains four representative ones.

For a small number of clauses, speed-up is negatively linear, i.e. decreases with the number of workers. This is explained by the large initialisation time, which gets larger for more processors. For a medium number of clauses, speed-up increases at first (as it is natural), but again the large initialisation time outperforms the performance gain from sharing the computation load. Finally, for many clauses speed-up increases with the number of processors, because the performance gain by sharing the load among the processors is greater than the initialisation time. But again, the speed-up is not linear. For even more clauses, the speed-up behaves linearly for a large range of processors.

The speed-up curves for the rest of the combinations of database accesses and the presence of process manager exhibit similar properties, for reasons already explained. Disk access avoids the large initialisation time for each worker, but the single disk acts as a bottleneck to the performance of the system.

Therefore, it is clear that in order to obtain linear speed-ups the system must be expanded with multiple disks, or in failure to do so, clever caching techniques must be employed, to increase the I/O throughput of the system.

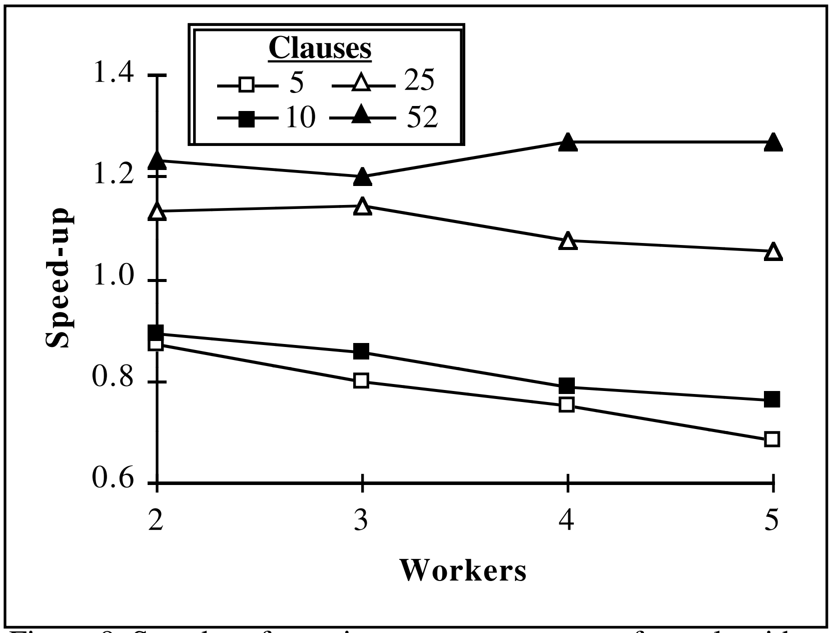

Figure 9. Speed-up for main-memory, processor-farm algorithm

CONCLUSIONS

This work has tried to unify logic programming and databases through a deductive database management system. The advantages of such an integration have been identified long ago [4, 6, 13]. Problems arising from the slow execution of deductive database queries have been tackled using parallelism.

The user query is transformed into Disjunctive Normal Form, which consists of independent clauses that can be executed in parallel. That means we exploit the inherent OR-parallelism of logic [7, 16], at the IDB level. The independence of disjuncted clauses requires that some work may be redundantly duplicated at the different processors, but this will hopefully lead to faster query execution.

The parallel execution of clauses is performed through the processor farm algorithm with and without management. The above algorithms have been fully tested and the latter always outperformed both the former and the sequential execution.

Furthermore, disk access of multiple processors on a single disk seemed to offer only minor speed improvement over the sequential execution, due to disk "thrashing". Future work will include the simulation of multiple disks attached to workers, as well as OR-parallelism at the EDB level (data partitioning, [7, 9]).

Acknowledgments

This work has been supported by the Aristotle University of Thessaloniki, under the contract 7540. The first author was supported by a grant from the COMETT program. The second author is supported by a scholarship from the Greek State Scholarships Foundation (I.K.Y.).

References

[1] P.M.G. Apers et al., "PRISMA/DB: A parallel main memory relational DBMS," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 4, pp. 541-554, Dec. 1992.

[2] N. Vasiliades and I. Vlahavas, "LBASE: A Logical Database Management System," *Proc. 1st Conf. Balkan Physical Union*, Thessaloniki, 1991, pp. 640-642.

[3] N. Bassiliades, "Study and Development of a Parallel Database Management System, using a Fuzzy Query Language," TR/AUTH-7540, Sept. 1993.

[4] D.A. Bell, J. Shao and M.E.C. Hull, "Integrated deductive database system implementation: A systematic study," *The Computer Journal,* Vol. 33, pp. 40-48, Jan. 1990.

[5] J. Bocca, "EDUCE: a marriage of convenience: Prolog and a Relational Database," *Symp. on Logic Programming*, New York:IEEE, 1986, pp. 36-45.

[6] C.L. Chang, "DEDUCE 2: Further investigations of deduction in relational data bases," in *Logic and Data Bases*. New York: Plenum Press, 1978, pp. 205-210.

[7] J.S. Conery, *Parallel Execution of Logic Programs*, Kluwer Academic Publishers, 1987.

[8] H.M. Dewan et al., "Incremental database rule processing in PARADISER," *Journal of Intelligent Information Systems*, Vol. 1, pp. 177-209, June 1992.

[9] D. DeWitt and J. Gray, "Parallel database systems: The future of high performance database systems," *CACM*, Vol. 35, pp. 85-98, June 1992.

[10] H. Garcia-Molina and K. Salem, "Main memory database systems: An overview," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 4, pp. 509-516, Dec. 1992.

[11] Inmos Ltd., *The Transputer Data Book*, 1988.

[12] P. Kacsuk and I. Futo, "Multi-transputer implementation of CS-Prolog," in *Proc. AI and Communication Process Architecture*, Wiley & Sons, 1989, pp. 131-148.

[13] R. Kowalski, "Logic for data description," in *Logic and Databases*. New York: Plenum Press, 1978, pp.77-103.

[14] C. Maciazek, "Parallel Management of Large Data Bases in a Distributed Environment of Transputers," BEng Thesis, IRESTE, University of Nantes, France, June 1993.

[15] D.S. Moffat and P.M.D. Gray, "Interfacing Prolog to a persistent data store," *Proc. 3rd Internat. Conf. on Logic Programming*, 1986, pp. 577-584.

[16] S. Taylor, *Parallel Logic Programming Techniques*, Prentice Hall, 1989.

[17] J.D. Ullman, *Principles of Database and Knowledge-Base Systems, Volume II: The New Technologies*, Computer Science Press, 1989.

[18] O. Wolfson, “Sharing the load of logic-program evaluation,” *IEEE Data Engineering*, Vol. 12, pp. 58-64, Mar. 1989.