# The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems

*M. Hermenegildo*[*] *F. Bueno*[*] *D. Cabeza*[*] *M. Carro*[*]
*M. García de la Banda*[†] *P. López*[*] *G. Puebla*[*]

**Abstract**

CIAO is an advanced programming environment supporting Logic and Constraint programming. It offers a simple concurrent kernel on top of which declarative and non-declarative extensions are added via libraries. Libraries are available for supporting the ISO-Prolog standard, several constraint domains, functional and higher order programming, concurrent and distributed programming, internet programming, and others. The source language allows declaring properties of predicates via assertions, including types and modes. Such properties are checked at compile-time or at run-time. The compiler and system architecture are designed to natively support modular global analysis, with the two objectives of proving properties in assertions and performing program optimizations, including transparently exploiting parallelism in programs. The purpose of this paper is to report on recent progress made in the context of the CIAO system, with special emphasis on the capabilities of the compiler, the techniques used for supporting such capabilities, and the results in the areas of program analysis and transformation already obtained with the system.

## 1 Introduction

CIAO is a multi-paradigm compiler, run-time system, and program development environment. It efficiently supports the programming styles of LP, CLP, and CC languages on multiprocessor machines. It also supports distributed execution and Internet/WWW programming. The program development tools in the system include execution visualizers and modular global program analyzers. The latter infer several properties including types, modes, determinacy, non-failure, and independence. Such properties can be checked against user supplied assertions for debugging purposes. Several program transformations (including parallelization) are also performed automatically.

The system offers a simple but versatile kernel language with sequential,

[*] CS Dept., Technical U. of Madrid. {herme,bueno,dcabeza,mcarro,plg,german}@fi.upm.es

[†] Dept. of CS, Monash University, Clayton 3168, Australia. mbanda@bruce.cs.monash.oz.au

parallel, concurrent, and distributed execution capabilities. All extensions to this language and support packages are provided as libraries, which are intended to be portable so that they can be used with little modification in other logic and constraint logic programming systems.

The kernel language is directly supported by a comparatively simple abstract machine, mainly based on the parallelism and concurrency capabilities of the &-Prolog parallel abstract machine (the PWAM [Her86, HG91]). The CIAO system is an evolution of &-Prolog [Her86, HG91], which uses the &-Prolog abstract machine as underlying kernel execution mechanism and significantly extends the &-Prolog parallelizing compiler to support several logic programming paradigms.

The implementation of the CIAO system makes use of the observation that the languages and programming styles that it implements share much at both the semantic and the implementation levels [HtCg93, HtCg94]. In that work we discussed several methodological aspects regarding the design and efficiency of a class of future logic programming systems. In particular, we proposed a novel view of parallel and concurrent logic programming systems. We argued that a large number of the actual systems and models can be described through the application of only a few basic principles. We also argued that, in fact, a system supporting several models can be implemented using a comparatively simple, common abstract machine. These principles include determinism, non-failure, independence (also referred to as stability), and task granularity. We also argued for a separation between those principles which have to do with the computation rule (i.e., to performing the least work possible) and those directly related to parallelism (i.e., to performing such work in the smallest amount of time by splitting it among several processors). Finally, and basing our discussion on the convergence of concepts that this view brought, we sketched the design of the CIAO system.

In CIAO the different source-level constructs, and even full sub-languages, are supported by compilation via *source to source* transformations into the kernel language. This allows simplifying the core of the compiler which can then implement a reduced set of analysis and transformation techniques. These techniques are based on novel semantic modeling of CLP and CC program behavior and on the exploitation of fundamental optimization principles (independence/stability and determinism), and techniques based on global analysis (program specialization and abstract executability). This approach allows reusing significant portions of modern Prolog implementation technology. The same philosophy is used in the language syntax design where the better part of the Prolog syntax and language design is used extensively. As a result CIAO subsumes ISO-Prolog (in some cases using a library) and ISO-Prolog programs run in CIAO.

Given the characteristics mentioned above, CIAO can be used quite effectively for developing applications. However, one of its fundamental objectives is to be a tool for easily experimenting with and evaluating language design issues, including program analysis and transformation methods and lower-level implementation techniques.

The main aim of this paper is to offer a general overview of the system and provide references to publications or technical reports where the actual techniques used are described. We hope that the reader will understand in this context the large number of references to CIAO-related publications. A good overview of work in the parallelization of logic programs can be found in [CC94]. Good overviews of recent progress in Constraint Logic Programming and Concurrent (Constraint) Logic Programming can be found respectively in [JM94] and [Tic95]. Most of the papers and technical reports referenced can be obtained from our Laboratory's WWW server `http://www.clip.dia.fi.upm.es/`.

## 2 Some Design Guidelines of the CIAO System

We now expand on some of the considerations outlined in the introduction, which guide the design of the CIAO system. Such guidelines include:

- *Extensive property inference capabilities, but few compulsory assertions:* The system includes a global analyzer capable of inferring many properties such as types, modes, determinacy, non-failure, freeness, independence, etc. However, in contrast to strongly typed systems such as Mercury [HSC96], none of these annotations are compulsory. In addition, the user can add annotations regarding this properties. Such annotations can optionally be 'checked" by the compiler or "trusted," i.e., used to guide the analysis (for example, when stating the interface of a module). The analyzer communicates the results of analysis in the same assertion language [BCHP96, PBH97, Gro97, BDD$^{+}$97, HtCG97].

- *Support for ISO-Prolog:* The system provides support for ISO-Prolog. However, this is not done directly but rather via a library. The basic language (when this library is not loaded) can be seen as constructed by first eliminating "unnecessary" impure features in ISO-Prolog (which are put in the library) and adding other features, but keeping the basic syntax. For example, the CIAO system allows constraints, functions, higher order syntax (e.g., `P(X)`), and other Prolog extensions, but all syntactically correct ISO-Prolog programs are also syntactically correct CIAO programs.

- *Versatile compilation options and small executables:* The system provides several different ways of producing executables. These include, in addition to the traditional modes of compilation in Prolog systems, support for the use of CIAO as a scripting language (which avoids any need to use the compiler or top level for program development), compilation to bytecode executables, compilation into C files for linking into C applications, compilation to small native executables, etc.

- *Support for Multiple Models and Paradigms:* The system supports a range of LP, CLP, and CC programming languages. It also supports several computation rules, including standard left-to-right SLD resolution and the determinate-first principle (as in the Andorra model [SCWY90, dMSC93]).

- *Support for Distributed Execution:* In the belief that many distributed applications are a good target for computational logic systems, the CIAO system includes extensive distributed execution capabilities. Such capabilities allow the transparent execution of parallel/concurrent code written for a multiprocessor in a distributed environment (of course, granularity control is an issue here). Furthermore, the system includes the notions of "team" of workers and "active module" (or active object), which conveniently encapsulate different types of functionalities desirable from a distributed system, from parallelism for achieving speedup to client-server applications (the notion of active object is provided as a smooth extension of the built-in module/object system). The distributed capabilities of CIAO are described in [CH95]. In addition, the system also offers several facilities and a library for developing WWW-based applications.[1]

- *Implementation via Compilation into a Simple Kernel Language:* This is based on the belief that medium to high performance implementations of many LP systems can be obtained in this way, with the advantages then that optimizations can be performed via source to source transformations and the low level machinery can be kept minimal. Optimizations, which include parallelization, reduction of concurrency and synchronization, reordering of goals, code simplification, specialization, etc., are performed via source to source transformation. Most analysis phases are performed at the kernel language level, so that the same analyzer can be used for several models. For example, a single analyzer framework can handle Prolog programs with delay and concurrent (constraint) programs.

- *Explicit Control in the Kernel Language:* makes it possible to perform many control-related optimizations at the source level. Such explicit control is performed via operators which include:

  – *Sequential, Parallel, and Concurrent Operators:* the presence of separate sequentiality (“`,`”), concurrency (“`&/1`”) and parallelism (“`&/2`”, “`&>/2`”, “`<&/1`”) operators allows performing optimizations such as parallelization (task creation based on dependencies), partitioning and schedule analysis (task coalescence based on dependencies), and granularity control (task coalescence based on task size considerations) as source to source transformations.

    The *parallel operators* allow indicating points where parallelism can be exploited. Their behavior is otherwise equivalent to that of the

[1] See `http://www.clip.dia.fi.upm.es/miscdocs/html_pl/html_pl.html` for details.

sequential operator. Thus, full backtracking is supported. These operators essentially assume independence among goals. Communication of bindings is therefore not guaranteed until the join. No variable locking is performed. The *concurrency operator* allows concurrent programming in the style of CC languages (also, the parallelism in such concurrent execution may be exploited if resources are available). Backtracking is limited to allow a relatively straightforward implementation. In particular, in the current version of the CIAO system no "active shared binding" is allowed to be undone via backtracking. An active shared binding is a binding to a variable that is shared among active (i.e., non-finished) processes. Variable communication (and locking) is performed.

  - *Explicit And-Fairness Operator:* based on the observation that and-fairness in concurrent systems is still expensive to implement, a fair concurrency operator ("`&&/1`") is introduced which explicitly requests the (efficient) association of computational resources (e.g., an operating system thread) to a goal. Note that this leaves open the possibility of implementing a fair source language that compiles efficiently into this and the above operators (perhaps via an analysis which can determine the program points where fairness is really needed – to ensure, for example, termination).
  - *Explicit Synchronization:* handled in the kernel language by means of "`wait/1`" and "`ask/1`" operators (the latter as in concurrent constraint programming, the former as in &-Prolog), augmented with some meta-tests on the variables (such as `ground/1` or `nonvar/1`).
  - *Explicit Placement Operator:* The "`@`" operator allows control of task placement in distributed execution. This operator can be combined with any of the parallelism and concurrency operators mentioned before. Other primitives for controlling distributed execution include primitives for dealing with teams of workers, and for using active modules or active objects.

- *Generic Abstract Machine:* a comparatively simple abstract machine directly supports the kernel language. The design of the abstract machine is based on the belief that there is much in common at the abstract machine level among many of the LP, CLP, and CC models, and thus builds strongly on the parallelism and concurrency capabilities of the PWAM/&-Prolog abstract machine [Her86, HG91] and recent work on extending its capabilities and efficiency [PGT$^+$96]. The abstract machine includes native support for *attributed variables* [Hol92, Hui90, Neu90] [Hol92] which are used in the implementation of constraint solvers (as in other systems such as Eclipse [Eur93] and SICStus 3 [Swe95]) and in supporting communication among concurrent tasks [HCC95]. While the current abstract

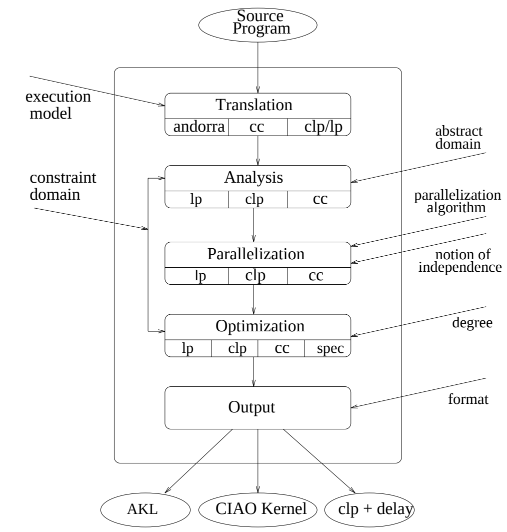


Figure 1: CIAO Compiler Structure

machine supports only ("dependent" and "independent") and-parallelism, it is expected that combination with or-parallelism will be possible by applying the techniques developed in [GSCYH91, GC92, GHPSC94].

## 3 The CIAO Compiler

The CIAO compiler provides the required support for the different programming paradigms and their optimization. As mentioned before, it is strongly based on program analysis and transformation. The compilation process can be viewed as a translation process from the input language to (kernel) CIAO. The system is able to translate the input source, automatically extracting parallelism, compiling synchronization, and optimizing the final program. Optimizations include simplifying the code to avoid run-time tests and suspensions, and specializing predicates in order to generate much simpler and efficient code in the back end.

This compilation process is depicted in Figure 1, which illustrates the inputs and outputs, as well as the compilation options, which are selected via either menus or program flags. The compilation process is structured into several steps. First, a module in a given input language is translated into the kernel language. Then, analysis is performed if required to support the rest of the compilation process. Some degree of analysis may also be performed to aid in the translation step. After analysis, the program is optionally annotated for parallel execution, simplified and specialized.

The output can then be loaded for execution on the abstract machine. As an alternative, and using the transformational approach, most of the capability of the system is also supported (with sometimes somewhat lower efficiency) on any Prolog system with delay declarations and attributed variables (e.g., SICStus Prolog Version 3 [Swe95]). In that sense, the CIAO compiler can also be viewed as a library package for Prolog systems with these capabilities.

The compiler steps and options are discussed in the following sections. As mentioned before, our aim herein is to offer a general description and provide references for publications or technical reports where the techniques used are described. An extended description of the capabilities of the compiler can be found in the User's Manual [Bue95].

### 3.1 Source Languages Supported and Transformations Performed

The compiler can deal with several languages and computation rules simultaneously and perform several translations among them. Currently, there are three languages supported: the CIAO kernel language (backwards compatible with Prolog, plus the specific CIAO primitives), languages based on the basic Andorra model, and basic CC languages. Also, for each of these languages several constraint domains can be chosen. Currently, the system supports those of Prolog, CLP(R), and CLP(Q).

The mode of the system can be changed by typing at the top level the commands `ciao(Domain), andorra(Domain)`, or `cc(Domain)`, where the variable `Domain` has to be instantiated to either `h`, `q`, or `r`, indicating the desired constraint domain, i.e. Herbrand, Q, or R, respectively. Programs read from then on will be assumed to have the characteristics associated to the new mode:

- `ciao(Domain)`: CIAO full syntax (backwards compatible with Prolog, plus the specific CIAO primitives) language; left-to-right and (encapsulated) concurrency.
- `andorra(Domain)`: Prolog language; computation rule based on the basic Andorra principle.
- `cc(Domain)`: basic CC language; concurrent computation rule.

Alternatively, the mode can be directly included in the program. This is done in the module declaration, which has one additional argument available for the specification of the mode.

Program transformations bridge the semantic gaps between the different programming paradigms supported. The methods used for translating programs based on the (Basic) Andorra model to CIAO are described in [BDGH94]. The methods used for translating CC languages are an extension of those of [DGB94, Deb93] and are described in [BH95b].

### 3.2 Analysis

The CIAO compiler includes both local and global analysis of programs. Local analysis of program clauses is usually very simple but not accurate. Nonetheless, it is sometimes partially useful in some optimizations, as in program parallelization [BGH94]. Global analysis is performed in the context of abstract interpretation [CC77, Deb92, CC92]. The underlying framework of analysis is that of PLAI [HWD92, MH90, MH92]. PLAI implements a generic (goal-dependent and goal-independent) top-down driven abstract interpreter. The whole computation is domain–independent. This allows plugging in different abstract domains, provided suitable interfacing functions are defined. PLAI also incorporates incremental analysis [HPMS95] in order to deal with large programs and is capable of analyzing full languages (in particular, full standard Prolog [BCHP96, CRV94]).

A modification of the PLAI framework capable of analyzing dynamically scheduled programs is also provided in order to support the concurrent models. Note that, thanks to the transformational approach, only two frameworks are used (one for simple, left-to-right execution and another for the case when there are dynamically scheduled goals). The compiler automatically decides the framework to be used.

The CIAO analyzer incorporates the following domains, which are briefly explained below: *Sh*, *Sh+Fr*, *ASub*, *Sh+ASub*, and *Sh+Fr+ASub*, which are used in logic programming, and *Def*, *Fr*, *Fd*, which can be used either in logic or constraint logic programming, and *LSign*, which is more specific to constraint logic programming.[2] Programs with dynamic scheduling can be analyzed with the *Sh+Fr* and *Def* domains.

3.2.1 HERBRAND: For the analysis of (classical) logic programs (over the Herbrand domain) the CIAO compiler includes a number of traditional domains proposed in the literature for capturing properties such as variable groundness, freeness, sharing, and linearity information. This includes the set sharing *Sh* [JL89, MH89], set sharing and freeness Sh+Fr [MH91], and pair sharing *ASub* [Son86] domains. Combinations of the *Sh* and *Sh+Fr* domains with *ASub* are also supported, resulting in the *Sh+ASub* and *Sh+Fr+ASub* domains. The combination is done in such a way that the original domains and operations of the analyzer over them are re-used, instead of redefining the domains for the combination [CC79, CMB+95]. Two other domains, a modified version of *Path sharing* [KS95] and *Aeqns* (abstract equations) [MSJB95] are currently being incorporated to the system. In addition the system performs type and determinacy/non-failure analysis [BJ88, GdW94, DLH97].

3.2.2 CONSTRAINT PROGRAMMING: Several domains are available, some of which have been implemented by other users of the PLAI system, notably

[2] Some of these domains have been implemented by other users of the PLAI system, notably the K. U. Leuven, Monash University, and the U. of Melbourne.

the K. U. Leuven, Monash University, and the U. of Melbourne.

The abstract domain *Def* [GHB$^+$96] determines whether program variables are definite, i.e. constrained to a unique value. In doing this it keeps track of *definite* dependencies among variables. The abstract domain *Fr* [GHB$^+$96] determines which variables act as *degrees of freedom* with respect to the satisfiability of the constraint store in which they occur. In doing this it keeps track of *possible* dependencies among variables. The definite and possible dependencies are used to perform accurate definiteness and non-freeness propagation, respectively, and are also useful in their own right to perform several program optimizations. A combined domain *Fd* which infers both definiteness and freeness is also integrated.

A preliminary version of the domain *LSign* [MS94] is also supported. This domain is aimed at inferring accurate information about possible interaction between linear arithmetic equalities and inequalities. The key idea is to abstract the actual coefficients and constants in constraints by their "sign". A preliminary implementation of this domain shows very promising accuracy, although at a cost in efficiency.

3.2.3 Dynamically Scheduled Programs: CIAO also includes a version of the PLAI framework which is capable of accurately analyzing (constraint) programs with dynamic scheduling (e.g., including delay declarations [Col82, Car87]). Being able to analyze constraint languages with dynamic scheduling also allows analyzing CC languages with angelic nondeterminism.[3] This is based on the observation that most implementations of the concurrent paradigm can be viewed as a computation which proceeds with a fixed, sequential scheduling rule but in which some goals suspend and their execution is postponed until some condition wakes them. Initial studies showed that accurate analysis in such programs is possible [MGH94], although this technique involves relatively large cost in analysis time. The analysis integrated into the CIAO compiler uses a novel method which improves on the previous one by increasing the efficiency without significant loss of accuracy [GMS95]. The approach is based on approximating the delayed atoms by a closure operator.

A direct method for analysis of CC programs has also been developed and is currently being integrated into the compiler. This method is an extension of previous work of Debray [DGB94, Deb93]. It is based on the observation that for certain properties, it is possible to extend existing analysis technology for the underlying fixed computation rule in order to deal with such programs [BH95b]. In particular, this idea has been applied using as starting point the original framework for the analysis of sequential programs. The resulting analysis can deal with programs where concurrency is governed by the Andorra model as well as standard CC models. The advantage with respect to the method above

[3]This is a kind of nondeterminism which does not give rise to an arbitrary choice when applying a search rule.

is lower analysis time, in exchange for a certain loss of accuracy.

### 3.3 Parallelization

LP, CLP, and CC offer an interesting case study for automatic parallelization. On the one hand these languages pose significant challenges to automatic parallelization, such as a symbolic nature, non-determinism, irregular computations, dynamic control flow, and (well behaved) pointers. On the other hand they offer a clear semantics on which formal program analysis and transformation techniques can be easily based. In the CIAO compiler the information inferred during the analysis phase is used for independence detection, which is the core of the LP and CLP parallelization process [BGH94, GBH96]. The compile-time parallelization module is currently aimed at uncovering goal-level, restricted (i.e., fork and join), independent and-parallelism (IAP). Independence has the very desirable properties of correct and efficient execution w.r.t. standard sequential execution of Prolog or CLP. In the context of LP, parallelization is performed based on the well-understood concepts of *strict* and *non-strict* independence [HR95], using the information provided by the abstract domains. While the notions of independence used in LP are not directly applicable to CLP, specific definitions for CLP (and constraint programming with dynamic scheduling) have been recently proposed [GHM93, Gar94] and they have been incorporated in the CIAO compiler in order to parallelize CLP and CC programs [GHM95]. Additionally, the compiler has side-effect and granularity analyzers (not depicted in Figure 1) which infer information which can yield the sequentialization of goals (even when they are independent) based on efficiency or maintenance of observable behavior.

The actual automatic parallelization of the source program is performed in CIAO during compilation of the program by the so called *annotation* algorithms. The algorithms currently implemented are: `mel`, `cdg`, `udg` [Mut91, Bue94], and `urlp` [CH94]. To our knowledge, the CIAO system is the first one to perform automatic compile-time (And-)parallelization of CLP programs [GBH96].

### 3.4 Optimization

The CIAO compiler performs several forms of code optimization by means of source to source transformations. The information obtained during the analysis phase is not only useful in automatic program parallelization, but also in this program specialization and simplification phase.

The CIAO compiler can optimize programs to different degrees, as indicated by the user. It can just simplify the program, where simplification amounts to reducing literals and predicates which are known to always succeed, fail, or lead to error. This can speed up the program at run-time, and also be useful to detect errors at compile-time. It can also specialize the program using the versions generated during analysis [PH95a]. This may involve generating different versions of a predicate for different *abstract call patterns*, thus increasing the program size whenever this allows more optimizations. In order to keep the size of the specialized program as reduced as possible, the number of versions of each predicate is minimized attaining the same results as with Winsborough's algorithm [Win92].

As well as handling sequential code, the optimization module of the CIAO compiler contains what we believe is the first automatic optimizer for languages with dynamic scheduling [PH95b]. The potential benefits of the optimization of this type of programs were already shown in [MGH94], but they can now be obtained automatically. These kinds of optimizations include simplification and elimination of suspension conditions and elimination of concurrency primitives (sequentialization).

### 3.5 Granularity Control

The compiler also performs granularity control using the techniques described in [DLH90, Tic88, LHD94, DGHL94, LH95]. The compiler estimates the granularity of parallel tasks, i.e. the work available under them, by generating expressions that are upper and lower bounds for the computation time of parallel tasks as a function of the size of task input data. These functions are used at run-time to perform granularity control by comparing cost bounds to suitable thresholds. The results so far are encouraging, and we plan to perform much further investigation in this very important area.

### 3.6 Output

The back end of the compiler takes the result of the previous program transformations and generates a number of final output formats. Normally, the result of the compiler is intended for the CIAO/&-Prolog abstract machine. Output possibilities are then byte-code ("`.ql`") files, C-files (for linking into traditional applications), stand-alone executables, and incore compilation (when the compiler is running inside the system rather than as a stand-alone application). As mentioned before, and as an alternative output, most of the capability of the system can also be handled by any Prolog which supports delay declarations and attributed variables. Alternatively, also AKL [JH91] can be used as a target, using the techniques described in [BH95a].

Finally, it is possible to obtain the results of each of the intermediate compilation phases. This allows visualizing and affecting the transformation, analysis, parallelization, and optimization processes. Because of the source to source nature of the compiler, this output is always a (possibly annotated) kernel CIAO program.

## 4 Performance Evaluation Tools

Two tools have been developed to complement the environment and help during performance debugging. VisAndOr [CGH93] is a tool for visualizing parallel executions. It supports both conjunctive and disjunctive execution graphs (or- and and-parallelism). It is currently in use by several other researchers in the field and it is distributed with other parallel systems such as Muse [AK90]. IDRA

[FCH94] is a simulator which can quite accurately compute ideal speedups from traces from a sequential execution of a parallelized program. It allows evaluating the performance of the parallel run-time system independently of the quality of the parallelization performed by the compiler, by comparing the obtained speedups with those predicted by IDRA for the given parallelized program.

## 5 Some Future Directions

We have briefly described the current status of the CIAO system. The current main objective of the system is to be an experimentation and evaluation vehicle for programming constructs and optimization and implementation techniques for the programming paradigms of LP, CLP, CC, and their combinations. The system has already shown itself useful in illustrating the power (or lack thereof) of a number of analysis and optimization techniques (see the referenced papers for details). Additionally, we are developing pilot applications with the system which should provide valuable feedback regarding its capabilities.

While the CIAO system illustrates that analysis and optimization of concurrent programs is possible, much work remains in improving the efficiency and accuracy of the analysis and in improving the performance gains obtained with the resulting optimizations.

As mentioned in Section 3.3, the automatic parallelization currently performed in the CIAO system is at the goal level. However, it is possible to parallelize at finer granularity levels, thus obtaining greater degrees of parallelism. The concept of *local independence* [MRB$^+$94, BHMR94] can be used for this purpose. Although some promising progress has been made in this direction [HCC95], it remains as future work to implement a system fully capable of efficiently exploiting this very fine grained level of parallelism.

While our work in detection of parallelism in the CIAO compiler concentrates on compile-time detection of parallelism, run-time detection also needs to be explored. Significant progress has been made in this area by models and systems such as DDAS [She92], Andorra-I, and AKL.

Some parts of the CIAO system (e.g., the PLAI analyzers, the parallelizers, the WWW libraries, the distributed programming libraries, etc.) have been distributed in the public domain and have been used experimentally by several researchers. Current versions of other parts of the system which are less developed are available for experimentation (please contact the authors; further information can be obtained from `http://www.clip.dia.fi.upm.es/`). Both kernel system and all the libraries are intended to be put gradually in the public domain.

## Acknowledgments

The development of the CIAO system is a collaborative effort of several groups and includes work developed at the U. of Arizona (S. Debray's group), New Mexico State University (G. Gupta and E. Pontelli), K. U. Leuven (M. Bruynooghe's group), and Monash and Melbourne U. (M. García de la Banda, K. Marriott, and P. Stuckey), in addition to the CLIP (Computational Logic, Implementation, and Parallelism) group at the Technical University of Madrid. Parts of this work have been funded by ESPRIT projects PRINCE, PARFORCE, and ACCLAIM, and CICYT project IPL-D.

## References

[AK90] K. A. M. Ali and R. Karlsson. The Muse Or-Parallel Prolog Model and its Performance. In *1990 North American Conference on Logic Programming*, pages 757–776. MIT Press, October 1990.

[BCHP96] F. Bueno, D. Cabeza, M. Hermenegildo, and G. Puebla. Global Analysis of Standard Prolog Programs. In *European Symposium on Programming*, number 1058 in LNCS, pages 108–124, Sweden, April 1996. Springer-Verlag.

[BDD+97] F. Bueno, P. Deransart, W. Drabent, G. Ferrand, M. Hermenegildo, J. Maluszynski, and G. Puebla. On the Role of Semantic Approximations in Validation and Diagnosis of Constraint Logic Programs. In *Proc. of the 3rd. Int'l Workshop on Automated Debugging–AADEBUG'97*, pages 155–170, Linkoping, Sweden, May 1997. U. of Linkoping Press.

[BDGH94] F. Bueno, S. K. Debray, M. García de la Banda, and M. Hermenegildo. QE-Andorra: A Quiche–Eating Implementation of the Basic Andorra Model. Technical Report CLIP13/94.0, T.U. of Madrid (UPM), September 1994.

[BGH94] F. Bueno, M. García de la Banda, and M. Hermenegildo. Effectiveness of Global Analysis in Strict Independence-Based Automatic Program Parallelization. In *International Symposium on Logic Programming*, pages 320–336. MIT Press, November 1994.

[BH95a] F. Bueno and M. Hermenegildo. An Automatic Translation Scheme from CLP to AKL. Technical Report CLIP7/95.0, Facultad de Informática, UPM, June 1995.

[BH95b] F. Bueno and M. Hermenegildo. Analysis of Concurrent Constraint Logic Programs with a Fixed Scheduling Rule. In *ICLP95 WS on Abstract Interpretation of Logic Languages*, Japan, June 1995.

[BHMR94] F. Bueno, M. Hermenegildo, U. Montanari, and F. Rossi. From Eventual to Atomic and Locally Atomic CC Programs: A Concurrent Semantics. In *Fourth International Conference on Algebraic and Logic Programming*, number 850 in LNCS, pages 114–132. Springer-Verlag, September 1994.

[BJ88] M. Bruynooghe and G. Janssens. An Instance of Abstract Interpretation Integrating Type and Mode Inference. In *Fifth International Conference and Symposium on Logic Programming*, pages 669–683, Seattle, Washington, August 1988. MIT Press.

[Bue94] F. Bueno Carrillo. *Automatic Optimisation and Parallelisation of Logic Programs through Program Transformation.* PhD thesis, Universidad Politécnica de Madrid (UPM), October 1994.

[Bue95] F. Bueno. The CIAO Multiparadigm Compiler: A User's Manual. Technical Report CLIP8/95.0, Facultad de Informática, UPM, June 1995.

[Car87] M. Carlsson. Freeze, Indexing, and Other Implementation Issues in the Wam. In *Fourth International Conference on Logic Programming*, pages 40–58. University of Melbourne, MIT Press, May 1987.

[CC77] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[CC79] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Sixth ACM Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979.

[CC92] P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*, 13(2 and 3):103–179, July 1992.

[CC94] J. Chassin and P. Codognet. Parallel Logic Programming Systems. *Computing Surveys*, 26(3):295–336, September 1994.

[CGH93] M. Carro, L. Gómez, and M. Hermenegildo. Some Paradigms for Visualizing Parallel Execution of Logic Programs. In *1993 International Conference on Logic Programming*, pages 184–201. MIT Press, June 1993.

[CH94] D. Cabeza and M. Hermenegildo. Extracting Non-strict Independent And-parallelism Using Sharing and Freeness Information. In *1994 International Static Analysis Symposium*, number 864 in LNCS, pages 297–313, Namur, Belgium, September 1994. Springer-Verlag.

[CH95] D. Cabeza and M. Hermenegildo. Distributed Concurrent Constraint Execution in the CIAO System. In *Proc. of the 1995*

*COMPULOG-NET Workshop on Parallelism and Implementation Technologies*, Utrecht, NL, September 1995. U. Utrecht / T.U. Madrid. Available from `http://www.clip.dia.fi.upm.es/`.

[CMB+95] M. Codish, A. Mulkers, M. Bruynooghe, M. García de la Banda, and M. Hermenegildo. Improving Abstract Interpretations by Combining Domains. *ACM Transactions on Programming Languages and Systems*, 17(1):28–44, January 1995. Available from `http://www.clip.dia.fi.upm.es`.

[Col82] A. Colmerauer et al. *Prolog II: Reference Manual and Theoretical Model.* Groupe D'intelligence Artificielle, Faculté Des Sciences De Luminy, Marseille, 1982.

[CRV94] B. Le Charlier, S. Rossi, and P. Van Hentenryck. An Abstract Interpretation Framework Which Accurately Handles Prolog Search–Rule and the Cut. In *International Symposium on Logic Programming*, pages 157–171. MIT Press, November 1994.

[Deb92] S. Debray, editor. *Journal of Logic Programming, Special Issue: Abstract Interpretation*, volume 13(1–2). North-Holland, July 1992.

[Deb93] S. K. Debray. Implementing logic programming systems: The quiche-eating approach. In *ICLP '93 Workshop on Practical Implementations and Systems Experience in Logic Programming*, Budapest, Hungary, June 1993.

[DGB94] S. Debray, D. Gudeman, and P. Bigot. Detection and Optimization of Suspension-free Logic Programs. In *1994 International Symposium on Logic Programming*, pages 487–501. MIT Press, November 1994.

[DGHL94] S. K. Debray, P. López García, M. Hermenegildo, and N.-W. Lin. Estimating the Computational Cost of Logic Programs. In *Static Analysis Symposium, SAS'94*, number 864 in LNCS, pages 255–265, Namur, Belgium, September 1994. Springer-Verlag.

[DLH90] S. K. Debray, N.-W. Lin, and M. Hermenegildo. Task Granularity Analysis in Logic Programs. In *Proc. of the 1990 ACM Conf. on Programming Language Design and Implementation*, pages 174–188. ACM Press, June 1990.

[DLH97] S. K. Debray, P. López García, and M. Hermenegildo. Non-Failure Analysis for Logic Programs. In *1997 International Conference on Logic Programming*, pages 48–62, Leuven, Belgium, June 1997. MIT Press, Cambridge, MA.

[dMSC93] Vítor Manuel de Morais Santos Costa. *Compile-Time Analysis for the Parallel Execution of Logic Programs in Andorra-I.* PhD thesis, University of Bristol, August 1993.

[Eur93] European Computer Research Center. *Eclipse User's Guide*, 1993.

[FCH94] M. Fernández, M. Carro, and M. Hermenegildo. IDRA (IDeal Resource Allocation): A Tool for Computing Ideal Speedups. In *ICLP WS on Parallel and Data Parallel Execution of Logic Programs*, June 1994.

[Gar94] M. García de la Banda. *Independence, Global Analysis, and Parallelism in Dynamically Scheduled Constraint Logic Programming.* PhD thesis, Universidad Politécnica de Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, September 1994.

[GBH96] M. García de la Banda, F. Bueno, and M. Hermenegildo. Towards Independent And-Parallelism in CLP. In *Programming Languages: Implementation, Logics, and Programs*, number 1140 in LNCS, pages 77–91, Aachen, Germany, September 1996. Springer-Verlag.

[GC92] G. Gupta and V. Santos Costa. And-Or Parallelism in Full Prolog based on Paged Binding Array. In *Parallel Architectures and Lenguages Europe '92*. Springer Verlag, June 1992.

[GdW94] J.P. Gallagher and D.A. de Waal. Fast and precise regular approximations of logic programs. In Pascal Van Hentenryck, editor, *Proceedings of the Eleventh International Conference on Logic Programming*, pages 599–613. The MIT Press, 1994.

[GHB+96] M. García de la Banda, M. Hermenegildo, M. Bruynooghe, V. Dumortier, G. Janssens, and W. Simoens. Global Analysis of Constraint Logic Programs. *ACM Transactions on Programming Languages and Systems*, 18(5):564–615, 1996.

[GHM93] M. García de la Banda, M. Hermenegildo, and K. Marriott. Independence in Constraint Logic Programs. In *1993 International Logic Programming Symposium*, pages 130–146. MIT Press, Cambridge, MA, October 1993.

[GHM95] M. García de la Banda, M. Hermenegildo, and K. Marriott. Independence and Search Space Preservation in Dynamically Scheduled Constraint Logic Languages. Technical Report CLIP10/95.0, Facultad de Informática, UPM, February 1995.

[GHPSC94] G. Gupta, M. Hermenegildo, E. Pontelli, and V. Santos-Costa. ACE: And/Or-parallel Copying-based Execution of Logic Programs. In *International Conference on Logic Programming*, pages 93–110. MIT Press, June 1994.

[GMS95] M. García de la Banda, K. Marriott, and P. Stuckey. Efficient Analysis of Constraint Logic Programs with Dynamic Scheduling. In *1995 International Logic Programming Symposium*, pages 417–431, Portland, Oregon, December 1995. MIT Press, Cambridge, MA.

[Gro97] The CLIP Group. Program Assertions. The CIAO System Documentation Series – TR CLIP4/97.1, Facultad de Informática, UPM, August 1997.

[GSCYH91] 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.

[HCC95] M. Hermenegildo, D. Cabeza, and M. Carro. Using Attributed Variables in the Implementation of Concurrent and Parallel Logic Programming Systems. In *Proc. of the Twelfth International Conference on Logic Programming*, pages 631–645. MIT Press, June 1995.

[Her86] M. Hermenegildo. An Abstract Machine for Restricted AND-parallel Execution of Logic Programs. In *Third International Conference on Logic Programming*, number 225 in Lecture Notes in Computer Science, pages 25–40. Imperial College, Springer-Verlag, July 1986.

[HG91] M. Hermenegildo and K. Greene. The &-Prolog System: Exploiting Independent And-Parallelism. *New Generation Computing*, 9(3,4):233–257, 1991.

[Hol92] C. Holzbaur. Metastructures vs. Attributed Variables in the Context of Extensible Unification. In *1992 International Symposium on Programming Language Implementation and Logic Programming*, pages 260–268. LNCS631, Springer Verlag, August 1992.

[HPMS95] M. Hermenegildo, G. Puebla, K. Marriott, and P. Stuckey. Incremental Analysis of Logic Programs. In *International Conference on Logic Programming*, pages 797–811. MIT Press, June 1995.

[HR95] M. Hermenegildo and F. Rossi. Strict and Non-Strict Independent And-Parallelism in Logic Programs: Correctness, Efficiency, and

Compile-Time Conditions. *Journal of Logic Programming*, 22(1):1–45, 1995.

[HSC96] F. Henderson, Z. Somogyi, and T. Conway. Determinism analysis in the mercury compiler. In *Proc. Australian Computer Science Conference*, Melbourne, Australia, January 1996.

[HtCg93] M. Hermenegildo and the CLIP group. Towards CIAO-Prolog – A Parallel Concurrent Constraint System. In *Proc. of the Compulog Net Area Workshop on Parallelism and Implementation Technologies*. FIM/UPM, Madrid, Spain, June 1993.

[HtCg94] M. Hermenegildo and the CLIP group. Some Methodological Issues in the Design of CIAO - A Generic, Parallel, Concurrent Constraint System. In *Principles and Practice of Constraint Programming*, number 874 in LNCS, pages 123–133. Springer-Verlag, May 1994.

[HtCG97] M. Hermenegildo and the CLIP Group. Programming with Global Analysis. In *Proceedings of ILPS'97*. MIT Press, October 1997. (abstract of invited talk).

[Hui90] S. Le Huitouze. A New Data Structure for Implementing Extensions to Prolog. In P. Deransart and J. Małuszyński, editors, *Proceedings of Programming Language Implementation and Logic Programming*, number 456 in Lecture Notes in Computer Science, pages 136–150. Springer, August 1990.

[HWD92] M. Hermenegildo, R. Warren, and S. K. Debray. Global Flow Analysis as a Practical Compilation Tool. *Journal of Logic Programming*, 13(4):349–367, August 1992.

[JH91] S. Janson and S. Haridi. Programming Paradigms of the Andorra Kernel Language. In *1991 International Logic Programming Symposium*, pages 167–183. MIT Press, 1991.

[JL89] D. Jacobs and A. Langen. Accurate and Efficient Approximation of Variable Aliasing in Logic Programs. In *1989 North American Conference on Logic Programming*. MIT Press, October 1989.

[JM94] J. Jaffar and M.J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[KS95] A. King and P. Soper. Depth-k Sharing and Freeness. In *International Conference on Logic Programming*. MIT Press, June 1995.

[LH95] P. López García and M. Hermenegildo. Efficient Term Size Computation for Granularity Control. In *International Conference on Logic Programming*, pages 647–661. The MIT Press, June 1995.

[LHD94] P. López García, M. Hermenegildo, and S. K. Debray. Towards Granularity Based Control of Parallelism in Logic Programs. In Hoon Hong, editor, *Proc. of First International Symposium on Parallel Symbolic Computation, PASCO'94*, pages 133–144. World Scientific, September 1994.

[MGH94] K. Marriott, M. García de la Banda, and M. Hermenegildo. Analyzing Logic Programs with Dynamic Scheduling. In *20th. Annual ACM Conf. on Principles of Programming Languages*, pages 240–254. ACM, January 1994.

[MH89] K. Muthukumar and M. Hermenegildo. Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation. In *1989 North American Conference on Logic Programming*, pages 166–189. MIT Press, October 1989.

[MH90] K. Muthukumar and M. Hermenegildo. Deriving A Fixpoint Computation Algorithm for Top-down Abstract Interpretation of Logic Programs. Technical Report ACT-DC-153-90, Microelectronics and Computer Technology Corporation (MCC), Austin, TX 78759, April 1990.

[MH91] K. Muthukumar and M. Hermenegildo. Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation. In *1991 International Conference on Logic Programming*, pages 49–63. MIT Press, June 1991.

[MH92] K. Muthukumar and M. Hermenegildo. Compile-time Derivation of Variable Dependency Using Abstract Interpretation. *Journal of Logic Programming*, 13(2/3):315–347, July 1992. Originally published as Technical Report FIM 59.1/IA/90, Computer Science Dept, Universidad Politecnica de Madrid, Spain, August 1990.

[MRB+94] U. Montanari, F. Rossi, F. Bueno, M. García de la Banda, and M. Hermenegildo. Towards a Concurrent Semantics based Analysis of CC and CLP. In *Principles and Practice of Constraint Programming*, number 874 in LNCS, pages 151–161. Springer-Verlag, May 1994.

[MS94] K. Marriott and P. Stuckey. Approximating Interaction Between Linear Arithmetic Constraints. In *1994 International Symposium on Logic Programming*, pages 571–585. MIT Press, 1994.

[MSJB95] A. Mulkers, W. Simoens, G. Janssens, and M. Bruynooghe. On the Practicality of Abstract Equation Systems. In *International Conference on Logic Programming*. MIT Press, June 1995.

[Mut91] Kalyan Muthukumar. *Compile-time Algorithms for Efficient Parallel Implementation of Logic Programs.* PhD thesis, University of Texas at Austin, August 1991.

[Neu90] U. Neumerkel. Extensible Unification by Metastructures. In *Proceeding of the META'90 workshop*, 1990.

[PBH97] G. Puebla, F. Bueno, and M. Hermenegildo. An Assertion Language for Debugging of Constraint Logic Programs. In *Proceedings of the ILPS'97 Workshop on Tools and Environments for (Constraint) Logic Programming*, October 1997. Available from `ftp://clip.dia.fi.upm.es/pub/papers/assert_lang_tr_discipldeliv.ps.gz`.

[PGT+96] E. Pontelli, G. Gupta, D. Tang, M. Carro, and M. Hermenegildo. Improving the Efficiency of Nondeterministic And–parallel Systems. *The Computer Languages Journal*, 22(2/3):115–142, July 1996.

[PH95a] G. Puebla and M. Hermenegildo. Implementation of Multiple Specialization in Logic Programs. In *Proc. ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation*, pages 77–87. ACM Press, June 1995.

[PH95b] G. Puebla and M. Hermenegildo. Specialization and Optimization of Constraint Programs with Dynamic Scheduling. Technical Report CLIP12/95.0, Facultad de Informática, UPM, September 1995. Presented at the 1995 COMPULOG Meeting on Program Development.

[SCWY90] 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.

[She92] 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.*, pages 717–731. MIT Press, 1992. To appear in JLP special issue.

[Son86] H. Sondergaard. An application of abstract interpretation of logic programs: occur check reduction. In *European Symposium on Programming, LNCS 123*, pages 327–338. Springer-Verlag, 1986.

[Swe95] Swedish Institute of Computer Science, P.O. Box 1263, S-16313 Spanga, Sweden. *Sicstus Prolog V3.0 User's Manual*, 1995.

[Tic88] E. Tick. Compile-Time Granularity Analysis of Parallel Logic Programming Languages. In *Int. Conf. on FGCS*. Tokyo, November 1988.

[Tic95] E. Tick. The Deevolution of Concurrent Logic Programming Languages. *The Journal of Logic Programming*, 23(1–3):89–125, 1995.

[Win92] W. Winsborough. Multiple Specialization using Minimal-Function Graph Semantics. *Journal of Logic Programming*, 13(2 and 3):259–290, July 1992.