Potentiality of Parallelism in Logic

Franz Kurfeß*

TR-91-055

Abstract

The processing of knowledge is becoming a major area of applications for computer systems. In contrast to data processing, the current stronghold of computer use, where well-structured data are manipulated through well-defined algorithms, the treatment of knowledge requires more intricate representation schemes as well as refined methods to manipulate the represented information. Among the many candidates proposed for representing and processing knowledge, logic has a number of important advantages, although it also suffers from some drawbacks. One of the advantages is the availability of a strong formal background with a large assortment of techniques for dealing with the representation and processing of knowledge. A considerable disadvantage so far is the amount and complexity of computation required to perform even simple tasks in the area of logic. One promising approach to overcome this problem is the use of parallel processing techniques, enabling an ensemble of processing elements to cooperate in the solution of a problem. The goal of this paper is to investigate the combination of parallelism and logic.

*Preprint from: Wrightson, G and Fronhöfer, B.; *Parallelization in Inference Systems.* Springer Verlag

1 Parallelism in Inference Systems

The best-known instance of an inference mechanism in computer science is the use of PROLOG as logic programming language. As a consequence, most approaches to combine parallelism and inference systems are based on PROLOG or a derivative thereof as language, and evaluation mechanisms based on the resolution calculus. Among these approaches, OR-parallelism and AND-parallelism are almost exclusively used, either isolated or a combination of both. In the following, we will show that there is quite a variety of other categories of parallelism, sometimes with distinct advantages over AND-/OR-parallelism. Our investigation will concentrate on Horn clause logic and first order predicate logic, and not include extensions or specializations like temporal logic, modal logic, higer order logic, fuzzy logic, probabilistic reasoning, etc.

For most of our investigations, we will also assume that a general-purpose parallel architecture will be used as execution vehicle. Such an architecure may provide inadequate support for certain basic operations in the evaluation of logic programs (selection of clause heads, unification), but experience with dedicated machines leads to the conclusion that the gain in performance in general-purpose architectures is overwhelming compared to special-purpose ones. We will not restrict ourselves, however, to a certain processing mode, architecture or topology; in particular, we want to overcome the severe implications imposed by the inherently sequential execution mode of a stack-based Warren Abstract Machine WAM [Warren, 1983] and resolution as underlying calculus. This will give us the freedom to investigate different computational models, especially with respect to their suitability for parallel evaluation.

2 Categories of Parallelism

This section gives an overview of various categories of parallelism which can be identified in inference systems. From a conceptual point of view, parallelism can be introduced by two ways: first by identifying independent parts in the program to be evaluated, and executing these parts separately. It is also possible, however, to view the program to be evaluated as data, which are transformed by certain operations according to a particular inference mechanism, and apply some of these operations in parallel to the whole, or parts of the original program.

Table 1 shows an overview of the categories of parallelism, arranged according to the granularity and the components of a logic program. It identifies the particular data structures and operations applied in a category.

The notation used in Table 1 is based on viewing a logic program as a collection of clauses, possibly organized into modules (or objects). The clauses consist of literals, arranged as head and tail. A literal is identified through a predicate, and may have arbitrarily complex terms composed from functions, constants, and variables. Further notational details depending on the particular category of parallelism will be given at the appropriate place.

Symbol	Meaning
P	logical program or formula
$\{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$	distinct programs
$\mathcal{P} = \{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$	program composed of modules
$\mathcal{P} = \{\mathcal{SPS}_1, \ldots, \mathcal{SPS}_s\}$	spanning sets in a program
$\widetilde{\mathrm{P}}$	modified program
C	clause
L	literal
$\{\mathcal{R}_1, \ldots, \mathcal{R}_r\}$	routes in a program
$t = \{st_1, \ldots, st_s\}$,	term, composed of subterms
(t, t')	pair of terms to be unified
$\{f^1, f^2, f^3, \ldots\}$	occurrences of one function
$f(t_1, \ldots, t_m)$,	function with arbitrary terms as arguments
$(g_1, \ldots, g_m)$,	ground terms
$\{\textsc{Inf}_1, \ldots, \textsc{Inf}_n\}$	different inference mechanisms
$\widetilde{\textsc{Inf}}$	modified inference mechanism
$\{\textsc{Red}_1, \ldots, \textsc{Red}_r\}$	reduction transformations
$\{\textsc{Unif}_1, \ldots, \textsc{Unif}_r\}$	unification mechanisms

Multitasking Independent programs can be evaluated at the same time. This concept is well-known from traditional operating systems and does not involve major problems, at least from our point of view. Difficulties which may arise are only on the operating system level, e.g. through the utilization of shared resources, load balancing, built-in predicates, and should be solved on that level without intervention from the inference mechanism or the user. Multitasking may be of interest for the evaluation of logic programs from different users on a shared, large multiprocessor machine.3

Competition The evaluation of a logic program can be done in a number of different ways, usually described in the form of a calculus. The idea of competition is to apply different evaluation mechanisms to one and the same program. Whereas in principle this results in redundant computation, different calculi are well suited for different classes of programs [Ertel, 1990, Ertel, 1991] (see also the contribution of W. Ertel in this volume). Often it is not even necessary to use different calculi: changing the strategy to traverse the search space can make a tremendous difference and is often the source of claims to superlinear speedup in the parallel evaluation of logic programs. To illustrate this point, consider the way PROLOG builds the proof tree for a program, with its depth-first, left-to-right strategy. In the worst case, it gets stuck in an infinite branch in the left part of the proof tree, whereas a solution would be found easily on the other side of the tree. Changing the search strategy, either from depth-first to breadth-first or from left-to-right to right-to-left, would lead

Level	Data Structures	Operations	Category
Formula	$\{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$	INF	*Multitasking*
	$\mathcal{P}$	$\{\text{INF}_1, \ldots, \text{INF}_n\}$	*Competition*
	$\mathcal{P}$	$\{\text{INF}_1, \ldots, \text{INF}_n\}$	*Cooperation*
	$\mathcal{P}$	$\{\text{INF}_{Pr_1}, \ldots, \text{INF}_{Pr_n}\}$	*Precision*
	$\mathcal{P} = \{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$	INF	*Modularity*
	$\mathcal{P}$	$\{\text{RED}_1, \ldots, \text{RED}_r\}$	*Reductions*
	$\widetilde{\text{P}}$	$\widetilde{\text{INF}}$	*Recursion*
	$\mathcal{P} = \{\mathcal{SPS}_1, \ldots, \mathcal{SPS}_s\}$	$\text{INF}_{ConnectionMethod}$	*Spanning Sets*
Clause	$\{\mathcal{L}, \mathcal{C}_1, \ldots, \mathcal{C}_s\}$	$\text{INF}_{Resolution}$	*OR-Parallelism*
Literal	$\{\mathcal{R}_1, \ldots, \mathcal{R}_r\}$	UNIF	*Routes*
	$\{\mathcal{L}_1, \ldots, \mathcal{L}_n; \mathcal{C}_1, \ldots, \mathcal{C}_n\}$	Resolution Steps	*AND-Parallelism*
	$\{\mathcal{L}_1, \ldots, \mathcal{L}_n; \mathcal{C}_1, \ldots, \mathcal{C}_n\}$	Pipelined Resolution Steps	*Pipelining-Parallelism*
Term	$\{(t_1, t'_1), \ldots, (t_n, t'_n)\}$	UNIF	*Term Parallelism*
	(t_1, t'_1)	$\{\text{UNIF}_1, \ldots, \text{UNIF}_r\}$	*Unification Competition*
	(t_1, t'_1)	$\{\text{UNIF}_1, \ldots, \text{UNIF}_r\}$	*Unification Cooperation*
	(t_1, t'_1)	Cycles	*Occur Check*
	$t = \{st_1, \ldots, st_s\}$, $t' = \{st'_1, \ldots, st'_s\}$	Unification	*Separability*
	$\{f^1, f^2, f^3, \ldots\}$	Function Evaluation	*Function Call*
	$f(t_1, \ldots, t_m)$, $f(t'_1, \ldots, t'_m)$	Decomposition of Unification	*Forwarding*
	$(g_1, \ldots, g_m)$, $(g'_1, \ldots, g'_m)$	Composition of Unification	*Bottom Up*
Atom	arrays	SIMD Operation	*Data Parallelism*
	lists	Incremental Evaluation	*Streams*
	e.g. feature maps	e.g. Search for Global Minimum	*Subsymbolic Representation*

Table 1: Categories of parallelism in logic

to a rapid detection of the solution. On the other hand, PROLOG programs written by experienced PROLOG programmers usually rely on knowledge about the underlying execution mechanism, and changing it may have disastrous results. The use of competition is better suited for areas like automated theorem proving where the formulation of a program is derived from the specification of the problem to be solved and not so much through the description of the solution process with logical means.

Cooperation The cooperation category is based on the exchange of useful information between different inference mechanisms working on the same program, and thus can be viewed as a counterpart to the previously described competition approach [Fronhöfer and Kurfeß, 1987]. Useful information can consist of intermediate results, e.g. in the form of lemmata, or a partitioning of the search space into sections treated by the different mechanisms, or meta-level information, e.g. the (estimated) probability that a proof can be found in a certain part of the search space. There is certainly a large potential of useful information to be exchanged, but also a tradeoff between the amount of data to be transferred as well as the overhead to determine which information is worth while transferring, and the gain that arises from the avaliability of these data.

Precision Whereas in the competition approach different inference mechanisms are employed, precision relies on the use of basically the same inference mechanism, but with various degrees of precision through changes in the unification procedure. The range of precision is from the propositional "skeleton" of the formula (no unification) over weak unification (substitutions are only computed locally) and unification without occur check (cyclic substitutions can occur) to predicate logic with full unification. Inference mechanisms with different precision operate fully in parallel or in a pipelined fashion, where the ones with low precision proceed faster, thus eliminating dead branches earlier and reducing the remaining search space for the more precise mechanisms.

Modularity A widely accepted methodology for the development of large software systems is to structure the overall system into largely independent subsystems with clearly defined interfaces. This approach obviously is also relevant for inference systems and logic programs, and emerges in three variants. First, it is possible not to impose any additional syntactical structuring on large programs, but to rely on the inherent structuring as result of the particular problem together with the style of the particular programmer. The resulting program is analysed, and independent parts are identified. The second approach is to explicitly characterize modules in the source code, which eliminates the potentially costly analysis of the program. A further step is to integrate object-oriented concepts into the logic programming paradigm.

The common theme for these approaches is to group the program into parts which have few interactions among each other, and thus can be evaluated more or less

independently. Restrictions in the parallel evaluation, however, can still be imposed through time or data dependencies.

Reductions An analysis of the program to be evaluated is also a central point for reductions. The goal here, however, is to eliminate redundant or unnecessary code, thus restricting the search space to be traversed. The application of reductions must maintain important properties of the program, and they are usually grouped into equivalence-preserving and satisfiability-preserving transformations. Some reductions operate locally on parts of the program (single clauses, pairs or sets of clauses), whereas others require information about the program as a whole. While reductions may change the text of the program, they can be applied in parallel since the transformations do not change the important properties of the program.

Recursion The use of recursion on one hand can be used to write simpler programs, but on the other hand is difficult for parallelization because the unrolling of recursive clauses typically is done dynamically at execution time. This is due to the fact that the number of iterations can only be determined from the actual goal together with the program. Once that goal is available, however, simple calculations often can be made to determine or estimate the number of iterations. Then in some cases it is possible to apply conventional parallelization techniques to transform the sequence of operations in the loop into operations executed in parallel [Millroth, 1991]. The potential gain from this technique is especially large if it can be used to predetermine the structure of the AND/OR-tree, and thus reduce the complexity of managing of the binding environments.

Spanning Sets One important advantage of the spanning set concept as well is to make the run-time management of bindings simpler. This is based on a statical analysis of the program with the goal of identifying parts of the program (spanning sets of connections) which represent alternative solutions [Bibel, 1987, Wang, 1989, Ibañez, 1988, Ibañez, 1989, Kurfeß, 1990] (see also the contribution of Wang, Marsh and Lavington in this volume). These solutions then can be computed completely independent of each other, without the necessity of maintaining complex run-time environments for variable bindings. The limitation of this concept lies in the treatment of recursive parts, because these cannot in general be expanded statically. A combination of spanning sets with the techniques for unrolling recursion looks quite promising at least for certain simple kinds of recursion.

OR-Parallelism Spanning sets and OR-parallelism exploit the same feature, namely the computation of alternative solutions, but with different techniques. Whereas spanning sets use statical analysis, OR-parallelism usually invokes new threads of computation dynamically whenever there are multiple clauses for resolving a subgoal. This technique is a straightforward extension of the resolution mechanism and

can be incorporated into existing PROLOG implementations without major problems. The disadvantage, however, is the management of the variable bindings for different threads of computation, which must be open to backtracking in the case that a candidate for a solution turns out to be invalid.

Routes The routes concept again relies on a statical analysis of the program, identifying sequences of connections (or resolution steps) which will have to be followed through in the evaluation, and which contain OR-parallel connections only at the end points [Ibañez, 1988, Ibañez, 1989, Kurfeß, 1990]. The crucial point with respect to parallelism is that these routes do not necessarily have to be treated in a sequential way; they may as well be combined pairwise with logarithmic execution time instead of linear. The problem here is the same as with the spanning sets: the appearance of recursive parts requires dynamical treatment, but again a combination with the unrolling of recursion can improve the situation.

AND-Parallelism This category is widely used in parallel logic programming, and relies on a concurrent evaluation of the literals in the clause body (subgoals). Problems occur if variables are shared between literals, since they must assume identical values [DeGroot, 1984, DeGroot, 1988]. On the other hand, shared variables are a convenient way to express synchronization between different threads of computation, which makes languages based on AND-parallelism also well suited for low-level tasks such as systems programming. Especially when OR- and AND-parallelism are combined in one execution model, the management of variable bindings can get very complicated, so that sometimes the parallel evaluation is restricted to independent subgoals, i.e. subgoals which do not share variables.

Pipelining-Parallelism Instead of evaluating the subgoals in a clause fully in parallel, they can be processed in a pipelined fashion [Beer and Giloi, 1987, Beer, 1989]. The advantage is the full compatibility with the standard PROLOG evaluation mechanism, which proceeds from left to right in the body of a clause. The performance gain achieved through this scheme (which relies on dedicated hardware), however, is very limited, and is easily outperformed by the performance progress in general-purpose architectures.

Term Parallelism The task of evaluating a logic program can conceptually be devided into two parts: one is to navigate through the search space as indicated by the relations between the predicates in the program, the other to guarantee that the variable bindings made during the traversal are consistent. The second is based on unification as underlying operation for determining the variable bindings in order to make two terms equal. In analogy to the simultaneous application of inference operations to distinct logic programs, unification can be performed simultaneously on different term pairs (or, more generally, sets of terms). Parallelism on the term level

underlies two important restrictions: first, the task of unification can be inherently sequential in the worst case [Yasuura, 1984, Dwork et al., 1988]; second, the grain size for unification tasks tends to be rather small with a PROLOG-based evaluation model. The first restriction is of a fundamental nature, but depends to a large degree on the formulation and representation of the problem to be solved. The second restriction can be overcome by choosing a different evaluation mechanism [Bibel, 1987, Wang, 1989, Ibañez, 1988, Ibañez, 1989, Kurfeß, 1990].

Unification Competition Conceptually similar to competition in the inference mechanism, competition can be used for unification by applying different unification mechanisms to one and the same term pair. It is questionable, however, if this kind of parallelism is very useful since the variety of unification mechanisms is not too large, and performance differences largely can be attributed to the term size, and not so much to the internal structure of the terms [Corbin and Bidoit, 1983].

Unification Cooperation The use of cooperation during unification has a close affinity to unification competition described above: it seems feasible, but with current techniques probably does not result in substantial advantages.

Occur Check The task of the occur check is to identify situations where a term is substituted by a subterm of itself, resulting in cyclic substitutions [Plaisted, 1984]. Whereas it is possible to define a semantics which allows the occurrence of infinite substitutions [Colmerauer, 1982], or in many situations to just neglect potential problems and omit an occur check (as in many PROLOG implementations), there are cases where cyclic substitutions should not be accepted. The occur check is basically a search of a graph for cycles, an operation which can be implemented independent of unification per se. It is not possible to perform these two operations completely in parallel since the structure of the terms involved is changed during unification, and cyclic substitutions may be introduced through unification. At best unification and occur check can be performed in a pipelined fashion, where the occur check is performed incrementally on the section of the terms already unified [Hager and Moser, 1989].

Separability This category corresponds to modularity on the program level by identifying sections in the terms to be unified which are more or less independent. It is probably only useful to exploit this kind of parallelism if there is already a partitioning induced by the structure of the program or its representation, since the cost of performing such an analysis is not much smaller than actually performing unification.

Function Call A considerable amount of work during the unification process can consist of evaluating functions; this work may be done independent of the rest of unification, possibly even at compile time. The basic idea is the to try to evaluate

all or some of the function calls in parallel to the rest of unification, replacing the calls by the computed results. To some degree this parallelism is speculative since work may be done which actually is not needed; the advantage, however, is that it can be used to occupy idling processing elements, for example in the initial phase of unification when only a few processing elements are working close to the roots of the terms.

Forwarding The process of unifying tow terms can be viewed as assignment or comparison for atomic components (variable or constant symbols), and propagating the task of unification to the corresponding subterms. Forwarding denotes the propagation of unification to subterms in parallel. A problem arises for multiple occurrences of one and the same symbol (or whole subterm), which must have the same value. It can be solved by a suitable representation, for example as dags (directed acyclic graph), where multiple occurrences of substructures are represented only once, but with severla incoming pointers.

Bottom Up In this case, activation starts from the constants and instantiated variables in order to reduce the complexity of the terms to be unified by replacing function calls with their results and instantiating variables where possible. This is again a case of speculative parallelism since the correlation between substructures from the left and right term can only be determined in a top-down way.

Data Parallelism Data structures in logic programs are usually characterized through terms, which may have a highly irregular internal structure. In many practical cases, however, regular data structures are used, the prototype being an array of elements of some type. Such regular data structures open the door to another category of parallelism, usually referred to as data parallelism [Hillis and Steele, 1986]. Here, one and the same operation is applied to all or a subset of the single elements in a SIMD (Single Instruction Multiple Data) way. Whereas typical logic programs may not contain many data structures where data parallelism can be applied, there certainly are many applications with highly regular data structures which can profit from this combination, such as deductive data bases, image processing, or scientific computations [Fagin, 1991, Succi and Marino, 1991].

Streams The concept of streams is based on an incremental evaluation of large data structures, e.g. lists which change dynamically [Ito et al., 1987, Takeuchi et al., 1987]. Usually operations involving the whole data structure, like comparing two lists, are only applicable if and when the whole data structures are available. In some cases, operations can be applied while these data structures still evolve, like appending elements to the end of the list. This category may be on the borderline of parallelism, but eases some synchronization constraints in the parallel evaluation of programs.

Subsymbolic Representations All the categories described so far implicitly rely on a symbolic representation, where an atomic part of the program has a direct correspondence to the internal representation in the execution vehicle, typically one or more memory cells. The most important operations here are assignment, where the value of a symbol is set to the result of an operation, and pointer manipulation, where references between symbols are established or changed. An important aspect of this representation scheme is that applied to symbols, or at least to substantial parts of symbols. It is conceivable – and considerable interest has been devoted to this recently – to construct inference mechanisms based on subsymbolic representations, where there is no direct correlation between symbols and machine-internal representations [Touretzky and Hinton, 1985, Touretzky and Hinton, 1988, Ballard, 1986, Smolensky, 1987, Lange and Dyer, 1989, Shastri, 1988, Shastri and Ajjanagadde, 1989, Pinkas, 1990]. One approach is to represent symbols and programs as patterns of activation and interconnection distributed a network of simple units, which are connected through weighted links [Hölldobler, 1990a, Hölldobler and Kurfeß, 1991][1]. The computation then is performed by a spreading activation scheme which settles into a stable state when a solution is found. This category opens up a whole new dimension of parallel evaluation due to its different representation and computation paradigm.

3 Exploitation of Parallelism

The identification of sources for parallelism in inference systems is an interexting topic for itself, but in order to be practically applicable must be accompanied by an investigation of which kinds of parallelism are worth while to be exploited. This becomes rather complicated, especially when one tries to combine different categories of parallelism in one evaluation mechanism.

3.1 Analysis of Parallelism

An inportant method for the exploitation of parallelism in logic is the analysis of programs, for example by determining the potential parallel factor [Harland and Jaffar, 1987] or more complex measures. This can be done in a general investigation, analysing many logic programs with the aim of identifying common, useful ways to exploit parallelism for (classes of) logic programs [Onai and et al., 1984, Delcher and Kasif, 1989, Debray, 1989, Debray et al., 1990]. The same can be done for individual programs through a static analysis at compile time, or through symbolic evaluation to capture also some dynamic aspects of the program execution. To go even further, one can perform sample executions of a program with a set of typical goals, thus gaining information for the dynamic behavior of the program in real use.

[1] see also this volume

3.2 Control of Parallelism

The analysis of programs before the actual execution can provide important information about the expected dynamic behavior, but will in many cases not be sufficient to completely determine the actual execution pattern. The control of parallelism at run-time can be achieved on three different levels: through specifications provided by the user in the program itself, through strategies determined by the compiler, and through measures of the operating system. A problem here is to find a balance between two conflicting goals: high performance, and easy program development. Highest performance can be achieved by giving the user direct access to operating system features, e.g. for communication, synchronization, load balancing, I/O, etc. The price to be paid is that in this case the user needs an intimate knowledge of the execution mechanism, the operating system, and the underlying hardware architecture. The other extrem is to relieve the user form all these low-level tasks, only requiring a formal problem specification based on logic. This, however, may result in disappointingly poor performance, at least as long as the development in terms of automated analysis and control mechanisms is not developped very far.

A partial solution to this dilemma might lie in the use of *control abstraction*, which follows a strategy similar to data abstraction [Crowl and LeBlanc, 1991]. The basic idea is to provide a safe way to introduce explicit, user-definable control constructs for parallelism. The introduction of these constructs is separated into two parts: one describes the desired properties in a formal definition, the other provides the actual implementation, based both on the formal definition as well as on features of the machine used for execution. Whereas this method still requires some effort from the user's side in order to achieve high performance, it allows the definition of problem-specific control constructs based on efficiently implemented control primitives provided by the operating system. Another advantage is better partability, since the architecture-dependent features are concentrated in one place, namely the particular implementation of the construct.

This method fits nicely into the logic framework e.g. through modules (or objects) and meta-evaluation, but requires some higher-order logic features.

3.3 Restrictions

While the abstract potential of parallelism in logic is quite high, its practical exploitation is subject to a number of restrictions. Some of these are under a limited control from the user, like the problem structure, specification and representation, as well as the actual encoding in the form of a program. Others are implied through the underlying evaluation mechanism, the runtime environment (shared binding environments), operating sytem, and hardware architecture.

4 Conclusions

The goal of this paper has been to illuminate the potentiality of parallelism in logic programming and inference systems. For this purpose, we investigated different categories of parallelism derived from the evaluation of predicate logic program, without having a particular calculus, abstract machine, or underlying hardware in mind. The outcome is a variety of different categories, far beyond the AND-/OR-parallelism usually found in attempts to parallelize PROLOG. Admittedly some of these categories are not very relevant in combination with present technology. Others, however, in particular the ones based on static program analysis in combination with programming methodologies aiming at easier program development while maintaining the possibility to finetuning for high performance, have a very good potential for the exploitation of parallelism in inference systems.

References

[Ajjanagadde, 1990] Ajjanagadde, V. (1990). Reasoning with function symbols in a connectionist system. Technical report, Department of Computer and Information Science, University of Pennsylvania.

[Ajjanagadde and Shastri, 1991] Ajjanagadde, V. and Shastri, L. (1991). Rules and variables in neural nets. *Neural Computation*, 3:121–134.

[Ali, 1987] Ali, K. (1987). OR-parallel execution of Prolog on a multi-sequential machine. *Parallel Programming*, 15(3).

[Amthor, 1989] Amthor, R. (1989). Simulation eines Beweisers auf einer Multi-Prozessor Architektur. Master's thesis, Institut für Informatik, Technische Universität München.

[Aso and Onai, 1983] Aso, M. and Onai, R. (1983). XP's: An Extended OR-Parallel Prolog System. Technical Report TR 023, Institute for New Generation Computer Technology (ICOT).

[Auburn, 1989] Auburn (1989). Parallel logic programming architectures: Final report. Technical report, Department of Computer Science and Engineering, Auburn University, Auburn, AL.

[Bachinger, 1987] Bachinger, J. (1987). Implementierung eines parallelen Theorembeweisers und Simulation der Ausführung auf einer Mehrprozessormaschine. Master's thesis, Institut für Informatik, Technische Universität München.

[Ballard, 1986] Ballard, D. (1986). Parallel Logical Inference and Energy Minimization. Technical Report TR 142, Computer Science Department, University of Rochester.

[Bansal and Potter, 1990] Bansal, A. and Potter, J. (1990). A data-parallel model for efficient execution of logic programs on associative supercomputers. In *North American Conference on Logic Programming*.

[Barnden, 1988] Barnden, J. (1988). Simulations of Conposit, a Supra-Connectionist Architecture for Commonsense Reasoning. In *2nd Symposium on the Frontiers of Massively Parallel Computation, Fairfax, VA.*, Las Cruces.

[Baron et al., 1988] Baron, U., Chassin, J., and Syre, J. (1988). The Parallel ECRC Prolog System PEPSys: An overview and evaluation results. In *FGCS '88*.

[Baron et al., 1987] Baron, U., Ing, B., Ratcliffe, M., and Robert, P. (1987). A Distributed Architecture for the PEPSys Parallel Logic Programming System. Technical report, ECRC Computer Architecture Group, München.

[Beer, 1989] Beer, J. (1989). *Concepts, Design, and Performance Analysis of a Parallel Prolog Machine*, volume 404 of *Lecture Notes in Computer Science*. Springer.

[Beer and Giloi, 1987] Beer, J. and Giloi, W. K. (1987). POPE - A Parallel-Operating Prolog Engine. *Future Generations Computer Systems*, pages 83–92.

[Ben-Ari, 1984] Ben-Ari, M. (1984). *Principles of Concurrent Programming*. Prentice Hall.

[Bibel, 1987] Bibel, W. (1987). *Automated Theorem Proving*. Vieweg, Braunschweig, Wiesbaden, second edition.

[Bibel and Aspetsberger, 1985] Bibel, W. and Aspetsberger, K. (1985). A Bibliography on Parallel Inference Machines. *Symbolic Computation*, 1(1):115–118.

[Bibel and Buchberger, 1984] Bibel, W. and Buchberger, B. (1984). Towards a Connection Machine for Logic Inference. *Future Generation Computer Systems*, 1(3):177–188.

[Bibel and Jorrand, 1986] Bibel, W. and Jorrand, P., editors (1986). *Fundamentals of Artificial Intelligence*, volume 232 of *Lecture Notes in Computer Science*, Berlin. Springer.

[Bibel et al., 1987] Bibel, W., Kurfeß, F., Aspetsberger, K., Hintenaus, P., and Schumann, J. (1987). Parallel inference machines. In [Treleaven and Vanneschi, 1987], pages 185–226.

[Bic, 1984] Bic, L. (1984). A Data-Driven Model for Parallel Interpretation of Logic Programs. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1984*, pages 517–523. Institute for New Generation Computer Technology (ICOT).

[Bitar and Chen, 1990] Bitar, P. and Chen, C. (1990). The OR+AND Modeling Framework for Parallel Prolog Models. UCB/CSD 90/604, Computer Science Division, University of California, Berkeley, CA 94720.

[Blelloch, 1989] Blelloch, G. (1989). Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538.

[Blelloch and Sabot, 1990] Blelloch, G. and Sabot, G. (1990). Compiling collection-oriented languages onto massively parallel computers. *Parallel and Distributed Computing*, 8(2):119–134.

[Böck, 1989] Böck, K.-H. (1989). Studying an application for a parallel logic programming system. Master's thesis, Institut für Informatik, Technische Universität München.

[Bode, 1991] Bode, A., editor (1991). *Distributed Memory Computing. 2nd European Conference, EDMCC2*, number 487 in Lecture Notes in Computer Science, Munich, FRG. Apringer.

[Borgwardt, 1984] Borgwardt, P. (1984). Parallel Prolog Using Stack Segments on Shared Memory Multiprocessors. In *International Symposium On Logic Programming*, Atlantic City, NJ.

[Bose et al., 1989] Bose, S., Clarke, E. M., Long, D. E., and Spiro, M. (1989). Parthenon. Technical report, LICS.

[Brogi and Gorrieri, 1989] Brogi, A. and Gorrieri, R. (1989). A Distributed, Net Oriented Semantics for Delta Prolog. In *TAPSOFT '89*, pages 162–177.

[Caferra and Jorrand, 1985] Caferra, R. and Jorrand, P. (1985). Unification with Refined Linearity Check as a Network of Parallell Processes. Technical report, LIFIA, Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle IMAG, Grenoble, France.

[Chandy and Misra, 1988] Chandy, K. M. and Misra, J. (1988). *Parallel Program Design*. Addison-Wesley, Reading, MA.

[Chassin de Kergommeaux et al., 1988] Chassin de Kergommeaux, J., Baron, U., Rapp, W., and Ratcliffe, M. (1988). Performance Analysis of Parallel Prolog: A Correlated Approach. Technical report, ECRC Munich.

[Chassin de Kergommeaux et al., 1989] Chassin de Kergommeaux, J., Codognet, P., Robert, P., and Syre, J.-C. (1989). Une programmation logique parallele: premiere partie: Langages gardes. *Technique et Science Informatiques*, 8:205–224.

[Cheese, 1991] Cheese, A. (1991). Implementing committed-choice logic programming languages on distributed memory computers. In [Bode, 1991].

[Chen et al., 1988] Chen, C., Singhal, A., and Patt, Y. N. (1988). PUP: An Architecture to Exploit Parallel Unification in Prolog. Technical Report UCB/CSD 88/414, University of California, Computer Science Department), Berkeley, CA 94720.

[Chengzheng and Yungui, 1990] Chengzheng, S. and Yungui, C. (1990). The OR-forest-based parallel execution model of logic programs. *Future Generation Computer Systems*, 6(1):25–34.

[Chu and Itano, 1984] Chu, Y. and Itano, K. (1984). Organisation of a Parallel PROLOG Machine. *Proc. Intern. Workshop on HLCA*.

[Ciepielewski and Haridi, 1984a] Ciepielewski, A. and Haridi, S. (1984a). Control of Activities in the OR-parallel Token Machine. Technical report, Department Of Telecommunications and Comping Systems, Royal Institute of Technology, Stockholm.

[Ciepielewski and Haridi, 1984b] Ciepielewski, A. and Haridi, S. (1984b). Execution of Bagof on the OR-parallel Token Machine. In *International Conference on Fifth Generation Computer Systems*, pages 551–562, Tokyo.

[Citrin, 1988] Citrin, W. (1988). Parallel Unification Scheduling in Prolog. Technical Report UCB/CSD 88/415, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA.

[Clark and Gregory, 1981] Clark, K. and Gregory, S. (1981). A Relational Language for Parallel Programming. In *ACM Conference on Functional Programming Languages and Computer Architecture*, pages 171–178.

[Clark and Gregory, 1984] Clark, K. and Gregory, S. (1984). Notes on Systems Programming in Parlog. *Proc. Of the International Conference On Fifth Generation Computer Systems*, pages 299–306.

[Clark and Gregory, 1986] Clark, K. and Gregory, S. (1986). PARLOG: Parallel Programming in Logic. *ACM Transactions on Programming Languages and Systems*, 1986(8):1–49.

[Clark, 1988] Clark, K. L. (1988). Parlog and Its Applications. *IEEE Transactions on Software Engineering*, 14:1792–1804.

[Colmerauer, 1982] Colmerauer, A. (1982). Prolog and Infinite Trees. *Logic Programming*, pages 231–251.

[Conery, 1983] Conery, J. S. (1983). *The AND/OR Process Model for Parallel Execution of Logic Programs*. PhD thesis, University of California, Irvine. Technical report 204, Department of Information and Computer Science.

[Corbin and Bidoit, 1983] Corbin, J. and Bidoit, M. (1983). A Rehabilitation of Robinson's Unification Algorithm. *Information Processing '83*, pages 909–914.

[Corsini et al., 1989a] Corsini, P., Frosini, G., and Rizzo, L. (1989a). Implementing a Parallel PROLOG Interpreter by Using OCCAM and Transputers. *Microprocessors and Microsystems*, 13(4):271–279.

[Corsini et al., 1989b] Corsini, P., Frosini, G., and Speranza, G. (1989b). The Parallel Interpretation of Logic Programs in Distributed Architectures. *Computer Journal*, 32:29–35.

[Crammond, 1985] Crammond, J. (1985). A Comparative Study of Unification Algorithms for OR-Parallel Execution of Logic Languages. *IEEE*, pages 131–138.

[Crammond, 1986] Crammond, J. A. (1986). An Execution Model for Committed-Choice Nondeterministic Languages. In *Symposium on Logic Programming '86*, pages 148–158.

[Crowl and LeBlanc, 1991] Crowl, L. A. and LeBlanc, T. J. (1991). Architectural adaptability in parallle programming via control abstraction. Technical Report 359, Department of Computer Science, University of Rochester, Rochester, NY 14627.

[Cunha et al., 1989] Cunha, J. C., Ferreira, M. C., and Moniz Pereira, L. (1989). Programming in Delta Prolog. In *Logic Programming Conference '89*.

[Darlington and Reeve, 1983] Darlington, J. and Reeve, M. (1983). ALICE and the Parallel Evaluation of Logic Programs. *10th Annual International Symposium on Computer Architecture*.

[Davison, 1989] Davison (1989). Polka: A parlog object oriented language. Technical report, Dept. of Computing, Imperial College, London, UK.

[de Boer and Palamidessi, 1990a] de Boer, F. S. and Palamidessi, C. (1990a). Concurrent logic programming: Asynchronism and language comparison. Technical Report TR - 6/90, University of Pisa, Department of Computer Science, 56100 Pisa, Italy.

[de Boer and Palamidessi, 1990b] de Boer, F. S. and Palamidessi, C. (1990b). A fully abstract model for concurrent logic languages. Technical Report CS-R9046, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands.

[De Nicola and Ferrari, 1990] De Nicola, R. and Ferrari, G. (1990). Observational logics and concurrency models. Technical Report TR - 10/90, University of Pisa, Department of Computer Science, 56100 Pisa, Italy.

[Debray, 1989] Debray, S. K. (1989). Static inference of modes and data dependencies in logic programs. *ACM Transactions on Programming Languages and Systems*, 11(3):419–450.

[Debray et al., 1990] Debray, S. K., Lin, N.-W., and Hermenegildo, M. (1990). Task granularity analysis in logic programs. Technical Report TR 90-16, Department of Computer Science, University of Arizona, Tucson, AZ 85721.

[DeGroot, 1984] DeGroot, D. (1984). Restricted AND-Parallelism. In *International Conference on Fifth Generation Computer Systems*, pages 471–478. Institute for New Generation Computer Technology (ICOT).

[DeGroot, 1988] DeGroot, D. (1988). A Technique for Compiling Execution Graph Expressions for Restricted And-Parallelism in Logic Programs. *Journal of Parallel and Distributed Computing*, (5):494–516.

[DeGroot, 1990] DeGroot, D. (1990). On the inherently speculative nature of parallel logic programming. In *North American Conference on Logic Programming*.

[Delcher and Kasif, 1989] Delcher, A. and Kasif, S. (1989). Some results in the complexity of exploiting data dependency in parallel logic programs. *Logic Programming*, 6:229–241.

[Delcher and Kasif, 1988] Delcher, A. L. and Kasif, S. (1988). Efficient Parallel Term Matching. Technical report, Computer Science Department, Johns Hopkins University, Baltimore, MD 21218.

[Delgado-Rannauro et al., 1991] Delgado-Rannauro, S. A., Dorochevsky, M., Scherman, K., Véron, A., and Xu, J. (1991). A shared environment parallel logic programming system on distributed memory architectures. In [Bode, 1991].

[Despain et al., 1986] Despain, A., Patt, Y., Dobry, T., Chang, J., and Citrin, W. (1986). High Performance Prolog: The Multiplicative Effect of Several Levels of Implementation. In *COMPCON 86*, Berkeley.

[Despain and Patt, 1985] Despain, A. M. and Patt, Y. N. (1985). Aquarius - A High Performance Computing System for Symbolic and Numeric Applications. In *COMPCON '85*, Berkeley, CA.

[Diel et al., 1986] Diel, H., Lenz, N., and Welsch, H. M. (1986). System Structure for Parallel Logic Programming. *Future Generation Computer Systems*, 2:225–231.

[Dixon and deKleer, 1988] Dixon, M. and deKleer, J. (1988). Massively parallel assumption-based truth maintenance. In Smith, R. G. and Mitchell, T. M., editors, *Seventh National Conference on Artificial Intelligence*, volume 1/2, pages 199–204, St. Paul, MN. American Association for Artificial Intelligence, Morgan Kaufman.

[Dwork et al., 1986] Dwork, C., Kanellakis, P., and Stockmeyer, L. (1986). Parallel Algorithms for Term Matching. In *CADE '86*, pages 416–430, Berlin. Springer.

[Dwork et al., 1988] Dwork, C., Kanellakis, P. C., and Stockmeyer, L. (1988). Parallel algorithms for term matching. *SIAM Journal of Computing*, 17(4):711–731.

[Dyer, 1989] Dyer, M. G. (1989). Symbolic processing techniques in connectionist networks and their application to high-level cognitive tasks. In Brauer, W. and Freksa, C., editors, *International GI Congress on Knowledge Based Systems*, Informatik Fachberichte, Munich. Springer.

[Eckmiller et al., 1990] Eckmiller, R., Hartmann, G., and Hauske, G., editors (1990). *Parallel Processing in Neural Systems and Computers*. Elsevier.

[Eisenstadt and Brayshaw, 1988] Eisenstadt, M. and Brayshaw, M. (1988). The Transparent Prolog Machine (TPM): An Execution Model and Graphical Debugger for Logic Programming. *Logic Programming*, pages 277–342.

[Eliens, 1991a] Eliens, A. (1991a). Distributed logic programming for artificial intelligence. *AI Communications*, 4(1):11–21.

[Eliens, 1991b] Eliens, A. (1991b). *DLP - A Language for Distributed Logic Programming*. PhD thesis, Centre for Mathematics and Computer Science, Amsterdam, Netherlands.

[Engels, 1988] Engels, J. (1988). A Model for Or-parallel Execution of (Full) Prolog and its Proposed Implementation. Technical report, Institut für Informatik III, Universität Bonn, Bonn.

[Ertel, 1990] Ertel, W. (1990). Random competition: A simple, but efficient method for parallelizing inference systems. Technical Report FKI 143-90, Institut für Informatik, Technische Universität München.

[Ertel, 1991] Ertel, W. (1991). Performance of competitive or-parallelism. ICLP Workshop on Parallel Inferencing.

[Ertel et al., 1989] Ertel, W., Kurfeß, F., Letz, R., Pandolfi, X., and Schumann, J. (1989). PARTHEO: A Parallel Inference Machine. In *PARLE '89*.

[Fagin, 1990] Fagin, B. (1990). Data-parallel logic programming. In *North American Conference on Logic Programming*.

[Fagin, 1991] Fagin, B. (1991). Data-parallel logic programming systems. Technical report, Thayer School of Engineering, Dartmouth College, Hanover, NH 03755.

[Fagin and Despain, 1987] Fagin, B. and Despain, A. M. (1987). Performance Studies of a Parallel Prolog Architecture. Technical report, Computer Science Division, University of California, Berkeley, CA. preprint from ISCA, June '87.

[Fagin and Despain, 1990] Fagin, B. S. and Despain, A. M. (1990). The performance of parallel Prolog programs. *IEEE Transactions on Computers*, 39(12):1434–1445.

[Fahlman and Hinton, 1987] Fahlman, S. and Hinton, G. (1987). Connectionist Architectures for Artificial Intelligence. *Computer*, 20:100–118.

[Feldman, 1990] Feldman, J. A. (1990). Conventional and connectionist parallel computation. GMD-Spiegel.

[Foster, 1990] Foster, I. (1990). *Systems Programming in Parallel Logic Languages*. Prentice Hall.

[Foster and Taylor, 1987] Foster, I. and Taylor, S. (1987). Flat Parlog: A Basis for Comparison. *Parallel Programming*, 16:87–125.

[Foster and Taylor, 1989] Foster, I. and Taylor, S. (1989). *STRAND: New Concepts in Parallel Programming*. Prentice Hall.

[Fronhöfer and Kurfeß, 1987] Fronhöfer, B. and Kurfeß, F. (1987). Cooperative Competition: A modest proposal concerning the use of multi-processor systems for automated reasoning. Technical report, Institut für Informatik, Technische Universität München.

[Futo, 1988] Futo, I. (1988). Parallele Programmierung in CS-Prolog. *Artificial Intelligence Newsletter*, 9, 10:13–15, 16–19.

[Futo and Kacsuk, 1989] Futo, I. and Kacsuk, P. (1989). CS-Prolog on multitransputer systems. *Microprocessors and Microsystems*, 13:103–112.

[Georgescu, 1986] Georgescu, I. (1986). An Inference Processor based on reactive memory. Technical report, Institute for Computers and Informatics, Department of Robotics and Artificial Intelligence, Bucharest.

[Giambiasi et al., 1989] Giambiasi, N., Lbath, R., and Touzet, C. (1989). Une approche connexionniste pour calculer l'implication floue dans les systemes a base de regles. Technical report, Universite de Nimes.

[Gonzalez-Rubio et al., 1987] Gonzalez-Rubio, R., Bradier, A., and Rohmer, J. (1987). DDC Delta Driven Computer - a Parallel Machine for Symbolic Processing. In [Treleaven and Vanneschi, 1987].

[Goto et al., 1983] Goto, A., Aida, H., Maruyama, T., Yuhara, M., Tanaka, H., and Moto-OKA, T. (1983). A Highly Parallel Inference Engine: PIE. In *Logic Programming Conference*. Institute for New Generation Computer Technology (ICOT).

[Goto et al., 1988] Goto, A., Sato, M., Nakajima, K., Taki, K., and Matsumoto, A. (1988). Overview of the Parallel Inference Machine Architecture (PIM). In *International Conference on Fifth Generation Computer Systems*, pages 209–229, Tokyo. Institute for New Generation Computer Technology (ICOT).

[Goto et al., 1984] Goto, A., Tanaka, H., and Moto-Oka, T. (1984). Highly Parallel Inference Engine: PIE. Goal Rewriting Model and Machine Architecture. *New Generation Computing*.

[Goto and Uchida, 1985] Goto, A. and Uchida, S. (1985). Current Research Status of PIM: Parallel Inference Machine. In *Third Japanese-Swedish Workshop*, number TM - 140. Institute for New Generation Computer Technology (ICOT).

[Goto and Uchida, 1986] Goto, A. and Uchida, S. (1986). Toward a High Performance Parallel Inference Machine – The Intermediate State Plan of PIM –. Technical report, Institute for New Generation Computer Technology (ICOT).

[Goto and Uchida, 1987] Goto, A. and Uchida, S. (1987). Towards a High Performance Parallel Inference Machine - The Intermediate Stage Plan for PIM. In [Treleaven and Vanneschi, 1987].

[Gregory, 1984] Gregory, S. (1984). Implementing PARLOG on the ALICE Machine. Technical report, Imperial College, London.

[Gregory, 1987] Gregory, S. (1987). *Parallel Logic Programming with PARLOG: The Language and its Implementation*. Addison Wesley.

[Gregory et al., 1989] Gregory, S., Foster, I. T., Burt, A. D., and Ringwood, G. A. (1989). An Abstract Machine for the Implementation of PARLOG on Uniprocessors. *New Generation Computing*, (6):389–420.

[Güntzer et al., 1986] Güntzer, U., Kiessling, W., and Bayer, R. (1986). Evaluation Paradigms for Deductive Databases: from Systolic to As-You-Please. Technical Report TUM-I-86-05, Institut für Informatik, Technische Universität München.

[Hager and Moser, 1989] Hager, J. and Moser, M. (1989). An Approach to Parallel Unification Using Transputers. In *GWAI '89*, pages 83–91. Springer.

[Hailperin and Westphal, 1986] Hailperin, M. and Westphal, H. (1986). A Computational Model for PEPSys. Technical Report CA-16, ECRC.

[Halim, 1986] Halim, Z. (1986). A Data-Driven Machine for Or-Parallel Evaluation of Logic Programs. *New Generation Computing*, Vol.4, No.1:5–33.

[Harland and Jaffar, 1987] Harland, J. and Jaffar, J. (1987). On Parallel Unification for Prolog. *New Generation Computing*, 5:259–279.

[Hattori et al., 1989] Hattori, A., Shinogoi, T., Kumon, K., and Goto, A. (1989). PIM-p: A Hierarchical Parallel Inference Machine. Technical Report TR-514, Institute for New Generation Computer Technology (ICOT), Tokyo, Japan.

[Hellerstein and Shapiro, 1986] Hellerstein, L. and Shapiro, E. (1986). Implementing Parallel Allgorithms in Concurrent Prolog: The Maxflow Experience. *Logic Programming*, 2:157–184.

[Hermenegildo, 1986] Hermenegildo, M. (1986). An Abstract Machine for Restricted AND-Parallel Execution of Logic Programs. In *Third International Conference On Logic Programming 86*, pages 25–39.

[Hermenegildo and Nasr, 1986] Hermenegildo, M. and Nasr, R. (1986). Efficient Management of Backtracking in AND-Parallelism. In *Third International Conference On Logic Programming 86*, pages 40–54.

[Hermenegildo and Rossi, 1989] Hermenegildo, M. and Rossi, F. (1989). On the Correctness and Efficiency of Independent AND-Parallelism in Logic Programs. In *North American Conference on Logic Programming*, pages 369–389. MIT Press.

[Hertzberger and van de Riet, 1984] Hertzberger, L. and van de Riet, R. (1984). Progress in the Fith Generation Inference Architectures. *Future Generations Computer Systems*, 1(2):93–102.

[Hillis, 1985] Hillis, D. W. (1985). *The Connection Machine*. MIT Press, Cambridge, MA.

[Hillis and Steele, 1986] Hillis, W. and Steele, G. (1986). Data Parallel Algorithms. *Communications of the ACM*, 29:1170–1183.

[Hillyer and Shaw, 1983] Hillyer, B. and Shaw, D. (1983). Rapid Execution of AI Production Systems on the NON-VON Supercomputer. Technical report, Department of Computer Science, Columbia University, New York.

[Hölldobler, 1990a] Hölldobler, S. (1990a). CHCL – a connectionist inference system for horn logic based on the connection method. Technical Report TR-90-042, International Computer Science Institute, Berkeley, CA 94704.

[Hölldobler, 1990b] Hölldobler, S. (1990b). A structured connectionist unification algorithm. In *AAAI '90*, pages 587–593. A long version appeared as Technical Report TR-90-012, International Computer Science Institute, Berkeley, CA.

[Hölldobler, 1990c] Hölldobler, S. (1990c). Towards a connectionist inference system. In *Proceedings of the International Symposium on Computational Intelligence*.

[Hölldobler and Kurfeß, 1991] Hölldobler, S. and Kurfeß, F. (1991). CHCL – A Connectionist Inference System. International Computer Science Institute.

[Houri and Shapiro, 1986] Houri, A. and Shapiro, E. (1986). A Sequential Abstract Machine for Flat concurrent Prolog. Technical Report CS86-20, Weizmann Institute of Science, Rehovot, Isael.

[Hwang and Briggs, 1984] Hwang, K. and Briggs, F. (1984). *Computer Architecture and Parallel Processing*. Mc Graw-Hill, New York.

[Hwang et al., 1987] Hwang, K., Ghosh, J., and R.Chokwanyun (1987). Computer Architectures for Artificial Intelligence Processing. *Computer*, 20:19–30.

[Ibañez, 1988] Ibañez, M. B. (1988). Parallel inferencing in first-order logic based on the connection method. In *Artificial Intelligence: Methodology, Systems, Applications '88*. Varna, North-Holland.

[Ibañez, 1989] Ibañez, M. B. (1989). *Inférence parallèle et processus communicants pour les clauses de Horn. Extension au premier ordre par la méthode de connexion*. PhD thesis, I.N.P. de Grenoble, France.

[ICLP91, 1991] ICLP91 (1991). *ICLP 91 Workshop on Parallel Execution of Logic Programs*, Paris, France.

[Ino and Koelbl, 1988] Ino, E. and Koelbl, D. (1988). Sequentielle und parallele Architekturansätze für logische Programmiersprachen. *Informatik Forschung und Entwicklung*, 3:182–194.

[Ito et al., 1987] Ito, N., Kuno, E., and Oohara, T. (1987). Efficient Stream Processing in GHC and Its Evaluation on a Parallel Inference Machine. *Journal of Information Processing*, 10:237–244.

[Ito et al., 1983a] Ito, N., Masuda, K., and Shimizu, H. (August 1983a). Parallel Prolog Machine. Technical report, Institute for New Generation Computer Technology (ICOT), Tokyo.

[Ito and Masuda, 1983] Ito, N. and Masuda, Y. (1983). Parallel Inference Machine Based on the Data Flow Model. Technical Report TR-033, Institute for New Generation Computer Technology (ICOT).

[Ito et al., 1983b] Ito, N., Onai, R., Masuda, K., and Shimizu, H. (1983b). Parallel Prolog Machine Based on Data Flow Mechanism. In *Logic Programming Conference '83*. Institute for New Generation Computer Technology (ICOT).

[Jorrand, 1986] Jorrand, P. (1986). Term Rewriting as a Basis for the Design of a Functional and Parallel Programming Language. A case study: the Language FP2. In [Bibel and Jorrand, 1986], pages 221–276.

[Jorrand, 1987] Jorrand, P. (1987). Design and Implementation of a Parallel Inference Machine for First-Order Logic: An Overview. In *PARLE 87*, volume 258 of *Lecture Notes in Computer Science*, Berlin. Springer.

[Kacsuk, 1991] Kacsuk, P. (1991). A Parallel PROLOG Abstract Machine and its Multi-Transputer Implementation. *Computer*, 34(1):52–63.

[Kacsuk and Bale, 1987] Kacsuk, P. and Bale, A. (1987). DAP Prolog: A Set-oriented Approach to Prolog. *Computer*, 30(5):393–403.

[Kahn, 1986] Kahn, K. e. a. (1986). Objects in concurrent logic languages. In *OOPSLA 86*, pages 242–257.

[Kalé, 1988] Kalé, L. (1988). A Tree Representation for Parallel Problem Solving. In *AAAI '88*, pages 677–681.

[Kalé, 1985] Kalé, L. V. (1985). *Parallel Architectures for Problem Solving.* PhD thesis, State University of New York, Stony Brook.

[Kalé, 1987] Kalé, L. V. (1987). Completeness and full parallelism of parallel logic programming schemes. In *IEEE Symposium on Logic Programming*, pages 125–133, San Francisco, CA. IEEE.

[Kalé, 1989] Kalé, L. V. (1989). The REDUCE OR Process Model for Parallel Execution of Logic Programs. *Journal of Logic Programming.*

[Kalé and Ramkumar, 1990] Kalé, L. V. and Ramkumar, B. (1990). Joining AND Parallel Solutions in AND/OR Parallel Systems: Part I - Static Analysis. In *ICLP '90.*

[Kalé et al., 1988] Kalé, L. V., Ramkumar, B., and Shu, W. (1988). A Memory Organisation Independent Binding Environment for AND and OR Parallel Execution of Logic Programs. In *ICLP '88*, volume 2, University of Illinois at Urbana-Champaign.

[Kalé and Saletore, 1988] Kalé, L. V. and Saletore, V. (1988). Obtaining first solution faster in parallel problem solving. Technical Report UIUCDCS-R-88-1481, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

[Kaplan, 1988] Kaplan, S. (1988). Algorithmic complexity of logic programs. In *International Conference and Symposium on Logic Programming*, pages 780–793, Seattle, WA.

[Karam, 1988] Karam (1988). Prototyping Concurrent Systems with Multilog. Technical report, Department of Systems and Computer Engineering, Carleton University.

[Kasif et al., 1983] Kasif, S., Kohli, M., and Minker, J. (1983). PRISM: A Parallel Inference System for Problem Solving. In *Logic Programming Workshop '83*, pages 123–152, Lisboa, Portugal. Universidade Nova de Lisboa.

[Kasif and Minker,] Kasif, S. and Minker, J. The Intelligent Channel: A Scheme for Result Sharing in Logic Programs. Technical report, University of Maryland.

[Kasif et al., 1987] Kasif, S., Reif, J. H., and Sherlekar, D. D. (1987). Formula Dissection: A Parallel Algorithm for Constraint Satisfaction. In *IJCAI '87.*

[Kibler and Conery,] Kibler, D. and Conery, J. Parallelism in AI Programs. Technical report, Irvine Computational Intelligence Project, Information and Computer Science Department, University of California, Irvine.

[Kimura and Chikayama, 1987] Kimura, Y. and Chikayama, T. (1987). An Abstract KL1 Machine and its Instruction Set. Technical Report TR-246, Institute for New Generation Computer Technology (ICOT), Tokyo, Japan.

[Kliger et al., 1988] Kliger, S., Yardeni, E., Kahn, K., and Shapiro, E. (1988). The Language FCP(:,?). In *FGCS '88*, pages 763–783, Tokyo. Institute for New Generation Computer Technology (ICOT).

[Knight, 1989] Knight, K. (1989). Unification: A multidisciplinary survey. *ACM Computing Surveys*, 21(1):93–124.

[Kober, 1988] Kober, R., editor (1988). *Parallelrechner-Architekturen*. Springer, Berlin.

[Kumar et al., 1988] Kumar, V., Ramesh, K., and Rao, V. (1988). Parallel Best-First Search of State-Space Graphs: A Summary of Results. In *AAAI '88*, volume 1, pages 122–127.

[Kumon et al., 1986] Kumon, K., Masuzawa, H., Itashiki, A., Satoh, K., and Sohma, Y. (1986). Kabu-Wake: A New Parallel Inference Method and its Evaluation. *IEEE*, pages 168–172.

[Kung, 1985] Kung, C.-H. (1985). High Parallelism and a Proof Procedure. *Decision Support Systems*, 1:323–331.

[Kurfeß, 1988] Kurfeß, F. (1988). Logic and reasoning with neural models (extended abstract). *Neural Networks*, 1, Suppl. 1 (Abstracts of INNS 88):192.

[Kurfeß, 1990] Kurfeß, F. (1990). *Parallelism in Logic — Its Potential for Performance and Program Development*. PhD thesis, Institut für Informatik, Technische Universität München. published as book by Vieweg Verlag, Wiesbaden (1991).

[Kurfeß, 1991] Kurfeß, F. (1991). Massive parallelism in inference systems. IJCAI '91 Workshop on Parallel Processing for Artificial Intelligence.

[Kurfeß et al., 1989] Kurfeß, F., Pandolfi, X., Belmesk, Z., Ertel, W., Letz, R., and Schumann, J. (1989). PARTHEO and FP2: Design of a parallel inference machine. In [Treleaven, 1989], chapter 9.

[Kurfeß and Reich, 1989] Kurfeß, F. and Reich, M. (1989). Logic and reasoning with neural models. In *Connectionism in Perspective*, pages 365–376, Amsterdam. Elsevier.

[Kurozumi, 1989] Kurozumi, T. (1989). Outline of the fifth generation computer systems project and ICOT activities. Technical Report TR-523, Institute for New Generation Computer Technology (ICOT), Tokyo, Japan.

[Lake, 1988] Lake, T. (1988). Languages for Parallel Processing (Sprachen für die parallele Datenverarbeitung). *Informationstechnik it*, 30(2).

[Lange and Dyer, 1989] Lange, T. E. and Dyer, M. G. (1989). High-level inferencing in a connectionist network. *Connection Science*, 1:181–217.

[Levi, 1986] Levi, G. (1986). Concurrency Issues in Logic Languages. In [Treleaven and Vanneschi, 1987].

[Levi and Palamidessi, 1988] Levi, G. and Palamidessi, C. (1988). Contributions to the Semantics of Logic Perpetual Processes. *Acta Informatica*, 25:691–711.

[Levy, 1986a] Levy, J. (1986a). CFL-A Concurrent Functional Language Embedded in a Concurrent Logic Programming Environment. Technical report, Weizmann Institute of Science, Rehovot, Israel.

[Levy, 1986b] Levy, J. (1986b). Shared Memory Execution of Committed-Choice Languages. In *Conference On Logic Programming 86*, pages 299–312.

[Levy and Friedmann, 1986] Levy, J. and Friedmann, N. (1986). Concurrent Prolog Implementations - Two New Schemes. Technical report, Weizmann Institute of Science, Rehovot, Israel.

[Li and Wah, 1985] Li, G. and Wah, B. (1985). MANIP-2: A Multicomputer Architecture for Evaluating Logic Programs. *IEEE*, pages 123–130.

[Lichtenwalder, 1988] Lichtenwalder, K. (1988). Spezifikation einer parallelen Inferenzmaschine in Hinblick auf ein Transputersystem. Master's thesis, Institut für Informatik, Technische Universität München.

[Lin and Kumar, 1988] Lin, Y.-J. and Kumar, V. (1988). An Execution Model for Exploiting AND-Parallelism in Logic Programs. *New Generation Computing*, 5:393–425.

[Lusk et al., 1988] Lusk, E., Butler, R., Disz, T., Olson, R., Overbeek, R., Stevens, R., Warren, D. H. D., Calderwood, A., Szeredi, P., Haridi, S., Brand, P., Carlsson, M., Ciepielewski, A., and Haussmann, B. (1988). The AURORA OR-Parallel PROLOG System. In *International Conference on Fifth Generation Computer Systems*, pages 819–830.

[Lütke-Holz, 1989] Lütke-Holz, B. (1989). Simulation eines parallelen Hornklauselinterpreters nach dem Prinzip der Cooperative Competition. Master's thesis, Institut für Informatik, Technische Universität München.

[Mariyama and et al., 1983] Mariyama, T. and et al. (1983). A Highly Parallel Inference Engine PIE. In *Electronic Computer Society of IEEE of Japan*, volume EC 83-39, Japan.

[Matsuda and Kokata, 1985] Matsuda, H. and Kokata, M. e. a. (1985). Parallel Prolog Machine PARK: Its Hardware Structure and Prolog System. In *Conference on Logic Programming '85*, pages 148–158.

[Matsumoto, 1985] Matsumoto, H. (1985). A Static Analysis of Prolog Programs. *SIGPLAN Notices*, 20(10):48–59.

[Mayr and Reich, 1988] Mayr, K. and Reich, M. (1988). Hochparallele Algorithmen für das Erfüllbarkeitsproblem in der Aussagenlogik – implementiert auf einem Simulator für neuronale Netze. Technical report, Institut für Informatik, Technische Universität München.

[McCorduck, 1983] McCorduck, P. (1983). Introduction to the Fifth Generation. *Communications of the ACM*, 6(9):629–645.

[Meseguer, 1990a] Meseguer, J. (1990a). Conditional rewriting logic: Deduction, models and concurrency. Technical Report SRI-CSL-90-14, SRI International, Menlo Park, CA 94025.

[Meseguer, 1990b] Meseguer, J. (1990b). A logical theory of concurrent objects. Technical Report SRI-CSL-90-07, SRI International, Menlo Park, CA 94025.

[Millroth, 1991] Millroth, H. (1991). Reforming compilation of logic programs. In *ILPS 91*.

[Mills, 1989] Mills, J. W. (1989). A pipelined architecture for logic programming with a complex but single-cycle instruction set. Technical Report TR 284, Computer Science Department, Indiana University, Bloomington, IN 47405.

[Mills, 1990] Mills, J. W. (1990). Connectionist logic programming. Technical Report TR 315, Computer Science Department, Indiana University, Bloomington IN 47405.

[Mills et al., 1990] Mills, J. W., Beavers, M. G., and Daffinger, C. A. (1990). Lukasiewicz logic arrays. Technical Report TR 296, Computer Science Department, Indiana University, Bloomington, IN 47405.

[Mills and Daffinger, 1990a] Mills, J. W. and Daffinger, C. A. (1990a). An Analog VLSI Array Processor for Classical and Connectionist AI. Technical Report TR 313, Computer Science Department, Indiana University, Bloomington, IN 47405.

[Mills and Daffinger, 1990b] Mills, J. W. and Daffinger, C. A. (1990b). CMOS VLSI Lukasiewicz Logic Arrays. Technical Report TR 312, Computer Science Department, Indiana University, Bloomington, IN 47405.

[Minsky, 1990] Minsky, M. (1990). Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy. In *Frontiers of Artificial Intelligence*, chapter 9, pages 218–243. MIT Press.

[Moniz Pereira et al., 1988] Moniz Pereira, L., Monteiro, L., and Cunha, Jose C.and Aparicio, J. (1988). Concurrency and Communication in Delta Prolog. In *IEEE International Specialists Seminar on The Design and Applications of Parallel Digital Processors*, pages 94–104, Lisbon.

[Moto-Oka and Fuchi, 1983] Moto-Oka, T. and Fuchi, K. (1983). The architectures in the fifth generation computer. *Information Processing*.

[Moto-Oka et al., 1984] Moto-Oka, T., Tanaka, H., Aida, H., Hirata, K., and Maruyama, T. (1984). The Architecture of a Parallel Inference Engine PIE. In *Conference on Fifth Generation Computer Systems*, pages 479–488. Institute for New Generation Computer Technology (ICOT).

[Muller, 1984] Muller, J.-P. (1984). Paralog: A Parallel Logic Programming System. In *ECAI '86*, pages 115–119. Elsevier.

[Munsch, 1989] Munsch, F. (1989). Ausnutzung von Parallelität bei Theorembeweisern durch Kooperation. Master's thesis, Institut für Informatik, Technische Universität München.

[Murakami et al., 1984] Murakami, K., Kakuta, T., and Onai, R. (1984). Architectures and hardware systems: Parallel inference machine and knowledge base machine. *Fifth Generation Computer Systems*, pages 18–35.

[Nilsson and Tanaka, 1988] Nilsson, M. and Tanaka, H. (1988). Massively Parallel Implementation of Flat GHC on the Connection Machine. In *International Conference on Fifth Generation Computer Systems*, pages 1031–1040.

[NSF / ICOT, 1990] NSF / ICOT (1990). *NSF / ICOT Joint Workshop on Parallel Logic Programming and Knowledge Representation*, Tokyo, Japan.

[Ohki et al., 1987] Ohki, M., Takeuchi, A., and Furukawa, K. (1987). An Object-oriented Language Based on the Parallel Logic Programming Language KL1. In *Conference on Logic Programming '87*, pages 894–909. MIT Press.

[Onai et al., 1984] Onai, R., Asou, M., and Takeuchi, A. (1984). An approach to a parallel inference machine based on control- driven and data-driven mechanisms. Technical Report TR-042, Institute for New Generation Computer Technology (ICOT).

[Onai and et al., 1984] Onai, R. and et al. (May 1984). Analysis of Sequential Prolog Programs. Technical Report 48, Tokyo.

[Onai et al.,] Onai, R., Shimizu, H., Masuda, K., Matsumoto, A., and Aso, M. Architecture and evaluation of a reduction-based parallel inference machine: Pim-r. Technical report, Institute for New Generation Computer Technology (ICOT), Tokyo.

[Park et al., 1988] Park, C.-I., Park, K. H., and Kim, M. (1988). Efficient Backward Execution in AND/OR Process Model. *Information Processing Letters*, 29:191–198.

[Percebois et al., 1991] Percebois, C., Signès, N., and Agnoletto, P. (1991). A compiler for a distributed inference model. In [Bode, 1991].

[Pereira and Nasr, 1984] Pereira, L. and Nasr, R. (1984). Delta-Prolog: A Distributed Logic Programming Language. In *International Conference On Fifth Generation Computer Systems*, pages 283–291. Institute for New Generation Computer Technology (ICOT).

[Peterson and Stickel, 1982] Peterson, G. and Stickel, M. (1982). Complete Systems of Reductions Using Associative AND/OR Commutative Unifications. Technical report, SRI International, Menlo Park, CA.

[Pinkas, 1990] Pinkas, G. (1990). Connectionist energy minimization and logic satisfiability. Technical report, Center for Intelligent Computing Systems, Department of Computer Science, Washington University.

[Pinkas, 1991] Pinkas, G. (1991). Symmetric neural networks and propositional logic satisfiability. *Neural Computation*, 3(2):282–291.

[Plaisted, 1984] Plaisted, D. (1984). The Occur-Check Problem in Prolog. *New Generation Computing*, 2:309–322.

[Pollack, 1990] Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46:77–105.

[Ponder and Patt, 1984] Ponder, C. and Patt, Y. (1984). Alternative Proposals for Implementing Prolog Concurrently and Implications Regarding their Respective Microarchitectures. In *17th Annual Microprogramming Workshop*.

[Potter, 1985] Potter, J. (1985). *The Massively Parallel Processor*. MIT Press.

[Powers, 1990a] Powers, D. M. W. (1990a). Compartmentalized Connection Graphs for Logic Programming I: Compartmentalization, Transformation and Examples. SEKI - Report SR-90-16, Fachbereich Informatik, Universität Kaiserslautern, D-6750 Kaiserslautern, Germany.

[Powers, 1990b] Powers, D. M. W. (1990b). Compartmentalized Connection Graphs for Logic Programming II: Parallelism, Indexing and Unification. SEKI - Report SR-90-17, Fachbereich Informatik, Universität Kaiserslautern, D-6750 Kaiserslautern, Germany.

[Ramesh and Ramakrishnan, 1990] Ramesh, R. and Ramakrishnan, I. (1990). Parallel tree pattern matching. *Symbolic Computation*, 9(4):704–716.

[Ramesh et al., 1989] Ramesh, R., Verma, R., Krishnaprasad, T., and Ramakrishnan, I. (1989). Term matching on parallel computers. *Logic Programming*, pages 213–228.

[Ramkumar and Kalé, 1989a] Ramkumar, B. and Kalé, L. V. (1989a). Compiled Execution of the REDUCE-OR Process Model on Multiprocessors. In *NACLP '89*, pages 313–331.

[Ramkumar and Kalé, 1989b] Ramkumar, B. and Kalé, L. V. (1989b). On the Compilation of Parallel Prolog for Shared and Nonshared Memory Machines. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

[Ramkumar and Kalé, 1990] Ramkumar, B. and Kalé, L. V. (1990). A Chare Kernel Implementation of a Parallel Prolog Compiler. In *Second Conference on Principles and Practice of Parallel Programming*, Seattle.

[Rapp, 1988] Rapp, W. (1988). PEPSys Sequential Module on the MX-500 Users Manual. Technical Report PEPSys-26, European Computer Research Center (ECRC), München, München.

[Ratcliffe and Robert, 1986] Ratcliffe, M. and Robert, P. (1986). PEPSys: A Prolog for Parallel Processing. Technical Report CA-17, European Computer Research Center (ECRC), München, München.

[Ratcliffe and Syre, 1987] Ratcliffe, M. and Syre, J.-C. (1987). Virtual Machines for Parallel Architectures. Technical report, European Computer Research Center (ECRC), München, München.

[Ringwood, 1988] Ringwood, G. (1988). Parlog86 and the Dining Logicians. *Communications of the ACM*, 31:10–25.

[Rohmer et al., 1986] Rohmer, J., Gonzalez-Rubio, R., and Bradier, A. (1986). Delta driven computer: A parallel machine for symbolic processing. In [Treleaven and Vanneschi, 1987].

[Safra, 1986] Safra, S. (1986). Partial Evaluation of Concurrent Prolog and its Implication. Technical Report CS86-24, Weizmann Institute of Science.

[Saletore and Kalé, 1990] Saletore, V. A. and Kalé, L. V. (1990). Consistent Linear Speedups to a First Solution in Parallel State-Space Search. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign.

[Saletore et al., 1990] Saletore, V. A., Ramkumar, B., and Kalé, L. V. (1990). Consistent First Solution Speedups in OR-Parallel Execution of Logic Programs. Technical Report UIUCDCS-R-90-1586, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

[Saraswat, 1986] Saraswat, V. (1986). Problems with Concurrent Prolog. Technical Report CME-CS-86-100, Carnegie-Mellon University.

[Schmid, 1988] Schmid, E. (1988). Implementierung eines parallelen Theroembeweisers auf einem Multiprozessor-Simulator. Fortgeschrittenenpraktikum für informatiker, Institut für Informatik, Technische Universität München, München.

[Schumann, 1991] Schumann, J. (1991). *Efficient Theorem Provers based on an Abstract Machine*. PhD thesis, Institut für Informatik, Technische Universität München.

[Schumann and Letz, 1990] Schumann, J. and Letz, R. (1990). PARTHEO: A High Performance Parallel Theorem Prover. In Stickel, M., editor, *CADE '90*, volume 449 of *Lecture Notes in Computer Science*, pages 40 – 56, Kaiserslautern. Springer.

[Schumann et al., 1990] Schumann, J., Letz, R., and Kurfeß, F. (1990). Tutorial on high-performance theorem provers: Efficient implementation and parallelization. In [Stickel, 1990].

[Schwaab and Tusera, 1988] Schwaab, F. and Tusera, D. (1988). Un Algorithme Distribue pour l'Execution Parallele de Prolog. Technical report, INRIA, Le Chesnay.

[Shapiro, 1983] Shapiro, E. (1983). A Systolic Concurrent PROLOG Machine – Lecture notes on the Bagel. Technical Report TR-035, Institute for New Generation Computer Technology (ICOT), Tokyo.

[Shapiro, 1984] Shapiro, E. (1984). Systolic Programming: A Paradigm of Parallel Processing. *International Conference on Fifth Generation Computer Systems*, pages 458–470.

[Shapiro, 1986] Shapiro, E. (1986). Concurrent prolog: A progress report. *Computer*, 1986(8):44–58. also in [Bibel and Jorrand, 1986].

[Shapiro, 1988] Shapiro, E. (1988). *Concurrent Prolog*. MIT Press.

[Shapiro, 1989a] Shapiro, E. (1989a). The family of concurrent logic programming languages. *ACM Computing Surveys*, 21(3):413–510.

[Shapiro, 1989b] Shapiro, E. (1989b). Or-Parallel PROLOG in Flat Concurrent PROLOG. *Logic Programming*, (6):243–267.

[Shapiro and Takeuchi, 1983] Shapiro, E. and Takeuchi, A. (1983). Object-Oriented Programming in Concurrent Prolog. *New Generation Computing*, (1):25–48.

[Shastri, 1988] Shastri, L. (1988). A connectionist approach to knowledge representation and limited inference. *Cognitive Science*, 12:331–392.

[Shastri and Ajjanagadde, 1989] Shastri, L. and Ajjanagadde, V. (1989). A connectionist system for rule based reasoning with multi-place predicates and variables. Technical report, University of Pennsylvania, Computer and Information Science Department, Philadelphia.

[Shastri and Ajjanagadde, 1990] Shastri, L. and Ajjanagadde, V. (1990). From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings. Technical Report MS-CIS-90-05, Computer And Information Science Department, University of Pennsylvania, Philadelphia, PA 19104.

[Shastri and Feldman, 1985] Shastri, L. and Feldman, J. A. (1985). Evidential Reasoning in Semantic Networks: A Formal Theory. In *IJCAI '85*, pages 465–474.

[Shaw, 1981] Shaw, D. (1981). NON-VON: A Parallel Machine Architecture for Knowledge Based Information Processing. In *IJCAI '81*, pages 961–963, Vancouver.

[Shaw, 1987] Shaw, D. (1987). On the range of applicability of an artificial intelligence machine. *Artificial Intelligence*, 32:252–172.

[Shen and Warren, 1987] Shen, K. and Warren, D. (1987). A simulation study of the Argonne model for OR-parallel execution of Prolog. In *Symposium on Logic Programming '87*, pages 54–86.

[Shrobe et al., 1988] Shrobe, H., Aspinall, J., and Mayle, N. (1988). Towards A Virtual Parallel Inference Engine. In *AAAI '88*, pages 654–659.

[Singhal, 1990] Singhal, A. (1990). *Exploiting Fine Grain Parallelism in Prolog.* PhD thesis, University of California, Berkeley. TR CSD 90/588.

[Smolensky, 1987] Smolensky, P. (1987). On variable binding and the representation of symbolic structures in connectionist systems. CU-CS 355-87, Department of Computer Science and Institute of Cognitive Science, University of Colorado.

[Sohma et al., 1985] Sohma, Y., Satoh, K., Kumon, K., Masuzawa, H., and Itashiki, A. (1985). A new parallel inference mechanism based on sequential processing. In *IFIP TC-10 Working Conference on Fifth Generation Computer Architecture*, UMIST, Manchester.

[Stanfill, 1988] Stanfill, C. (1988). Parallel computing for information retrieval. Technical Report DR88-1, Thinking Machines Corporation, Cambridge, MA.

[Stanfill and Waltz, 1986] Stanfill, C. and Waltz, D. (1986). Toward Memory-Based Reasoning. *Communications of the ACM*, 29:1213–1228.

[Stanfill and Waltz, 1988a] Stanfill, C. and Waltz, D. (1988a). Artificial intelligence on the connection machine: A snapshot. Technical Report G88-1, Thinking Machines Corporation, Cambridge, MA.

[Stanfill and Waltz, 1988b] Stanfill, C. and Waltz, D. (1988b). The memory-based reasoning paradigm. In *Case-Based Reasoning Workshop*, pages 414–424, Clearwater Beach, FL.

[Stender, 1987] Stender, J. (1987). Parallele Prolog-Implementierung auf Transputern. *Hard and Soft*, Juli/August 87:17–23.

[Stern, 1988] Stern, A. (1988). *Matrix Logic.* North Holland, Amsterdam.

[Stickel, 1989] Stickel, M., editor (1989). *1989 AAAI Spring Symposium on Representation and Compilation in High Performance Theorem Proving.* SRI International.

[Stickel, 1990] Stickel, M., editor (1990). *CADE '90: 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, Kaiserslautern. Springer.

[Stolcke, 1989] Stolcke, A. (1989). Unification as constraint satisfaction in structured connectionist networks. *Neural Computation*, (1):559–567.

[Stolcke and Wu, 1991] Stolcke, A. and Wu, D. (1991). Tree matching with recursive distributed representations. International Computer Science Institute.

[Stolfo, 1983] Stolfo, S. (1983). The DADO Parallel Computer. Technical report, Department of Computer Science, Columbia University, New York.

[Stolfo, 1987a] Stolfo, S. (1987a). Initial Performance of the DADO-2 Prototype. *Computer*, 20:75–85.

[Stolfo, 1987b] Stolfo, S. (1987b). On the Limitations of Massively Parallel (SIMD) Architectures for Logic Programming. In *US-Japan AI Symposium*.

[Stolfo et al., 1983] Stolfo, S., Miranker, D., and Shaw, D. (1983). Architecture and applications of DADO: A large-scale parallel computer for artificial intelligence. In *IJCAI '83*, pages 850–854, Karlsruhe, BRD.

[Succi and Marino, 1991] Succi, G. and Marino, G. (1991). Data Parallelism in Logic Programming. In [ICLP91, 1991].

[Syre, 1985] Syre, J.-C. (1985). A Review of Computer Architectures for Functional and Logic Programming Systems. Technical report, European Computer Research Center (ECRC), München.

[Syre and Westphal, 1985] Syre, J.-C. and Westphal, H. (1985). A Review of Parallel Models for Logic Programming Languages. Technical report, European Computer Research Center (ECRC), München.

[Szeredi, 1989] Szeredi, P. (1989). Performance Analysis of the Aurora OR-parallel Prolog System. In *North American Conference on Logic Programming*.

[Takeuchi and Furukawa, 1985] Takeuchi, A. and Furukawa, K. (1985). Interprocess Communication in Concurrent Prolog. Technical report, Institute for New Generation Computer Technology (ICOT), Tokyo.

[Takeuchi and Furukawa, 1986] Takeuchi, A. and Furukawa, K. (1986). Parallel Logic Programming Languages. In *Third International Conference On Logic Programming '86*, pages 242–254.

[Takeuchi et al., 1987] Takeuchi, A., Takahashi, K., and Shimizu, H. (1987). A Description Language with AND/OR Parallelism for Concurrent Systems and Its Stream-Based realization. Technical report, Institute for New Generation Computer Technology, Tokyo.

[Tamaki, 1985] Tamaki, H. (1985). A Distributed Unification Scheme for Systolic Logic Programs. pages 552–559. IEEE.

[Tamura and Kanada, 1984] Tamura, N. and Kanada, Y. (1984). Implementing Parallel Prolog on a Multiprocessor Machine. In *International Symposium On Logic Programming '84*, Atlantic City, NJ.

[Tanaka, 1988] Tanaka, J. (1988). Meta-interpreters and Reflective Operations in GHC. In *Future Generation Computer Systems*, pages 775–783, Tokyo. Institute for New Generation Computer Technology (ICOT).

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

[Taylor et al., 1983] Taylor, S., Maio, C., Stolfo, S., and Shaw, D. (1983). Prolog on the DADO Machine: A Parallel System for High-speed Logic Programming. Technical report, Department of Computer Science, Columbia University, New York.

[Tick, 1988] Tick, E. (1988). Compile-time granularity analysis for parallel logic programming languages. In *International Conference on Fifth Generation Computer Systems*, pages 994–1000, Tokyo, Japan.

[Tick and Warren, 1984] Tick, E. and Warren, D. (1984). Towards a Pipelined Prolog Processor. *New Generation Computing*, 2:323–345.

[Touretzky and Hinton, 1985] Touretzky, D. and Hinton, G. (1985). Symbols Among the Neurons: Details of a Connectionist Inference Architecture. In *IJCAI '85*, pages 238–243, Pittsburgh.

[Touretzky and Hinton, 1988] Touretzky, D. S. and Hinton, G. E. (1988). A distributed connectionist production system. *Cognitive Science*, 12:423–466.

[Treleaven, 1989] Treleaven, P. C., editor (1989). *Parallel Computers: Object-Oriented, Functional and Logic*. Wiley, Chichester.

[Treleaven and Refenes, 1985] Treleaven, P. C. and Refenes, A. N. (1985). Fifth Generation and VLSI Architectures. *Future Generation Computer Systems*, 1(6):387–396.

[Treleaven et al., 1987] Treleaven, P. C., Refenes, A. N., Lees, K., and McCabe, S. (1987). Computer Architectures for Artificial Intelligence. In [Treleaven and Vanneschi, 1987].

[Treleaven et al., 1986] Treleaven, P. C., Refenes, A. N., Lees, K. J., and Mccabe, S. C. (1986). Computer Architectures for Artificial Intelligence. Technical report, University College, London; Ferranti Computer Systems, Bracknell; THORN-EMI Central Research Labs., Hayes.

[Treleaven and Vanneschi, 1987] Treleaven, P. C. and Vanneschi, M., editors (1987). *Future Parallel Computers*, volume 272 of *Lecture Notes in Computer Science*, Berlin. Springer.

[Uchida, 1983] Uchida (1983). Inference Machine: From Sequential to Parallel. In *10th Annual International Symposium On Computer Architecture*, Schweden.

[Uchida,] Uchida, S. Inference Machines in FCGS Project. Technical Report TR-278, Institute for New Generation Computer Technology, Tokyo.

[Uchida, 1987] Uchida, S. (1987). Parallel Inference Machines at ICOT. *Future Generation Computer Systems*, (3):245–252.

[Uchida et al., 1986] Uchida, S., K, T., Goto, A., Nakajima, K., Nakashima, H., Yokota, M., Nishikawa, H., Yamamoto, A., and Mitsui, M. (1986). Logic Computers and Japan's FGCS Project. In [Treleaven and Vanneschi, 1987].

[Uchida et al., 1988] Uchida, S., Taki, K., Nakajima, K., Goto, A., and Chikayama, T. (1988). Research and Development of the Parallel Inference Systemin the Intermediate Stage of the FGCS Project. In *International Conference on Fifth Generation Computer Systems*, pages 17–36, Tokyo. Institute for New Generation Computer Technology (ICOT).

[Ueda, 1985] Ueda, K. (1985). Guarded Horn Clauses. Technical Report TR-103, Institute for New Generation Computer Technology (ICOT).

[Ueda, 1986] Ueda, K. (1986). Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard. Technical Report TR-208, Institute for New Generation Computer Technology (ICOT), Tokyo, Japan.

[Ueda, 1989] Ueda, K. (1989). Parallelism in logic programming. Technical Report TR-495, Institute for New Generation Computer Technology (ICOT), Tokyo, Japan.

[Ultsch et al., 1990] Ultsch, A., Hannuschka, R., Hartmann, U., and Weber, V. (1990). Learning of control knowledge for symbolic proofs with backpropagation networks. In [Eckmiller et al., 1990], pages 499–502.

[Vitter and Simons, 1986] Vitter, J. and Simons, R. (1986). New classes for parallel complexity:. *IEEE Transactions on Computers*, 35(5):403–418.

[Wah, 1987] Wah, B. (1987). New Computers for Artificial Intelligence Processing. *Computer*, 20:10–19.

[Waltz, 1990] Waltz, D. L. (1990). Massively Parallel AI. In *Ninth National Conference on Artificial Intelligence*, Boston, MA. American Association for Artificial Intelligence.

[Waltz and Stanfill, 1988a] Waltz, D. L. and Stanfill, C. (1988a). Artificial Intelligence Related Research on the Connection Machine. In *International Conference on Fifth Generation Computer Systems*, pages 1010–1024, Tokyo. Institute for New Generation Computer Technology (ICOT).

[Waltz and Stanfill, 1988b] Waltz, D. L. and Stanfill, C. (1988b). Artificial Intelligence Related Research on the Connection Machine. In *International Conference on Fifth Generation Computer Systems*, pages 1010–1024, Tokyo. Institute for New Generation Computer Technology (ICOT).

[Wang, 1989] Wang, J. (1989). Towards a New Computational Model for Logic Languages. Technical report, Department of Computer Science, University of Essex, Colchester.

[Wang et al., 1990] Wang, J., Marsh, A., and Lavington, S. (1990). Non-WAM Models of Logic Programming and their Support by Novel Parallel Hardware. International Workshop on Massively Parallel Inference Systems.

[Warren, 1987] Warren, D. (1987). The SRI model for OR-parallel execution of Prolog. In *Symposium on Logic Programming '87*, pages 92–102.

[Warren, 1983] Warren, D. H. (1983). An Abstract Prolog Instruction Set. Technical Report 309, SRI International, Artificial Intelligence Center, Menlo Park, California.

[Watzlawik, 1991] Watzlawik, G. (1991). European Declarative System (EDS): Architecture and Interprocess Communication. In [Bode, 1991], pages 485–494.

[Weinbaum and Shapiro, 1986] Weinbaum, D. and Shapiro, E. (1986). Hardware Description and Simulation Using Concurrent Prolog. Technical Report CS86-25, Weizmann Institute of Science, Rehovot, Israel.

[Westphal, 1986] Westphal, H. (1986). Eine Beurteilung paralleler Modelle für Prolog. In *GI-Jahrestagung '86*, pages 227–240, Berlin. Springer.

[Westphal et al., 1987] Westphal, H., Robert, P., Chassin, J., and Syre, J. (1987). The PEPSys model: Combining Backtracking, AND- and OR-parallelism. In *Symposium on Logic Programming '87*, pages 436–448.

[Yamaguchi et al.,] Yamaguchi, T., Tezuka, Y., and Kakusho, O. Parallel Processing of Resolution. Technical report, Osaka University.

[Yang, 1987] Yang, R. (1987). *P-Prolog – A Parallel Logic Programming Language*. World Scientific, Singapore.

[Yasuura, 1984] Yasuura, H. (1984). On Parallel Computational Complexity of Unification. In *International Conference on Fifth Generation Computer Systems*, pages 235–243. Institute for New Generation Computer Technology (ICOT).

[Zhiyi and Shouren, 1990] Zhiyi, H. and Shouren, H. (1990). A compiling approach for exploiting AND-parallelism in logic programs. *Future Generation Computer Systems*, 1(1):35–42.