Towards CIAO-Prolog –
A Parallel Concurrent Constraint System

(Extended Abstract)

M. Hermenegildo and the CLIP Group

Facultad de Informática
Universidad Politécnica de Madrid (UPM)
herme@fi.upm.es

First steps towards a CIAO-Prolog system

We present an overview of some of the current work performed by us in the context of the efficient parallel implementation of logic programming systems. These efforts represent our first steps towards the development of the CIAO (Concurrent Independent And/Or parallel) family of systems – efficient implementations of *non-deterministic, concurrent, constraint logic programming languages*, on sequential and multiprocessor machines. CIAO can be in some ways seen as an evolution of &-Prolog [7] which builds on some of its ideas and adds several important extensions, such as or-parallelism, support for constraints, explicit concurrency and others inspired by proposals such as Muse [1], PNU-Prolog [15], IDIOM [6], DDAS [18], Andorra-I [17], AKL [10], and the extended Andorra model [21].

More than a precisely defined design, at this point CIAO-Prolog should be seen as a target which serves to motivate and direct our current development efforts. This impreciseness is purposely based on our belief that, in order to develop an efficient system with the characteristics that we desire, a number of technologies have to mature and others still have to be developed from scratch. Thus, our main focus at the moment is in the development of some of these technologies, which include improved memory management and scheduling techniques, development of parallelization technology for non-strict forms of independence, efficient combination of and- and or-parallelism, and the extension of current parallelization theory and global analysis tools for dealing with constraint-based languages.

We will start our discussion by dealing with some methodological issues. We will then introduce some of our recent work in the direction mentioned above. Given the space limitations the description will be aimed at providing an overall view of our recent progress and a set of pointers to some relevant recent publications and technical reports which describe our results more fully. We hope that in light of the objective of providing pointers, the reader will be kind enough to excuse the summarized descriptions and the predominance in the references of (at least recent) work of our group.

Control Rules and Parallelism

Many current proposals for parallel or concurrent logic programming languages and models are actually "bundled packages", in the sense that they offer a combined solution affecting a number of issues such as choice of computation rule, concurrency, exploitation of parallelism, etc. This is understandable since certainly a practical model has to offer solutions for all the problems involved. However, the bundled nature of (the description of) models often makes it difficult to compare different ones. It is our view that, in order to be able to perform such comparisons, a "separation analysis" of models isolating their fundamental principles in (at least) the coordinates proposed above must be performed. In fact, we also believe that such un-bundling brings the additional benefit of allowing the identification and study of the fundamental principles involved in a system independent manner and the transference of the valuable features of a system to another. In the following we present some brief ideas on how we believe the separation analysis mentioned above may be approached.

We start by discussing the separation of parallelism and computation rules in logic programming systems. Of the concepts mentioned above, probably the best understood from the formal point of view is that of computation rules. Assuming for example an SLD resolution-based system the "computation rules" amount to a "selection rule" and a "search rule". The objective of computation rules in general is to minimize work, i.e. to reduce the total amount of resolutions needed to obtain an answer. We believe it is useful, at least from a the point of view of analyzing systems, to make a strict distinction between parallelism issues and computation-rule related issues. To this end, we define parallelism as the simultaneous execution of a number of *independent* sequences of resolutions, *taken from those which would have to be performed in any case as determined by the computation rules.* We call each such sequence a *thread* of execution. Note that as soon as there is an *actual* (i.e., run-time) dependency between two sequences, one has to wait for the other and therefore parallelism does not occur for some time. Thus, such sequences contain several threads. Exploiting parallelism means taking a fixed-size computation (determined by the computation rules), splitting it into independent threads related by dependencies (building a dependency graph), and assigning these segments to different agents. Both the partitioning and the agent assignment can be performed statically or dynamically. The objective of parallelism in this definition is simply to perform the same amount of work in less time.

We consider as an example a typical or-parallel system. Let us assume a finite tree, with no cuts or side-effects, and that all solutions are required. In a first approximation we could consider that the computation rules in such a system are the same as in Prolog and thus the same tree is explored and the number of resolution steps is the same. Exploiting (or-)parallelism then means taking branches of the resolution tree (which have no dependencies, given the assumptions) and giving them to different agents. The result is a performance gain that is independent of any performance implications of the computation rule. As is well known, however, if only (any) one solution is needed, then such a system can behave quite differently from Prolog: if the leftmost solution (the one Prolog would find) is deep in the tree, and there is another, shallower solution to its right, the or-parallel system may find this other solution first. Furthermore, it may do this after having explored a different portion of the tree which is potentially smaller (although also potentially bigger). The interesting thing to realize from our point of view is that part of the possible performance gain (that which sometimes produces "super-linear" speedups) comes in a fundamental way from a change in the computation rule, rather than from parallel execution itself. It is not due to the fact that several agents are operating but to the different way in which the tree is being explored ("more breath-first").[1]

A similar phenomenon appears for example in independent and-parallel systems if they incorporate a certain amount of "intelligent failure": computation may be saved. We would like this to be seen as associated to a smarter computation rule that is taking advantage of the knowledge of the independence of some goals. In contrast, also the possibility of performing additional work arises: unless non-failure can be proved ahead of time, and-parallel systems necessarily need to be speculative to a certain degree in order to obtain speedups. However such speculation can in fact be controlled so that no slow down occurs [8].

Another interesting example to consider is the Andorra-I system. The basic Andorra prin-

[1] This can be observed for example by starting a Muse or an Aurora system with several "workers" on a uniprocessor machine. In this experiment is possible sometimes to obtain a performance gain w.r.t. a sequential Prolog system even though there is no parallelism involved – just a *coroutining* computation rule, in this case implemented by the multitasking operating system.

ciple underlying this system states (informally) that deterministic reductions are performed ahead of time and possibly in parallel. This principle would be seen from our point of view as actually two principles, one related to the computation rules and another to parallelism. From the computation rule point of view the bottom line is that deterministic reductions are executed first. This is potentially very useful in practice since it can result in a change (generally a reduction, although the converse may also be true) of the number of resolutions needed to find a solution. Once the computation rule is isolated the remaining part of the rule is related to parallelism and can be seen simply as stating that deterministic reductions can be executed in parallel. Thus, the "parallelism part" of the basic Andorra principle, once isolated from the computation rule part, brings a basic principle to parallelism: that of the general convenience of parallel execution of deterministic threads.

Abstracting Away the Granularity Level

We believe that the separation of computation rule and parallelism issues mentioned above allows enlarging the applicability of the interesting principles brought in by many current models. In this section we argue that moving a principle from one system to another can often be done quite easily if another such "separation" is performed: isolating the principle itself from the *level of granularity* at which it is applied. From the point of view of parallelism, for example, performing such separation means viewing the parallelizing principle involved (e.g., independence, which allows parallelism among non-deterministic threads and determinacy which allows parallelism among dependent threads, as well as others such as granularity and non-failure) as associated to a generic concept of thread, to be particularized for each system, according to the fundamental unit of parallelism used in such system.

For example, following these ideas the notions of determinism in the basic Andorra principle can be applied to the &-Prolog system. The basic unit of parallelism considered when parallelizing programs in the classical &-Prolog tools is the subtree corresponding to the complete resolution of a given goal in the resolvent. If the basic Andorra principle is applied at this level of granularity its implications are that deterministic subtrees can and should be executed in parallel (even if they are "dependent" in the classical sense).

In fact, the converse can also be done: the underlying principles of &-Prolog w.r.t. parallelism –basically its independence rules– can in fact be applied at the granularity level of the Andorra model. The concept of independence in the context of &-Prolog is defined informally as requiring that a part of the execution "will not be affected" by another. Sufficient conditions –strict [8] and non-strict independence [9]– are then defined which are shown to ensure this property. We argue that applying these concepts at the granularity level of the Andorra model gives some new ways of understanding the model and some new solutions for its parallelization. In order to do this it is quite convenient to look at the basic operations in the light of David Warren's *extended* Andorra model.[2] The extended Andorra model brings in the first place the idea of presenting the execution of logic programs as a series of simple, low level operations on and-or trees. In addition to defining a lower level of granularity, the extended Andorra model incorporates some principles which are related in part to parallelism and in part to computation rule related issues such as the above mentioned basic Andorra principle and the avoidance of re-computation of goals.

On the other hand the extended Andorra model also leaves several other issues relatively more open. One example is that of when nondeterministic reductions may take place in parallel. One answer for this important and relatively open issue was given in the instantiation of the model in the AKL language. In AKL the concept of "stability" is defined as

[2]This is understandable, given that adding independent and-parallelism to the basic Andorra model was one of the objectives in the development of its extended version.

follows: a configuration (partial resolvent) is said to be stable if it cannot be affected by other sibling configurations. In that case the operational semantics of AKL allow the non-determinate promotion to proceed. Note that the definition is, not surprisingly, equivalent to that of independence, although applied at a different granularity level. Unfortunately stability/independence is in general an undecidable property. However, applying the work developed in the context of independent and-parallelism at this level of granularity provides sufficient conditions for it. The usefulness of this is underlined by the fact that the current version of AKL incorporates the relatively simple notion of strict independence (i.e. the absence of variable sharing) as its stability rule. However, the presentation above clearly marks the way for incorporating more advanced concepts, such as non-strict independence, as a sufficient condition for the independence/stability rule. As will be mentioned, we are actively working on compile-time detection of non-strict independence, which we believe will be instrumental in this context. Furthermore, and as we will show, when adding constraint support to a system the traditional notions of independence are no longer valid and both new definitions of independence and sufficient conditions for it need to be developed. We believe that the view proposed herein allows the direct application of general results concerning independence in constraint systems to several realms, such as the extended Andorra model and AKL.

Similarly to the separations mentioned above (parallelism vs. computation rule and principles vs. granularity level of their application) we also believe in a separation of "concurrency" from both parallelism and computation rules. We believe concurrency is most useful when it is explicitly controlled by the user and should be separate from the implicit computation rules. This is in contrast with parallelism, which ideally should be transparent to the user, and with smart computation rules of which the user should only be aware in the sense of being able to derive an upper bound on the amount of computation involved in running a program for a given query using that rule. Space limitations prevent us from elaborating more on this topic or that of the separation between concurrency and parallelism. However, an example of an application of the latter can be seen in *schedule analysis*, where the maximal essential components of concurrency are isolated and sequenced to allow the most efficient possible execution of the concurrent program by one agent [11].

Parallelism based on Non-Strict Independence

One of our starting steps is to improve the independence-based detection of parallelism based on information that can be obtained from global analysis using the current state of the art in abstract interpretation. "Non-strict" independence is a more relaxed notion than the traditional notion of "strict" independence which still ensures important "no slow down" properties and can allow considerable more parallelism than strict independence [19]. However, the compile-time parallelization technology developed to date has been based exclusively on strict independence, presumably because of the intrinsic complexity of exploiting non-strict independence. This is related to the fact that non-strict independence cannot be determined "a priori" as strict independence. In [2] we describe some of our recent work filling this gap. Rules and algorithms are provided for detecting and annotating non-strict independence at compile-time. We also propose algorithms for combined compile-time/run-time detection, including run-time checks for this type of parallelism, which in some cases turn out to be different from the traditional groundness and independence checks used for strict independence. The approach is based on the knowledge of certain properties about run-time instantiations of program variables —sharing, groundness, freeness, etc.— for which compile-time technology is available, with new approaches being currently proposed. Rather than dealing with the analysis itself, we present how the analysis results can be used to parallelize programs.

Parallelization in the Presence of Constraints: Independence / Stability

In the CIAO-Prolog system, from the language point of view, we assume a constraint-based, non-deterministic logic programming language. As such, and apart from the concurrency/coroutining primitives, the user language can be viewed as similar to Prolog when working on the Herbrand domain, and to systems such as CLP(R) or CHIP when working over other domains. This implies that both the traditional notions of independence / stability and the traditional global analysis techniques need to be evaluated in this context and, if necessary, extended to deal with the fact that constraint solving is occurring in the actual execution in lieu of unification. We will deal with the first point above in this section by describing (in somewhat more detail than that given in other sections) some recent work which was performed in collaboration with Monash University. The following section will discuss some of our recent progress in extending traditional global analysis techniques to the analysis of constraint-based logic programs.

We start by noting that the traditional notions of strict and non-strict independence which were used in the context of LP are not valid in general for CLP. Consider for example the state $\langle \mathbf{p}(\mathbf{X}) : \mathbf{q}(\mathbf{Z}), \{\mathbf{X} > \mathbf{Y}, \mathbf{Y} > \mathbf{Z}\}\rangle$ in a CLP($\Re$) execution. Although $\mathtt{p}$ and $\mathtt{q}$ do not actually share variables just before their execution (and are thus strictly independent) it is clear that the execution of $\mathtt{p}$ may prune the search space of $\mathtt{q}$. We have proposed alternatives to the traditional notions by defining different levels of independence which are useful for different classes of applications such as goal reordering, intelligent backtracking, and parallelism. We will briefly illustrate the part of our results that applies directly to parallelism.

Let $\exists_{\mathbf{W}} \pi$ be a non-deterministic function which returns a constraint logically equivalent to $\exists \mathbf{V}_1 \exists \mathbf{V}_2 \cdots \mathbf{V_n} \pi$ where variable set $\mathbf{W} = \{\mathbf{V}_1, \ldots, \mathbf{V_n}\}$. Let $\overline{\exists}_{\mathbf{W}} \pi$ be the constraint π restricted to the variables in $\mathbf{W}$. Let $\tilde{\exists} \pi$ stand for "π is satisfiable." We define *strong independence* as follows: Let $\mathbf{g}_1, \mathbf{g}_2$ be two goals, and π a constraint. $\mathbf{g}_1$ is strongly independent of $\mathbf{g}_2$ for π iff all answers of $\langle \mathbf{g}_2, \pi\rangle$ are consistent with all partial answers of $\langle \mathbf{g}_1, \pi\rangle$ (by partial answers we refer to all the states of the constraint store along the derivations that start from the given goal, independently of whether they result in success or failure). We show that preservation of search space w.r.t. for example an original left to right execution is guaranteed when *independently* (i.e. in different environments) executing a set of goals which are strongly independent. The requirement for different environments can be avoided if a symmetric notion of strong independence is used. We refer to this concept as *search independence*: goals $\mathbf{g}_1$ and $\mathbf{g}_2$ are search independent for a constraint π iff all partial answers of $\langle \mathbf{g}_1, \pi\rangle$ are consistent with all partial answers of $\langle \mathbf{g}_2, \pi\rangle$. Note that this is equivalent to the intuitive notion of executions "not affecting" each other.[3]

Determining search independence among goals can be done at run-time, at compile-time (through global analysis), or partly at compile-time and partly at run-time. We will focus on describing sufficient conditions for run-time detection. The reason is that while compile-time detection can be based on the definitions themselves, run-time detection cannot since search independence has been defined in terms of the partial answers produced by the goals. However, in practice most applications require that run-time detection be performed just before executing the goals, and without actually having to execute them (we refer to this as "a priori" detection of independence).

Our first approach is to define conditions which ensure that strong independence must hold for any possible partial answer. We call this notion *projection independence*. Two goals

[3]Note that although these definitions and results are presented at the level of granularity of subtrees they are also applicable at finer levels of granularity.

$\mathbf{g}_1(\bar{\mathbf{x}})$ and $\mathbf{g}_2(\bar{\mathbf{y}})$ are projection independent for π iff $\forall \pi_1, \pi_2 \in \mathbf{Cons}\ \tilde{\exists}(\pi \wedge \bar{\exists}_{\bar{\mathbf{x}}}\pi_1).\tilde{\exists}(\pi \wedge \bar{\exists}_{\bar{\mathbf{y}}}\pi_2) \;\Rightarrow\; \tilde{\exists}(\pi \wedge \bar{\exists}_{\bar{\mathbf{x}}}\pi_1 \wedge \bar{\exists}_{\bar{\mathbf{y}}}\pi_2)$. Intuitively, for all possible pairs of constraints that the goals may pose, consistency with the store when tested in isolation implies consistency when combined. Thus, isolation of the executions cannot prevent any failures from occurring. In fact we show for example that if two goals are projection independent for a given constraint then they are search independent for that constraint. Intuitively this result holds because execution of an atom can only add constraints on local variables and the arguments of the atom.

The definition of projection independence provides a sufficient condition for search independence. However, naive application of this sufficient condition implies testing all possible consistent constraints over the variables of each goal. In order to provide a more useful sufficient condition we also show that two goals $\mathbf{g}_1(\bar{\mathbf{x}})$ and $\mathbf{g}_2(\bar{\mathbf{y}})$ are projection independent for constraint π iff $(\bar{\mathbf{x}} \cap \bar{\mathbf{y}} \subseteq \mathbf{def}(\pi)).(\bar{\exists}_{\bar{\mathbf{x}}}\pi \wedge \bar{\exists}_{\bar{\mathbf{y}}}\pi \Rightarrow \bar{\exists}_{\bar{\mathbf{y}} \cup \bar{\mathbf{x}}}\pi)$ (where $\mathbf{def}(\Pi)$ is the set of variables which π constrains to taking a unique value). This implies that $\mathbf{g}_1(\bar{\mathbf{x}}), \mathbf{g}_2(\bar{\mathbf{y}})$ are search independent for π if $\bar{\mathbf{x}} \cap \bar{\mathbf{y}} \subseteq \mathbf{def}(\pi)$ and $\bar{\exists}_{\bar{\mathbf{x}}}\pi \wedge \bar{\exists}_{\bar{\mathbf{y}}}\pi \Rightarrow \bar{\exists}_{\bar{\mathbf{y}} \cup \bar{\mathbf{x}}}\pi$. Intuitively this means that the results on search space preservation hold iff (a) the goals do not have (free) variables in common w.r.t. π and (b) π can be projected over the variables of each goal and later be "re-built" by each projected constraint without losing "interesting" information for the variables in $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$. For example, consider two goals $\mathbf{g}_1(\mathbf{Y}), \mathbf{g}_2(\mathbf{Z})$ and a constraint $\pi = \{\mathbf{Y} > \mathbf{X}, \mathbf{Z} > \mathbf{X}\}$. We have $\bar{\exists}_{\{\mathbf{Y}\}}\pi = \mathbf{true}$, $\bar{\exists}_{\{\mathbf{Z}\}}\pi = \mathbf{true}$, $\bar{\exists}_{\{\mathbf{Y},\mathbf{Z}\}}\pi = \mathbf{true}$. Therefore, from the results above we know that $\mathbf{g}_1(\mathbf{Y}), \mathbf{g}_2(\mathbf{Z})$ are search independent for π.

In the Herbrand domain, and due to the characteristics of the implementation of the terms and the operation of the unification algorithm, computing the projection function at run-time is immediate (just taking the objects pointed to by the variables) and the condition above simply implies ensuring that goals have no variables in common.[4] However, when constraint domains other than Herbrand are involved, the cost of performing a precise projection may be too high. A pragmatic solution is to find if variables are "linked" through the primitive constraints in the constraint store. Furthermore, we can do better by noticing that we can avoid the link established by variables that are constrained to take a unique value (e.g. $\mathbf{Y}$ and $\mathbf{Z}$ are not linked by $\{\mathbf{X} = 0, \mathbf{Y} > \mathbf{X}, \mathbf{X} > \mathbf{Z}\}$). However, the condition (formally defined in [4]) is somewhat conservative. For instance, although the goals $\mathbf{g}_1(\mathbf{Y}), \mathbf{g}_2(\mathbf{Z})$ are search independent for $\pi = \{\mathbf{Y} > \mathbf{X}, \mathbf{Z} > \mathbf{X}\}$, the condition would associate a link to $\mathbf{Y}$ and $\mathbf{Z}$ due to the transitive closure performed when computing those links. Thus, if projection can be efficiently performed for the particular constraint domain and solver being used it is better to use it to determine search independence at run time.

From the results mentioned above, it may be thought that search space independence is enough for ensuring not only the correctness but also the efficiency of any transformation applied to the search independent goals. Unfortunately, this is not true in general. Modifying the order in which a sequence of primitive constraints is added to the store may have a critical influence on the time spent by the constraint solver algorithm in obtaining the answer, even if the resulting constraint is consistent. In fact, this issue is the core of the reordering application described in [12]. This is because, unlike the cost of resolving an atom, which is independent of other factors, the cost of adding a primitive constraint greatly depends on the current state of the store. This issue of the variance of the cost of adding primitive constraints to the store has been ignored as a factor of negligible influence in traditional logic programming. This is due to the specific characteristics of the standard unification algorithms [16, 13]. However it cannot be ignored in the context of CLP languages. For

[4] Note thus that this condition, when considered in the context of traditional logic programs, is identical to the definition of strict independence among goals given in [8, 9] – the general "a priori" condition for LP.

this reason, we introduce the concept of *constraint solver independence* which, although orthogonal to search space independence, is also needed in order to ensure the efficiency of several optimizations, including parallelism. We claim that most reasonable constraint solvers satisfy "reasonable" levels of solver independence and that, therefore, the efficiency of many optimizations, such as and-parallelism, can be ensured once the adequate search space independence notion is proved to hold for the goals involved in the optimization. These results are more extensively described in [4].

Extending Global Analysis Technology to CLP

As mentioned before, since many optimizations, including independence / stability detection, are greatly aided by (and sometimes even require) global analysis, traditional global analysis techniques have to be extended to deal with the fact that constraint solving is occurring in the actual execution in lieu of unification. In [3] we present and illustrate with an implementation a practical approach to the dataflow analysis of programs written in constraint logic programming (CLP) languages using abstract interpretation. We argue that, from the framework point of view, it suffices to propose quite simple extensions to traditional analysis methods which have already been proved useful and practical and for which efficient fixpoint algorithms have been developed. This is shown by proposing a simple but quite general extension to the analysis of CLP programs of Bruynooghe's traditional framework, and describing its implementation – the "PLAI" system. As the original, the framework is parametric and we provide correctness conditions to be met by the abstract domain related functions to be provided. In this extension constraints are viewed not as "suspended goals" but rather as new information in the store, following the traditional view of CLP. Using this approach, and as an example of its use, a complete, constraint system independent, abstract analysis is presented for approximating definiteness information. The analysis is in fact of quite general applicability. It has been implemented and used in the analysis of CLP(R) and Prolog-III applications. Results from this implementation are also presented which show good efficiency and accuracy for the analysis.

This framework, combined with the ideas of [4] (and [14]) presented in the previous section, is the basis for our current development of automatic parallelization tools for CLP programs, and, in particular, of the parallelizer for the CIAO-Prolog system.

Memory Management and Scheduling in Non-deterministic And-parallel Systems

From our experience with the &-Prolog system implementation [7], the results from the DDAS simulator [18], and from informal conversations with the Andorra-I developers, efficient memory management in systems which exploit and-parallelism is a problem for which current solutions are not completely satisfactory. This appears to be specially the case with and-parallel systems which support don't-know nondeterminism or deep guards. We believe *non-deterministic* and-parallel schemes to be highly interesting in that they present a relatively general set of problems to be solved (including most of those encountered in the memory management of or-parallel only systems) and have chosen to concentrate on their study.

In collaboration with U. of Bristol, we have developed a distributed stack memory management model which allows flexible scheduling of goals. Previously proposed models are lacking in that they impose restrictions on the selection of goals to be executed or they may require a large amount of virtual memory. Our measurements imply that the above mentioned shortcomings can have significant performance impacts, and that the extension that we propose of the "Marker Model" allows flexible scheduling of goals while keeping (virtual) memory consumption down. We also discuss methods for handling forward and backward execution, cut, and roll back. Also, we show that the mechanism proposed for flexible scheduling can be applied to the efficient handling of the very general form of suspension that can occur in systems which combine several types of non-deterministic and-parallelism and advanced computation rules, such as PNU-Prolog, IDIOM [6], DDAS [18], AKL [10], and, in general, those that can be seen as an instantiation of the extended Andorra model [21]. Thus, we believe that the results may be applicable to a whole class of and- and or-parallel systems. Our solutions and results are described more fully in [20].

Incorporating Or-Parallelism: The ACE Approach

Another important issue is the incorporation of Or-parallelism to an and-parallel system. This implies well known problems related to or-parallelism itself, such as the maintenance of several binding environments, as well as new problems such as the interactions of the multiplicity of binding environments and threads of or-parallel computation with the scoping and memory management requirements of and-parallelism. The stack copying approach, as exemplified by the MUSE system, has been shown to be a quite successful alternative for representing multiple environments during or-parallel execution of logic programs. In collaboration with the U. of New Mexico and Bristol U. we have developed an approach for parallel implementation of logic programs, described more fully in [5], which we believe is capable of exploiting both or-parallelism and independent and-parallelism (as well as other types of and-parallelism) in an efficient way using stack copying ideas. This model combines such ideas with proven techniques in the implementation of independent and-parallelism, such as those used in &-Prolog. We show how all solutions to non-deterministic and-parallel goals are found without repetitions. This is done through re-computation as in Prolog (and &-Prolog), i.e., solutions of and-parallel goals are not shared. We propose a scheme for the efficient management of the address space in a way that is compatible with the apparently incompatible requirements of both and- and or-parallelism. This scheme allows incorporating and combining the memory management techniques used in (non-deterministic) and-parallel systems, such as those mentioned in the previous section, and memory management techniques of or-parallel systems, such as incremental copying. We also show how the full Prolog language, with all its extra-logical features, can be supported in our and-or parallel system so that its sequential semantics is preserved. The resulting system retains the advantages of both purely or-parallel systems as well as purely and-parallel systems. The stack copying scheme together with our proposed memory management scheme can also be used to implement models that combine dependent and-parallelism and or-parallelism, such as Andorra and Prometheus.

Acknowledgements

The work presented in this paper is funded in part by CEC DGXIII under ESPRIT contracts 6707 "PARFORCE" and 7195 "ACCLAIM", and by CICYT under contract TIC91-0106-CE. This work is part of the joint effort of the CLIP (Computational Logic, Implementation, and Parallelism) group at UPM which includes Francisco Ballesteros, Francisco Bueno, Daniel Cabeza, Manuel Carro, María José García de la Banda, Luis Gómez, Pedro López García, and Manuel Hermenegildo.

References

[1] K.A.M. Ali and R. Karlsson. The Muse Or-Parallel Prolog Model and its Performance. In *1990 North American Conference on Logic Programming*. MIT Press, October 1990.

[2] D. Cabeza and M. Hermenegildo. Extracting Non-Strict Independent And-parallelism Using Sharing and Freeness Information. Technical Report TR Number CLIP5/92.0, U. of Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, August 1992.

[3] M. García de la Banda and M. Hermenegildo. A Practical Approach to the Global Analysis of Constraint Logic Programs. Technical report, U. of Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, April 1992. Presented at the 1992 FGCS Post-Conference Workshop on Constraint Logic Programming.

[4] M. García de la Banda, M. Hermenegildo, and K. Marriott. Independence in Constraint Logic Programs. Technical report, U. of Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, November 1992.

[5] G. Gupta, M. Hermenegildo, Enrico Pontelli, and Vítor Santos Costa. ACE: And/Or-parallel Copying-based Execution of Logic Programs. Technical report, U. of Madrid (UPM) and U. New Mexico, November 1992.

[6] G. Gupta, V. Santos-Costa, R. Yang, and M. Hermenegildo. IDIOM: Integrating Dependent and-, Independent and-, and Or-parallelism. In *1991 International Logic Programming Symposium*, pages 152–166. MIT Press, October 1991.

[7] M. Hermenegildo and K. Greene. The &-prolog System: Exploiting Independent And-Parallelism. *New Generation Computing*, 9(3,4):233–257, 1991.

[8] M. Hermenegildo and F. Rossi. On the Correctness and Efficiency of Independent And-Parallelism in Logic Programs. In *1989 North American Conference on Logic Programming*, pages 369–390. MIT Press, October 1989.

[9] M. Hermenegildo and F. Rossi. Non-Strict Independent And-Parallelism. In *1990 International Conference on Logic Programming*, pages 237–252. MIT Press, June 1990.

[10] S. Janson and S. Haridi. Programming Paradigms of the Andorra Kernel Language. In *1991 International Logic Programming Symposium*, pages 167–183. MIT Press, 1991.

[11] A. King and P. Soper. Reducing scheduling overheads for concurrent logic programs. In *International Workshop on Processing Declarative Knowledge*, Kaiserslautern, Germany, (1991). Springer-Verlag.

[12] K. Marriott and P. Stuckey. The 3 R's of Optimizing Constraint Logic Programs: Refinement, Removal, and Reordering. In *Proceedings of the 19th. Annual ACM Conf. on Principles of Programming Languages*. ACM, 1992.

[13] A. Martelli and U. Montanari. An Efficient Unification Algorithm. *ACM Transactions on Programming Languages and Systems*, 4(3):258–282, 1982.

[14] U. Montanari, F. Rossi, F. Bueno, M. García de la Banda, and M. Hermenegildo. Contextual Nets and Constraint Logic Programming: Towards a True Concurrent Semantics for CLP. Technical report, T.U. of Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, January 1993. To be Presented at the ICLP'93 Post Conference Workshop on Concurrent Constraint Logic Programming.

[15] L. Naish. Parallelizing NU-Prolog. In *Fifth International Conference and Symposium on Logic Programming*, pages 1546–1564. University of Washington, MIT Press, August 1988.

[16] M. S. Paterson and M. Wegman. Linear Unification. *J. of Computer and System Sciences*, 16(2):158–167, 1978.

[17] V. Santos-Costa, D.H.D. Warren, and R. Yang. Andorra-I: A Parallel Prolog System that Transparently Exploits both And- and Or-parallelism. In *Proceedings of the 3rd. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.* ACM, April 1990.

[18] K. Shen. Exploiting Dependent And-Parallelism in Prolog: The Dynamic, Dependent And-Parallel Scheme. In *Proc. Joint Int'l. Conf. and Symp. on Logic Prog.* MIT Press, 1992.

[19] K. Shen and M. Hermenegildo. A Simulation Study of Or- and Independent And-parallelism. In *1991 International Logic Programming Symposium.* MIT Press, October 1991.

[20] K. Shen and M. Hermenegildo. A Flexible Scheduling and Memory Management Scheme for Non-Deterministic, And-parallel Execution of Logic Programs. Technical report, T.U. of Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, April 1993. Presented at the ICLP'93 Post Conference Workshop on Logic Program Implementation.

[21] D.H.D. Warren. The Extended Andorra Model with Implicit Control. In Sverker Jansson, editor, *Parallel Logic Programming Workshop*, Box 1263, S-163 13 Spanga, SWEDEN, June 1990. SICS.