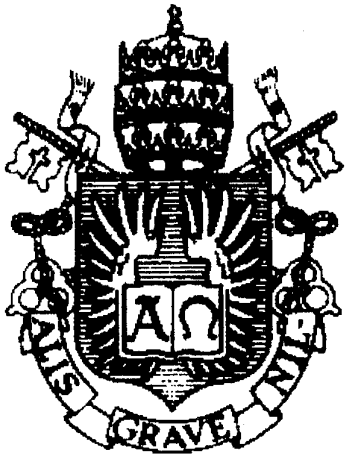

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 09/95

Practical Behavior of Parallelization Strategies for Datalog

Sérgio Lifschitz
Rubens N. Melo
Esther Pacitti

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, N° 09/95

Editor: Carlos J. P. Lucena April, 1995

Practical Behavior of Parallelization Strategies for Datalog *

Sérgio Lifschitz

Rubens N. Melo

Esther Pacitti

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

Practical Behavior of Parallelization Strategies for Datalog

Sérgio Lifschitz Rubens N. Melo
Esther Pacitti
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brasil
e-mail: {lifschitz,rubens,esther@inf.puc-rio.br}

PUC RioInf.MCC09/95

Abstract

We present in this paper a behavior study on a very important class of known parallelization strategies for evaluating Datalog, namely, the bottom-up rule instantiations partitioning paradigm. Its basic algorithm specialization is observed and many different variations are also verified in order to obtain a comprehensive set of implementation results. We show that the usually considered analytical models may not explain the actual behavior of the algorithms. We make careful observations on the impact of some of the factors that influence the behavior of the algorithms. Particularly, important issues related to inter-sites data transfers are analyzed and the practical results obtained show that this is clearly a fundamental factor to achieve acceptable performances.

Keywords: Deductive Databases, Datalog, Parallel Evaluation, Behavior Analysis.

Resumo

Neste trabalho é realizado um estudo prático de comportamento para uma classe importante de estratégias paralelas usadas na avaliação de consultas Datalog. A classe aqui considerada se baseia no paradigma de instanciações de regras com avaliação *bottom-up*. O algoritmo básico implementando o paradigma é analisado e algumas variantes são averiguadas com o objetivo de se obter resultados práticos que permitam observações precisas sobre seu comportamento. É mostrado que os modelos analíticos normalmente considerados podem não explicar adequadamente o real desempenho obtido na prática. Além disto, são comentados em detalhes outros fatores que podem influenciar na performance dos algoritmos. Especificamente, são analisados aspectos relacionados à comunicação de dados entre os *sites* envolvidos no processamento. Os resultados obtidos mostram claramente que este fator é fundamental para que os algoritmos apresentem um bom desempenho.

Palavras-Chave: Bancos de Dados Dedutivos, Datalog, Avaliação Paralela, Análise de Comportamento.

Practical Behavior of Parallelization Strategies for Datalog

Sérgio Lifschitz Rubens N. Melo
Esther Pacitti
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brasil
e-mail: {lifschitz,rubens,esther@inf.puc-rio.br}

PUC Rio Technical Report MCC09/95

Abstract

We present in this paper a behavior study on a very important class of known parallelization strategies for evaluating Datalog, namely, the bottom-up rule instantiations partitioning paradigm. Its basic algorithm specialization is observed and many different variations are also verified in order to obtain a comprehensive set of implementation results. We show that the usually considered analytical models may not explain the actual behavior of the algorithms. We make careful observations on the impact of some of the factors that influence the behavior of the algorithms. Particularly, important issues related to inter-sites data transfers are analyzed and the practical results obtained show that this is clearly a fundamental factor to achieve acceptable performances.

1 Introduction

Deductive databases theoretical issues have been studied extensively in both databases and logic programming research communities and its potential and actual applications have already come to date, such as exploratory data analysis, enterprise modeling and scientific databases [Tsu91, GRS95]. However, recursive Datalog programs evaluation is expensive and, to become practical, many strategies have been proposed and compared [DR94]. Among the possible solutions, parallelism has been considered the most promising one towards acceptable performances. It has been recognized that parallel processing holds the key to answering the performance requirements for the projected increases in data size and complexity of queries in the next generation database applications [DG92]. Many algorithms have been proposed for its processing in parallel, in particular for transitive closure, a special case of Datalog queries [CCH93].

Although the applicability of paradigms have been studied in detail, almost nothing has been done in order to developing an actual behavior study of the algorithms implementing the proposed strategies. This information would be interesting for the construction of an optimizer or to guide *ad-hoc* deductive systems implementations. Existing works on this subject mainly

discuss the expected good properties independently of other similar proposals. Except for a few transitive closure parallel strategies and particular implementations on special multiprocessor architectures (*e.g.* [HWF93]), estimations with respect to the potential speed-up that could be obtained are still not available. Furthermore, very little is known about the impact of the various parameters - such as number of sites, inter-processors communications strategy and data fragmentation policy - with respect to the actual performance of the proposed algorithms.

We present here a study on the practical behavior of an important class of parallel strategies for arbitrary Datalog programs, namely, the bottom-up rule instantiations partitioning paradigm [GST90, WO90]. To our knowledge, there is only one previous implementation study of algorithms based on the referred paradigm [WZB+93]. However, this work is mainly concerned with a particular query (the partially instantiated transitive closure query) and, once more, the analysis is based on non realistic assumptions, such as an even workload partitioning among processors. We look up further on implementations here. We start by implementing the algorithm based on the original ideas of [GST90] and observe the behavior and performance in many different aspects. The results obtained from the implementations give us a more comprehensive understanding of the proposed algorithms.

We look carefully at some of the parameters involved, such as the input data, the number of available parallel sites, the inter-processors communications model and their influence on the practical behavior of the algorithms. Indeed, we discuss the communication model given in [CLP94], where a coordinating node for distributing and *filtering* data transferred between the parallel sites is proposed as an alternative to the standard completely distributed model. We implement this strategy and compare its speedup and overall behavior with respect to the previous implementations. Some other experiments are done aiming at a further study of the algorithms behavior, *e.g* whether to transmit the derived facts local to each site at each iteration or at each local fixpoint. As we shall see, performance is directly dependent on this and many other factors, and some of them are difficult to quantify.

A challenging open problem is to define appropriate measurements for the performance of parallel evaluation strategies. In this regard, very few results are known. The performance analysis referenced in previous works (*e.g.* [VK88]) are mainly done using analytical simulation. Many simplyfing assumptions are made, such as uniform production of tuples, data transfers that do not saturate the network and existence of a fixed amount of time for sending a bucket of tuples from one side to another. Clearly, these and other similar considerations do not happen in practice and, consequently, the results of these studies will not reflect the actual behavior of systems implementing these strategies. Particularly, we show that the speedup analysis considering cost models that count joins or productive rule instantiations may not explain correctly the actual performances.

All experiments were executed in a parallel shared-nothing machine, the *IBM SP/1*, using *PVM* [GBD+94] and *C* code. This fact has permitted us to observe not only the processing of the queries but also, and more importantly, the price paid by the actual behavior of the evaluation with respect to inter-sites communications. It should be noted that our goal here is not to determine the best parallel algorithm neither to obtain a complete performance analysis for the bottom-up rule instantiations partitioning paradigm. Instead, we have done a large number of experiments aiming at detecting which factors are really important to the behavior of the parallel algorithms implementing the paradigm in order to become practical.

The remainder of the paper is organized as follows. In the next section, we present an overview of existing parallel strategies for Datalog evaluation. In Section 3, we briefly describe the hardware and software configurations used to implement the algorithms and the main parameters considered. The rule instantiations partitioning paradigm and its specialization into an algorithm are described in Section 4, together with a first set of implementation results and corresponding analysis. The centralized communications approach is presented in Section 5 and comparative experimental results are also given. Within this section we discuss other important factors that may influence the algorithms. Related work with respect to behavior analysis of Datalog and transitive closure queries are mentioned in Section 6 and the final observations and suggested extensions of this work appear in Section 7.

2 Parallel strategies for Datalog evaluation

In this section, the most important research works on the parallel evaluation of Datalog queries are described. We assume familiarity of the reader with basic logic programming and deductive databases concepts. Preliminary studies were concentrated on determining whether or not a Datalog program could be evaluated in parallel without communication among sites, which is referred to as *pure parallelization*. These results provide syntactic characterization of Datalog programs for which pure parallelization works. Such programs are called *sharable* [Wol88] or, in the case of disjoint results at each site, *decomposable* [WS88]. Sufficient syntactic conditions for decomposability have been provided [CW89, WS88, SL91]. However, decomposability is only applicable - in the sense of completeness - to a restricted class of programs and the property of being decomposable was shown undecidable for arbitrary Datalog programs [WO90]. A different pure parallelization approach for the evaluation of Datalog queries is discussed in [LV95]. It is shown that the evaluation is complete (*i.e.*, the result is entirely computed) for *almost all* (which is given a precise probabilistic meaning) inputs, without any considerations on the syntax of the program rules.

Thus, if one wants to guarantee completeness of the parallel evaluation for *all* Datalog programs, *inter-sites communication* must be considered. From a theoretical point of view, it has been studied whether a Datalog program belong to the *NC* complexity class, so that it could be evaluated in polylogarithmic time given a polynomial number of sites in the size of the database [Kan88, UvG88]. However, the number of available sites is limited in practice and these algorithms have to be adapted to fewer sites. This can be done by assigning the work of multiple processors to a single one, which can be harmful to the overall evaluation performance.

Assuming a constant number of sites, some other methods have been proposed. In general, these approaches follow two different paradigms. One is the so called *rule instantiation partitioning* strategy [GST90, WO90, ZWC91], which is a pure bottom-up method that parallelizes the evaluation by assigning subsets of the rule instantiations of the program among the sites, such that each site evaluates the same original program with less data. This scheme results in a non-redundant computation, in the sense that the same firing is never done by two distinct sites.

Another possibility is the so called *rule (or clause) decomposition*, where the rules of a program are evaluated in a top-down style, employing *sideways information passing* to focus

on relevant facts [vGel86, Hul89], sometimes considering a pipelined strategy [BSH91]. The rules are decomposed into concurrent modules that are assigned to distinct sites. An important drawback for rule decomposition is that the degree of parallelism is, in general, dependent upon the structure of the program, which means that even if more sites are available, they will not be used. Moreover, if each processor evaluates a different part of the rule (which corresponds to one or a conjunction of many subgoals), an unbalanced workload may occur. A *mixed* rule instantiation partitioning and rule decomposition is proposed in [SBH91] in order to address some of these problems. However, some synchronization among sites is still present and the actual implementation is very complex. Finally, the underlying idea of a backward-chaining-like evaluation is still not practical for arbitrary queries. If no constants are present, these methods are clearly not efficient or even not applicable.

Therefore, our emphasis here will be on the parallel bottom-up evaluation of Datalog program. In the presence of constants, optimization techniques like the *magic sets* rewriting method can be used orthogonally to parallelization [Ull89] yielding better results. We will describe in Section 4 the basic algorithm considered throughout the paper. First we present both the system configuration and software used for the experiments and also the main parameters considered for the behavior analysis.

3 Implementation environment and parameters

We have done all implementations on the IBM 9076 SP/1 (Scalable Parallel) machine. It is a shared-nothing parallel configuration supporting SPMD (Simple Program Multiple Data) program applications, with its basic configuration of up to 16 nodes, each with its own RISC 6000 processor, 256Mb of memory and 2Gb disk, giving a total processing power of 2 Gigaflops. The shared-nothing architecture seems to be well adapted to support very large databases, specially with respect to interference and scalability issues [Sto86, Val93]. Unfortunately, we could not use any of the RDBMS already running on this machine at the time we have done our experiments and all relational operators needed were directly implemented in *C* code for each node.

The chosen strategies were implemented using PVM (Parallel Virtual Machine) version 3.3, a public domain software developed at the OAK Ridge National Laboratory [GBD+94]. The main goal of this tool is to provide support for distributed and parallel application programs over a (maybe heterogeneous) network of workstations and interconnection protocols. In our case, we have considered TCP/IP for the inter-site communications. Its flexibility have permitted us to easily implement all of the strategies making calls to PVM library functions within the algorithm written in *C* code.

Among the parameters considered for the behavior analysis, the main ones are the Datalog queries, the input data, the chosen number of parallel sites and the data transmission criterion. Even if the parallel strategies are applicable to arbitrary Datalog queries, we have decided to consider the full linear transitive closure one (no variables instantiated) P^1

$$\begin{aligned} T(x,y) :- \quad & T(x,z), A(z,y). \\ T(x,y) :- \quad & A(x,y), \end{aligned}$$

defined by a single linear recursive rule and also its nonlinear version P^2

$$\begin{aligned} T(x,y) :- \quad & T(x,z), T(z,y). \\ T(x,y) :- \quad & A(x,y), \end{aligned}$$

which was shown not to be decomposable in [WS88], so to be able to make some comparisons with the specific parallel algorithms for the transitive closure query. Thus, binary relations were used as input data and the corresponding graphs were randomly generated, given a fixed number of nodes (with values also randomly generated) and a fixed edge probability between any two nodes.

We have constructed completely random graphs, DAGs, binary trees and lists, as suggested in [CCH93]. In the binary tree case, we have chosen the tree height as a parameter and for lists, the number of nodes. In both cases the existence of an edge is guaranteed (fixed topology) and only the node labels are randomly generated. The chosen number of nodes ranged from 200 to 1,000, with edge probabilities taken between 0.5% and 10%. The size of the base relations that were generated ranged from 400 to 10,000 tuples, with corresponding closure containing up to 300,000 tuples. Real data such as the french parisien subway graph (710 tuples and 85,000 in its closure) were also used as input.

Theoretically, the algorithms may be executed in as many parallel sites as we want. In our case we have run our experiments on all sites (up to 16) available. We could as well simulate more than one logical site for each physical one but we have decided not to do it as we were concerned with the speedup measured by the actual response time. It is worth saying that all previous experimental works were done on no more than 6 parallel sites. Many different situations were tested but we have mostly tried out 4, 7, 10 and 14 sites. Finally, concerning the data transmission criterion, it depends on the underlying communication model, which will be discussed together with the implemented algorithms in the next sections.

4 Basic parallel algorithm

In [GST90, WO90] parallel strategies based on the bottom-up rule instantiations paradigm are proposed, applicable to arbitrary Datalog programs, with more than one (linear and non-linear) recursive rule and more than one recursive predicate. They introduce the notion of restricting (or discriminating) predicates, that are used to partition the instantiations of the program rules among a set of sites. The basic idea is to append some evaluable predicate f_i to the body of each rule, where f is usually a hash-based function and i is the identification of some site, such that each instantiation of the rule is assigned to a unique site. In [ZWC91], the program is first rectified and multiple partition functions are appended to each rule. This defines also a fragmentation of the input data.

Thus, we construct distinct restricted versions P_i of a Datalog program P and each site will execute a distinct subset of the complet set of P rule instantiations. All derived facts in a site s_i are kept within its local database and are also sent to all other sites s_j $(j \neq i)$, with respect to a given data transmission criterion. The parallel evaluation finishes when all sites reach their local fixpoint and no fact is being transmitted from a site to another. The main steps of the algorithm executed at each site are shown in the next figure:

Execute Procedure *Initialization* ;
While global termination condition not reached:
    Execute Procedure *Evaluation*;
    Execute Procedure *Transmission*;
    Execute Procedure *Reception*;
End While
Execute Procedure *Final Pooling*;

Figure 1: Rule instantiations partitioning parallel algorithm

The *Initialization* procedure corresponds to the initial distribution of the input database and the firing of all exit rules. The main loop contains a procedure *Evaluation*, which executes one complete instantiation of all recursive rules of the restricted version of the program. A seminaive-like algorithm is usually considered, where at the end of each iteration the set of new facts produced are available for inter-sites communications. These are done by procedures *Transmission* and *Reception*, where the data transmission sets are sent to, and received from, all other parallel sites. When a global termination condition is reached, each site sends their local results to a given site that will contain the final result. The algorithm *PSNE*, presented in [WO90], gives a more detailed presentation of this strategy.

4.1 Example of algorithm application

Suppose that n sites are available and that one site can exchange data with any other site during the whole evaluation process. If one considers the strategy proposed in [GST90] and apply it to the Datalog program P^2 described in the previous section, some evaluable subgoals are appended to each rule and n restricted versions P_i^2

$$\begin{array}{ll} r_{1i}: & T(x,y) :- \quad T(x,z), T(z,y), z \bmod n = i. \\ r_{2i}: & T(x,y) :- \quad A(x,y), y \bmod n = i. \end{array}$$

of P^2 are determined, where each P_i^2 is evaluated in a distinct site s_i $(0 \leq i \leq n-1)$.

Each site s_i starts evaluating the restricted program P_i^2, performing rule instantiations *wrt* its local database. The new facts obtained by productive instantiations are added to the local database and are also transmitted to each other site s_j, $j \neq i$, according to a given transmission criterion. These data transmissions are done at each iteration, which means in this case, at each firing of the recursive rule. The parallel evaluation terminates when no restricted program in some site can compute a new fact and when there are no facts being transmitted among sites. The data transmission criterion from site s_i to site s_j in this case is defined as follows: a fact $T(a,b)$ belongs to the transmission set if $a \bmod n = j$ or $b \bmod n = j$.

The design of the restricted versions of the same program P^2 as proposed in [ZWC91] is based on an initial rectification of P^2 rules, followed by the definition of evaluable subgoals

local to each ordinary subgoal in the rules as given by

$$r_{1kl}: \quad T(x,y) :- \quad T(x,z), T(z_1,y), z = z_1, z \bmod n_1 = k, z_1 \bmod n_2 = l.$$
$$r_{2m}: \quad T(x,y) :- \quad A(x,y), y \bmod n_3 = m.$$

If we have $n_1 = 2$ ($k \in \{0,1\}$), $n_2 = 3$ ($l \in \{0,1,2\}$) and $n_3 = 6$ ($m \in \{0,1,2,3,4,5\}$), there are six restricted versions of each rule. They are combined in all possible ways, pooling them together so that only one restricted version of a rule is assigned to each distinct site. For the data transmission criterion, we must check for each given fact on site s_i if it satisfies a local evaluable subgoal associated to an ordinary subgoal relative to this fact. Independently of which pooling of restricted rules has been done, a fact $T(a,b)$ is said to be *potentially* useful for the instantiations of rule r_{1kl} if $b \bmod 2 = k$ or $a \bmod 3 = l$. This means that it is not guaranteed that the (entire) rule instantiation at the receiving site s_j is applicable with the new fact sent from s_i. Although the test of *utility* of fact *wrt* a given rule is claimed to be more efficient in [ZWC91], the design of the restricted versions of the program is clearly more complicated than the one in [GST90]. For this reason, only the latter will be further investigated in this work.

4.2 Experimental results

We discuss next some implementation results on the algorithm just described. A set of interesting and meaningful information that may backup our conclusions and observations over the actual behavior of the algorithm is presented. The input data groups that will be considered for showing the results are given in Table 1. These were chosen with respect to their similar practical behavior observed from the experiments.

Input Group	Nodes Range	Base Tuples	Derived Tuples
A	200 → 300	400 → 5,000	1,800 → 38,000
B	300 → 700	3,500 → 9,000	20,000 → 210,000
C	700 → 1,000	5,000 → 10,000	135,000 → 320,000

Table 1: Random Graphs and DAGs

An important observation is related to the chosen queries. As we have mentioned in section 3, we have executed both linear and non-linear versions of the transitive closure, basically over the same inputs and number of sites configuration. However, except for the number of redundant derivations (which is bigger for the non-linear transitive closure case, as in the monosite execution), all other behavior considerations are very close to the ones that will be shown for both queries. Other input structures, such as binary trees and lists, will be studied in further sections.

We show in table 2 interesting informations about the behavior of the algorithm for each input data group. First of all, we give some results on the observed response time speedup obtained. We have basically considered two situations, one for relativelly small number of sites (4 to 8) and for bigger number of sites (9 to 15). For example, when we execute the algorithm for a number of sites close to 4, we get in all cases an average (close to linear) speedup of 3 and when we get closer to 8 parallel sites, the speedup ranges from 5.3 to 6.5. For executions

Input Group	Speedup 4 → 8 sites	Speedup 9 → 15 sites	Iterations Range	Site Activations	Useless Transmissions	Useless Derivations
A	2.9 → 5.3	8.6 → 9.8	146 → 351	26 → 117	35% → 70%	26% → 85%
B	3.3 → 6.5	5.0 → 9.2	741 → 2389	117 → 867	50% → 90%	46% → 86%
C	2.6 → 5.3	5.3 → 11.6	221 → 833	49 → 279	55% → 83%	11% → 54%

Table 2: Basic parallel algorithm behavior

with 15 sites, we have obtained speedups around 10, with the best speedup reaching 11.6, not to close to the expected linear behavior. These results on speedups may be better observed with the graphical representation of figure 2.

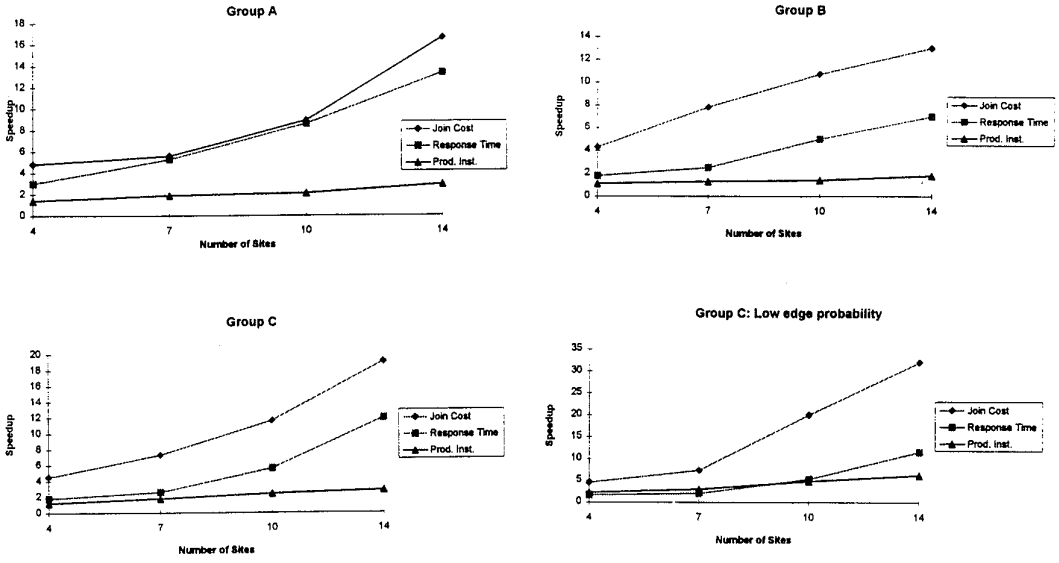

Figure 2: Response time, join and productive instantiation speedup

The results given in figure 2 are relative to chosen executions that represent the average behavior for the input data groups A, B and C. We show not only the speedup obtained by measuring the response time but also some computed results based on previous analytical models, such as the join and the number of productive rule instantiations costs. These will be discussed later in this section. As observed in figure 2 the response time speedups increase when the number of sites increases but not always in the same rate. This happens, for example, when running the algorithm for up to 8 sites. For more than 8 sites, we get results closer to a linear speedup, except for input data group B. This might be explained by the time consumed with useless transmissions and fact derivations, as shown in table 2.

Useless communications and productions

In fact, an important issue related to the communication cost is the rate of tuples transmitted from a site to another that were discarded by the receiving site. These transmissions are called *useless*, in the sense that they should be avoided as the local site evaluation loses time eliminating them so to duplicate rule instantiations. Indeed, there may be from 50% up to 90% of useless transmissions during the complete evaluation process for input group B.

It is also known that the distinct firing of rules paradigm does not guarantee that distinct facts cannot be derived in two (or more) different sites. Ideally, an optimal parallel execution would equally divide the productive instantiations among sites. Our experiments have shown that this desired situation is far to be achieved. Even if for some inputs, such as input groups A and C, the facts derivations overhead is no more than 25% of the final output, there are some cases where these useless derivations correspond to almost twice the number of all tuples that should be produced, as the 86% rate for group B on a large number of parallel sites.

Another information given in table 2 concerns the number of iterations (rule firings, or joins in our case) and the number of site activations (number of times the sites have reached a local fixpoint during the complete parallel evaluation). We show some extreme examples for each input data group that illustrate a clear uneven workload balancing among sites during the evaluation process. As an example, for input group B there are some experiments where one of the sites make 3 times less iterations than the other (varying from 741 to 2389) and where the number of sites activations may be up to 6 times different (117 to 867). It shoud be noted that for all the results shown in this section the input graphs tested were randomly generated on uniform data distribution and that each site gets an approximately equal number of tuples at the outset.

Effectiveness of analytical models

The analytical models mainly considered in previous works usually study the speedup of the parallel processing strategy with respect to two factors: the amount of computation, *e.g.* the number of joins and the size of relations involved, and the amount of communication among the processors, which includes the processing of messages before transmission and after reception of data sets. Each one of these factors depends on the parallel architecture, specially the number of sites available, and both the size (nodes and tuples) and topology (cyclic, acyclic, trees...) of the input graph.

When we look back at Figure 2, we see that by studying the speedup behavior of the algorithm by computing the join cost we may get similar relative results compared with the response time speedup. In some cases, even equivalent absolute values, as for input group A. However, the computed speedup is, in general, bigger than the actual speedup measured considering only the response time. As observed in figure 2 for an input of group B, all join costs speedups curves show better results with respect to the response time speedup. Even worse, for input groups A and C the join cost would give us more than linear speedup, as the speedup over 30 for a chosen input in group C (1000 nodes and low edge probability of 0.5%) and 14 parallel sites, when the actual speedup is close to 10. On the other hand, we have obtained some results where comparing only the cost of processing the joins we get practically

no speedup or a speedup less than 1, as shown in figure 3 for an input group B and less than 10 sites.

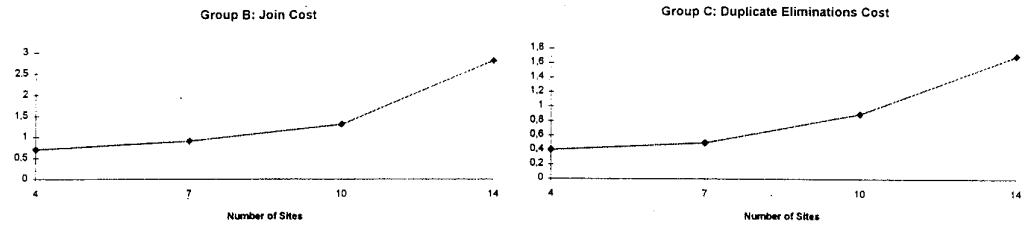

Figure 3: Join and duplicate elimination speedups

Actually, when seminaive-based algorithms are being used, the cost of duplicate eliminations made by relational difference operators should also be taken into account for behavior analysis. We have observed that the computed difference costs are very close to the computed join cost in many cases. Anyhow, even if both costs are computed, we may still have a speedup below 1. In an example obtained from our experiments for an input chosen in group C (a DAG with 1000 nodes and 1% edge probability), shown also in figure 3, we get no speedup (less than 1) when running on no more than 10 parallel sites. Furthermore, even if some speedup is obtained, as when running the precedent example with 14 sites, the speedup computed for the joins and duplicate elimination cost is equivalent to 1.8 and the actual speedup is 11.9 (Figure 2, input group C), about 6 times bigger.

Another computation cost referred in previous works is obtained by measuring the number of productive rule instantiations (successful rule firings) during the whole execution. We have computed this cost in our experiments and, once more, the conclusion is that also this cost cannot be effectively used for comparison purposes. Indeed, as observed in figure 2, we get the same speedup when the number of sites vary from 4 up to 14. For up to 7 sites, the productive instantiations cost gives a speedup that is approximately the same as the one computed measuring the response time. For the particular case of the low edge probability input group C we get close results even for a larger number of sites. However, this situation does not happen in general, and counting productive rule instantiations should be carefully considered for the performance evaluation.

Finally, it should be noted that some other assumptions made when using the referred analytical models for studying the performance of algorithms evaluating Datalog queries are in general not verified in practice. Indeed, there is no uniform facts derivation rate during the whole execution at each site. For some inputs (*e.g*, dense graphs), the intensional database tuples are produced mainly during the first iterations, with very few derivations close to the algorithm termination. Also, a balanced workload is rare. We have run experiments where, even for graphs randomly generated over an uniform data distribution, the computation cost at each site may vary considerably.

5 Centralized communication approach

In [CLP94], an alternative scheme for the inter-sites data communications is proposed, motivated by the observations made in the previous section about useless transmissions. If only satisfaction of evaluable subgoals is considered as the main test for the data transmission criterion, it is known that there will be certainly redundant and useless data transmitted among sites. Although the local redundant computation can be avoided, nothing is done in order to avoid the transmission of these useless facts. It may occur that facts of a given transmission set are all eliminated at the receiving site. It is worth saying that in [AJ88] the authors were already asking if better performances could be obtained if one reduces the computation cost and adds extra evaluation process control.

Therefore, rather than a completely distributed inter-sites communication scheme, one of the sites will assume the responsibility for the coordination of the data communications. In general, this site does not participate directly on computations, acting basically as a *filter* for data transmissions. The evaluation is parallelized with the same underlying idea discussed in the previous section, *i.e*, the partitioning of the rule instantiations among a set of sites that evaluate the same original program but with less data. A shared-nothing architecture is assumed, where communications are only required (and allowed) between the coordinating site and all others processing nodes in both directions. This is clearly a simpler configuration and extensibility in this case is straightforward.

The main goal of this alternative approach is to allow the definition of a more efficient data transmission criterion, which would reduce both the number of data receptions at each processing site and the total amount of data involved in communications during the whole execution. Also, redundant and useless data transmissions would be avoided and the test of whether to transmit a data set from (to) the coordinating node to (from) the processing nodes could be efficiently tested. Compared to the completely distributed scheme, the quality and amount of information available at the central site is richer than the one in any of the parallel sites. Therefore, the coordinating node may decide *what* and *where* send data using the information of data received from *all* sites. The correctness of the proposed algorithm and data transmission criteria are easily verified. The decision of not sending a given tuple from the coordinating site to all other sites and vice-versa is based on communications issues, never on possible program instantiations. The data transmission criterion defined guarantees that all tuples that may be useful to one of the program rules restricted versions are transmitted *at least* once. As redundant communications are avoided, these tuples are sent *at most* once between the central site and all other sites.

Unfortunately, all the nice features obtained by considering the a central coordinating node for the inter-sites communications may not always lead to an expected good performance. There is a risk of a bottleneck at the central site, as the tasks that should be assigned to distinct sites working in parallel are performed by only one of them. For example, a critical situation might happen whenever too many processing sites are idle at the same time and the termination condition has not yet been reached. In this case, an important number of processing sites keep on asking data for the coordinating node so to continue their local evaluation and the central site must determine many transmission sets to be transmitted to the processing sites. The results from the implementations to be presented next will be useful to observe the actual behavior of the algorithm in these and other practical situations.

5.1 Implementation results

In the following, we will refer to the basic parallel algorithm as the *distributed* one and the algorithm discussed in this section as the *centralized* one.

Speedup

In table 3 we show the results obtained for the same input groups as before but not the same behavior observations. On one hand, it should be noted that the new results concerning the useless derivations are equivalent to the ones shown in table 2 for the distributed algorithm. Thus, they will be ommited. On the other hand, we can make some comparisons that we could not do before, such as the rate of the total number of transmissions and its volume during similar executions, which are related to the total communication cost. Finally, as a direct consequence of the chosen model of a central coordinating node for inter-sites communications, there are no useless transmissions here.

Input Group	Speedup		Iterations Range	Site Activations	Number of Transmissions	Messages Volume
	4 → 8 sites	9 → 15 sites				
A	4.0→ 5.6	5.9 → 6.8	17→ 51	2 →16	15.7 → 30.2	1.4 →1.5
B	3.3 → 8.8	2.0 →8.8	96→171	12→36	2.3→ 21.5	1.2 → 2.6
C	3.8 → 4.6	5.6→ 7.6	46→404	8→96	15.3→ 89.8	1.4→ 1.7

Table 3: Centralized Algorithm

Comparing the speedups obtained in tables 3 and 2, we can observe that, for executions with less than 8 sites, the centralized algorithm performs better than the distributed one in almost all cases. However, for large and highly connected input graphs, the distributed algorithm may have better speedups as the number of sites increases. Again, for a better comprehension of the speed-up behavior of the centralized algorithm compared to the distributed one, we show in figure 4 a graphical representation of the practical results obtained. It is worth saying that the set of results for the input data groups considered were the same as before, so to be able to make the comparisons.

In general, for input graphs with low connectivity the distributed algorithm performs better than the centralized one even for a small number of sites. However, for an input data in group C with low edge probabilities, as the one shown in figure 4, we see a better speedup for the centralized algorithm running on up to 10 parallel sites. Beyond that, the effect of a bottleneck at the communications coordinating central node explains why the distributed algorithm perform better. As an exception, we see that both algorithms have equivalent performances for the chosen set of inputs in group B. This clearly shows the impact of the useless data communications and facts productions, a main drawback for the distributed algorithm.

Another interesting point to be noted is that both number of iterations and site activations are significantly smaller in the centralized algorithm compared to the distributed one but these do not have positive impact on the speedup. We explain this fact with the following

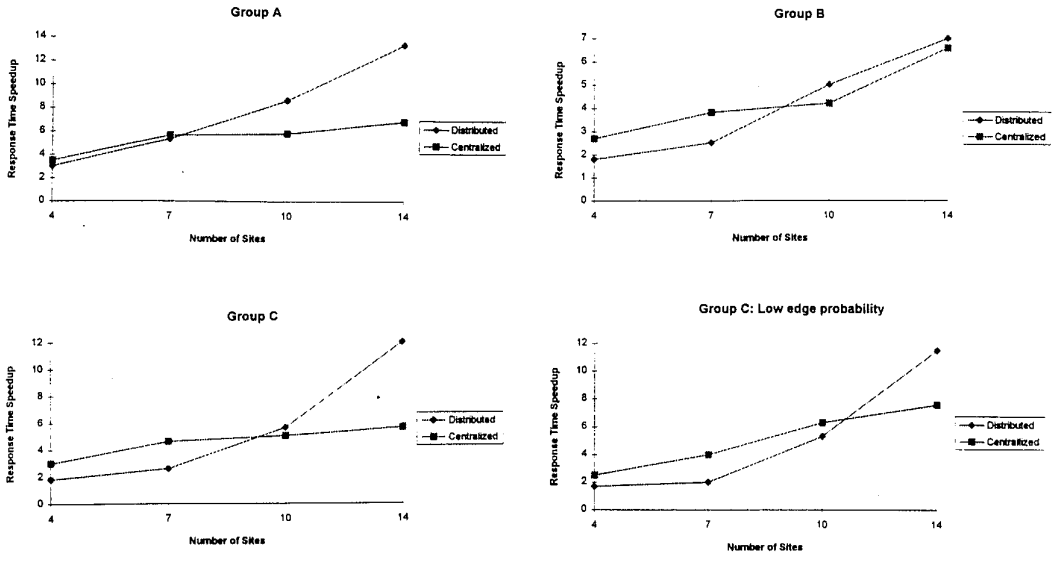

Figure 4: Centralized x Distributed speedups

observation: the distributed algorithm makes a large number of joins involving relations of different sizes, usually many times smaller than in the centralized case. This means that the resulting relations are made available faster to other nodes, and consequently, the chances a node may become idle is reduced.

We also give in table 3 the rates of number of data sets transmissions and their volume comparing the results obtained with the distributed algorithm and the centralized one. As expected, there are many more messages when considering the distributed algorithm than the centralized one, running over the same input data and number of parallel sites. For input group C, when both input and number of sites increase, there are 15 to almost 90 times less messages when the data transfers are centralized. There are also more tuples transferred from one site to another during the whole evaluation of the distributed communications algorithm. However, the rates obtained are smaller (1.5 in the average), showing that the filtering of tuples at the coordinating site do not reduce significantly the amount of tuples exchanged during the parallel evaluation process. Thus, as there are less communications, the data sets transmitted in the centralized algorithm are much bigger than in the distributed one.

5.2 Further results

We will describe next other experiments we have done over different inputs and also some slight variations on the algorithms and their performance implications. The main goal here was not to execute an extensive set of experiments as we have done before. Rather, we have *forced* a few particular situations so to further observe the behavior of the algorithms with respect to other parameters not studied yet.

Local fixpoint transmissions strategy

In all previous experiments we have assumed that all processing sites, whether in the distributed or centralized algorithm, execute their transmissions at each iteration, making their local derived facts available to other sites as soon as possible. As suggested in [WO90], one could consider to do these transmissions only at each local fixpoint reached. Surprisingly, for some of our executions this decision have effectively changed the speedup. Indeed, this is the case for an input corresponding to the parisien subway graph, with its results shown in figure 5. It is a graph within group A in our table of inputs but producing more than 85,000 tuples in the output. There are exactly 292 nodes but it is a highly connected graph.

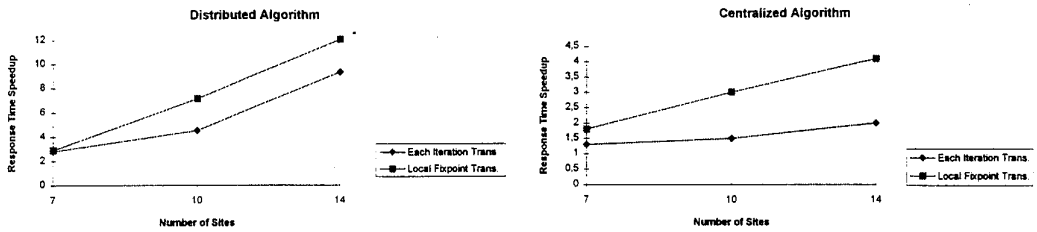

Figure 5: Parisien Subway Graph

As we can observe, the speedups are improved in both centralized and distributed algorithms, with the latter having a better performance in all cases. For 10 sites, the speedup with local fixpoint transmissions almost double in both algorithms. However, this good result is not always achieved. For DAGs with 10,000 tuples and low connectivity, transmissions at each fixpoint may reduce the speedup. Indeed, for 14 sites, rather than 11.9 we have obtained a 8.9 speedup. Anyway, we claim that this is an important aspect that must be further considered when implementing the algorithms.

Non-uniform data distribution, binary trees and lists

All previous results were given for input data randomly generated on an uniform data distribution. In [CLP94] it is mentioned that the centralization of the inter-sites communications could balance the workload - and consequently improve the expected performances - for data with a non-uniform distribution. In order to verify this, we have done some experiments on randomly generated graphs for edge existence probabilities with a large variation, specifically ranging from 0.1% to 10%. Once more, we have tried many different situations, for small and large number of nodes, 4 up to 15 parallel sites, random graphs and DAGs. As we shall see, there is a clear improvement in the centralized algorithm and this even for small input relations.

Taking one of this non-uniform inputs, a DAG with 2,500 tuples in group A and executing both algorithms with 10 parallel sites, we have obtained a speedup 8.2 for the centralized algorithm and only 4.6 for the distributed one, as shown in figure 6. Rather surprising, for a number of sites below 10 the centralized algorithm is, in the average, less than 10% faster and for a number of sites over 10, both algorithms perform equivalently, which is a very rare

situation, as already mentioned in this work. However, it should be noted that the speedup obtained for 14 sites is only around 9.2, which clearly shows that the central site bottleneck is still present when the number of sites increases.

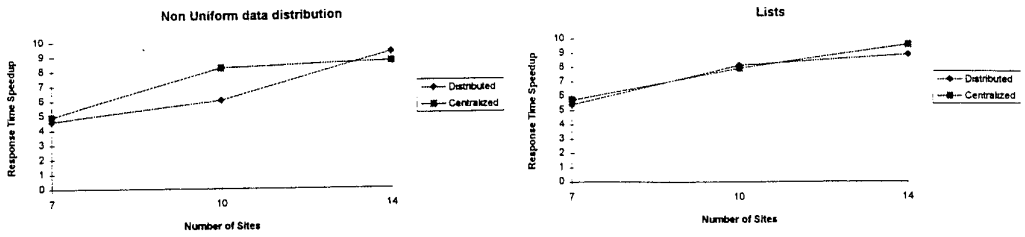

Figure 6: Non-uniform data distribution and lists

Besides completely random graphs and DAGs, we have also executed the parallel algorithms for inputs with a special structure such as complete binary trees and single lists. For lists varying from 200 to 1,000 nodes, the centralized algorithm performs slightly better, or as good as, the distributed one for any number of sites. The observed difference in the speedup is no more than 2 and for a few situations, the speedups are about the same. An important observation is that when the input is a list we get the best speedups for the centralized algorithm, *e.g* 7.8 for 10 sites and 9.6 for 14 sites, both for lists with 500 nodes, as shown in figure 6. Concerning complete binary trees, we have similar observations as the ones made for lists and for the lack of space the specific results will be omitted. However, it should be noted that the speedups obtained are better than when the algorithms are executed with random graphs or DAGs, close to linear speedup specially for the distributed algorithm.

Limiting size of transmission sets

The main problem of the centralized algorithm is clearly the bottleneck at the central node. From the previous experimental results we have observed that when the transmission set size is relativelly small, not only the coordinating node works more efficiently but also there is a better balance on distribution of the work among both parallel sites and the communications network. Thus, we have decided to simulate what would happen if there is a fixed size for the transmission set that can be sent from the central node to the processing sites.

We have done a few experiments limiting the size of the transmissions sets sent from the coordinating node and a reduction on the response time of the centralized algorithm was achieved in some situations. For example, for inputs on group C, such as DAG with 10,000 tuples and low edge probability, the speedup obtained before ranged from 4.4 to 5.7 when increasing the number of sites. When we fixed the transmission set size to 500 tuples (an arbitrary fixed number among several tested), this range changed to 4.7 to 6.5. The corresponding speedup for the distributed algorithm ranged in this case between 2.6 and 11.9, thus, still better than the centralized results for a large number of sites. However, when we fixed the transmission set size on 1,000 tuples, no gain was obtained and for more than 8 sites the speedup obtained 5.5, slightly worse than before. These examples and many others show us that this issue deserves more experiments, so to become an effective way to further tune the centralized algorithm. It

seems that a small size for the transmission set might give better results for the centralized algorithm, but determining the exact size (or range) it should be is out of the scope of this paper.

6 Related Work

One of the first performance analysis of recursive queries evaluation was done in [BR88]. For a given set of rules and queries, such as complete and partial transitive closure and same-generation, the performance analysis of some evaluation methods in the single processor context was simulated for specific input data. However, some restricting assumptions are made: no cyclic relations are considered and also there are no distinct paths between any two nodes with different lengths. Thus, redundant derivation of facts are avoided. The cost metrics taken into account is the number of successful productive rule instantiations. Joins have their cost associated to the size of the resulting relation. In [AJ88], some practical analysis of multiprocessor transitive closure algorithms are done, in both shared-memory and shared-nothing architectures. Only relatively small input relations are tested.

In [HWF93], the authors implement a particular algorithm for evaluating the transitive closure in a parallel main-memory DBMS called PRISMA//DB, namely the disconnection set approach [HAC90]. It is a very specific performance study *wrt* to the algorithm and hardware considered. In [CCH93], a performance analysis is simulated focusing on both communication cost, which is measured as the total number of tuples transmitted between sites and join cost, taken as proportional to the size of the resulting relation. The amount of data transmitted is mentioned as one of the main factors that may influence the performance. No practical implementations were done. In [DR94], a very complete study of the performance evaluation of sequential transitive closure algorithms is done. Important query and system parameters, like the number of nodes and average out-degree of input data, are considered. They also measure the I/O cost. Only DAGs are taken into account.

A first implementation study of algorithms based on the referred paradigm is done in [WZB+93]. However, this work is mainly concerned with a particular query (the partially instantiated transitive closure query) and the analysis is based on non realistic assumptions such as an even workload partitioning among the parallel sites. Very few implementation results are given and only speedup is analysed, with no details on the overall behavior. A dynamic workload balancing strategy is proposed, where one site assumes part of the work of another one that stays idle for a long time. It is not clear, though, how to effectively choose a working site, split its hash function and consequently its work. Moreover, the new scheme may not be better than the previous one and trying many changes like this during the evaluation may affect the performance. In [ZZO94], performance analysis is done as in [VK88], where many simplified assumptions are considered, such as an equally distributed workload among sites and production of the same number of tuples at each phase. These are clearly not applicable in the general situation, as we have shown in this work. A new parallel strategy applied to transitive closure computation is proposed with the focus on the physical data fragmentation design and its relationship with the reduction of the communication cost. Their strategy aims at avoiding the possible redundant work caused by multiple paths.

7 Conclusions and future work

In this paper we have investigated the practical behavior of the bottom-up parallel evaluation of Datalog queries. To our knowledge, we have done a first complete implementation of the bottom-up rule instantiations partitioning paradigm. A parallel environment such as the shared-nothing IBM SP/1 computer running PVM was used for the experiments. This has enabled us to have a closer taste of actual performance of the algorithms such as their speedup. An important result was discussed, where it is shown that the analytical models that have been considered in previous works may not predict the correct behavior of the parallel algorithms in practice. We have run the algorithms on randomly generated data with both uniform and non uniform edge probabilities and we have also observed what happened on a real test data such as the Parisien subway corresponding graph.

We are also concerned with a careful study of the behavior of these parallel algorithms with a close look at the influence of a set of parameters, mainly the ones related to the communication cost. Indeed, we have shown that the communication model chosen may affect considerably the performance. A centralized communication approach is discussed and we have shown that it is worth to be be further examined as an alternative to the completely distributed model usually adopted. Other *low level* factors such as the exact moment a given site should transmit its data sets are investigated and should be considered when implementing the above Datalog parallel evaluation paradigm.

The implementation results obtained by centralizing the inter-site communications during the evaluation process are very promising and we plan to continue analyzing its possible optimizations. Among the ideas to be explored, the more than one central sites alternative [FR90] and a careful communication policy that captures the *optimal* transmission moment and data set content should be further investigated. Concurrently, we have already started is to evaluate the I/O cost involved, which is still the main bottleneck to very large database systems performances. This will be possible with the recent availability of parallel DBMS (e.g. DB2 and ORACLE) running on multiprocessors architectures such as the IBM SP/1. An immediate consequence of our work is that there is a need of good cost estimation formulas in order to be able to compare different algorithms or even to work out on some of the alternatives due to the set of parameters involved.

A side result obtained from this work is that the performance of all algorithms is greatly influenced by redundancy on the production of derived facts. This issue deserves better analysis but we are already working out on parallel strategies that take into consideration the system's *knowledge* about which fact has been already produced so to avoid useless rule instantiations and duplicate facts derivations. Finally, we have started studying how the pure parallelization approach proposed in [LV95] can be effectively used in practical applications. In this case, it seems that an *ad-hoc* data partitioning scheme must be considered but one could also think about a more general, maybe sequential, pre-processing step followed by the parallel evaluation itself.

References

[AJ88] R. Agrawal and H.V. Jagadish, "Multiprocessor Transitive Closure Algorithms", *Proc.*

Intl. Symp. on Databases in Parallel and Distributed Systems, 1988, pp 56–66.

[BR88] F. Bancilhon and R. Ramakrishnan, "Performance Evaluation of Data Intensive Logic Programs", *Foundations of Deductive Databases and Logic Programming, Ed J. Minker*, Morgan Kauffman, 1988, pp 439–517.

[BSH91] D.A. Bell, J. Shao and M.E.C. Hull, "A Pipelined Strategy for Processing Recursive Queries in Parallel", *Data & Knowledge Engineering*, 6(5), 1991, pp 367–391.

[CCH93] F. Cacace, S. Ceri and M.A.W. Houtsma, "A Survey of Parallel Execution Strategies for Transitive Closure and Logic Programs", *Distributed and Parallel Databases*, 1(4), 1993, pp 337–382.

[CLP94] J.-P. Cheiney, S. Lifschitz and P. Picouet, "A Centralized Communication Approach for the Parallel Shared-Nothing Evaluation of Datalog" *Proc. Brazilian Symposium on Database Systems*, 1994, pp 51–64.

[CW89] S.R. Cohen and O. Wolfson, "Why a Single Parallelization Strategy is not Enough in Knowledge Bases", *Proc. ACM Symp. on Principles of Database Systems*, 1989, pp 200–216.

[DR94] S. Dar and R. Ramakrishnan, "A Performance Study of Transitive Closure Algorithms", *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, 1994, pp 454–465.

[DG92] D. Dewitt and J. Gray, "Parallel Database Systems: the Future of High Performance Database Systems", *Communications of the ACM*, 35(6), 1992, pp 85–98.

[FR90] D.G. Feitelson and L. Rudolph, "Distributed Hierarchical Control for Parallel Processing", *Computer*, 1990, pp 65–77.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam "PVM3 User's Guide and Reference Manual", *Oak Ridge National Laboratory TM-12187*, 1994.

[GST90] S. Ganguly, A. Silberschatz and S. Tsur "A Framework for the Parallel Processing of Datalog Queries", *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, 1990, pp 143–152.

[HAC90] M.A.W Houtsma, P.M.G. Apers and S. Ceri, "Distributed Transitive Closure Computations: the Disconnection Set Approach", *Proc. Intl. Conf. Very Large Databases*, 1990, pp 335–346.

[HWF93] M.A.W Houtsma, A.N. Wilschut and J. Flokstra, "Implementation and Performance Evaluation of a Parallel Transitive Closure Algorithms on PRISMA/DB", *Proc. Intl. Conf. Very Large Databases*, 1993, pp 206–217.

[Hul89] G. Hulin, "Parallel Processing of Recursive Queries in Distributed Architectures", *Proc. Intl. Conf. on Very Large Data Bases*, 1989, pp 87–96.

[Kan88] P.C. Kanellakis, "Logic Programming and Parallel Complexity", *Foundations of Deductive Databases and Logic Programming, Ed J. Minker*, Morgan Kauffman, 1988, pp 547–585.

[LV95] S. Lifschitz and V. Vianu, "A Probabilistic View of Datalog Parallelization", *Proc. Intl. Conf. on Database Theory*, 1995, pp 294–307. (extended version to appear in Theoretical Computer Science)

[GRS95] N. Goodman, S. Rozen and L. Stein, "Requirements for a Deductive Query Language in a Genome-Mapping Database", *Applications of Logic Databases, Ed. R. Ramakrishnan*, Kluwer Academic Publishers, 1995, pp 259–276.

[SL91] J. Seib and G. Lausen, "Parallelizing Datalog Programs by Generalized Pivoting", *Proc. ACM Symp. on Principles of Database Systems*, 1991, pp 78–87.

[SBH91] J. Shao, D.A. Bell and M.E.C. Hull, "Combining Rule Decomposition and Data Partitioning in Parallel Datalog Program Processing", *Proc. Intl. Conf. on Parallel and Distributed Information Systems*, 1991, pp 106–115.

[Sto86] M. Stonebraker, "The Case for Shared Nothing", *Database Engineering*, 9(1), 1986, pp 4–9.

[Tsu91] S. Tsur, "Deductive Databases in Action", *Proc. ACM Symp. on Principles of Database Systems*, 1991, pp 142–153.

[Ull89] J.D. Ullman, "Bottom-up beats Top-down for Datalog", *Proc. ACM Symp. on Principles of Database Systems*, 1989, pp 140–149.

[UvG88] J.D. Ullman and A. Van Gelder, "Parallel Complexity of Logic Query Programs", *Algorithmica*, 3, 1988, pp 5–42.

[Val93] P. Valduriez, "Parallel Database Systems: Open Problems and New Issues", *Distributed and Parallel Databases*, 1(2), 1993, pp 137–165.

[VK88] P. Valduriez and S. Khoshafian, "Parallel Evaluation of the Transitive Closure of a Database Relation", *Intl. Journal of Parallel Programming*, 17(1), 1988, pp 19–42.

[vGel86] A. Van Gelder, "A Message Passing Framework for Logical Query Evaluation", *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, 1986, pp 155–165.

[Wol88] O. Wolfson, "Sharing the Load of Logic-Programming Evaluation ", *Proc. Intl. Symp. on Databases in Parallel and Distributed Systems*, 1988, pp 46–55.

[WO90] O. Wolfson and A. Ozeri, "A New Paradigm for Parallel and Distributed Rule-Processing", *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, 1990, pp 133–142.

[WS88] O. Wolfson and A. Silberschatz, "Distributed Processing of Logic Programming", *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, 1988, pp 329–336.

[WZB+93] O. Wolfson, W. Zhang, H. Butani, A. Kawaguchi and K. Mok "A Methodology for Evaluating Parallel Graph Algorithms and its Applications to Single Source Reachability", *Proc. Intl. Conf. on Parallel and Distributed Information Systems*, 1993, pp 243–250.

[ZWC91] W. Zhang, K. Wang and S-C. Chau, "Data Partition: a Practical Parallel Evaluation of Datalog Programs", *Proc. Intl. Conf. on Parallel and Distributed Information Systems*, 1991, pp 98–105.

[ZZO94] X. Zhou, Y. Zhang and M.E. Orlowska, "A New Fragmentation Scheme for Recursive Query Processing", *Data & Knowledge Engineering*, 13, 1994, pp 177–192.