Distributing Constraints on Workstation Clusters by using a Structure-Oriented Programming Model

Lothar Hotz, Michael Trowe [a] *

[a]Universität Hamburg
Labor für Künstliche Intelligenz
Fachbereich Informatik
Vogt-Kölln-Str.30, D-22527 Hamburg, Germany
e-mail: {hotz, trowe}@kogs.informatik.uni-hamburg.de

In this paper we describe an extension to COMMON LISP which allows the definition of parallel programms at a highly abstract level. This is realized by introducing a structure-oriented programming model. The use of the programming model is demonstrated with an implementation of constraints on a network of workstations.

1. Motivation

One of the big problems of Artificial Intelligence (AI) is getting its applications to deliver their answers in time. Parallel computation is one way to solve this problem. But though there are many parallel implementations of basic AI techniques, there are very few AI applications which use them. This is true due to two reasons:

- Most of these implementations depend on special parallel hardware (e.g. [3], [1]). This hardware is expensive and not widely available. Furthermore, the specification of many applications excludes the use of special hardware (e.g. personal assistant).

- Most of them are written in special parallel programming languages unknown to the application programmer and lacking features important to develop a complete application ([9]). Hence their integration into such an application is difficult.

Our goal is to simplify the parallel implementation of standard AI techniques. We think this means using standard hardware and extending a language widely used for AI programming in a way that hides any kind of explicit parallel programming from the application programmer. So we extended COMMON LISP with two levels of features for parallel programming on workstation clusters. The upper level is intended for easy use by the AI programmer inexperienced with parallel programming, while the lower level is intended for implementation of the upper level. We tested our approach implementing parallel constraint filtering.

*This research has been supported by the Bundesminister für Bildung, Wissenschaft, Forschung und Technologie (BMBF) under the grant 01 IN 509 D 0, INDIA - Intelligente Diagnose in der Anwendung

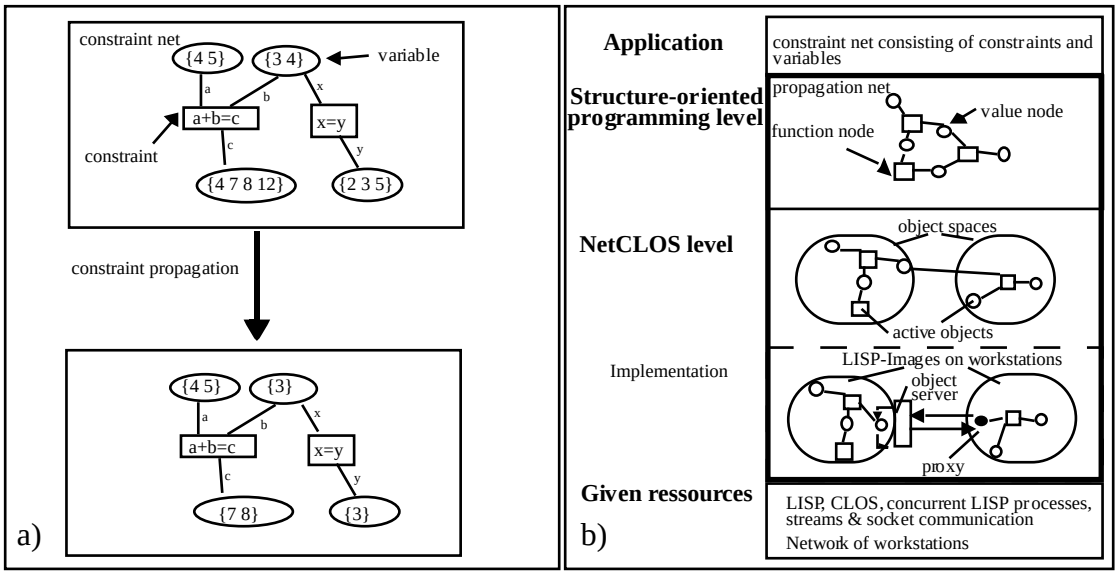

Figure 1. a) Example of constraint propagation. b) Levels of used abstractions.

2. Parallel programming using abstract datatypes — the programming model

Our programming model consists of two levels (Figure 1b)). The top level is the structure-oriented programming level. Datatypes representing complex structures and operations on them are introduced on this level. These may e.g. be an arbritray net, a tree or graph structure. These abstractions are designed and selected to support the development of AI methods. In our example the structure relaxation-net is used to implement constraint nets (section 3).

The second level, called NetCLOS[2], is used to implement the structures of the first level. It extends COMMON LISP with features for parallel and distributed object-oriented programming. The parallel and the distribution aspect of the implementation of a structure can be described independently. The parallel programming is done with active objects. These have their own processes and communicate via synchronous and asynchronous message passing (for a similar model see [8]). To distribute these objects over the workstation cluster, objects can be created on every workstation and moved from one workstation to the other. A runtime environment enables distributed garbage collection and transparent remote message passing.

This way we can divide the implementation of the structure types into two steps: A machine-independent description of the potential parallelism using active objects; and a description of the mapping of these objects to the workstation cluster.

[2]NetCLOS as an extension to ALLEGRO COMMON LISP is implemented and can be received from the authors. NetCLOS is implemented using the metaobject protocol of CLOS (see [6] and [4]).

3. An example structure — the relaxation net

Relaxation net is an abstraction implementing parallel discrete relaxation (see [3] for a similar approach). It consists mainly of

- a class of objects (*value nodes*) acting as shared stores. Accesses to these stores are automatically synchronized, i.e. this is done by the next lower, NetCLOS level. These objects can be used to implement the variables of the constraint net.
- a class of objects (*function nodes*) which, when activated, compute a function of the content of a set of stores. These objects can be used to implement the constraints.
- A structure class which organizes stores and functional objects into a network and provides for iterated activation and parallel execution of the functional objects (i.e. a relaxation operation). This *relaxation net* can be used to implement a constraint net.

To use parallel relaxation, the application programmer writes subclasses to the classes above, redefining some methods. There is no need for any explicit parallel programming. The relaxation net is then partitioned and distributed over the workstation cluster automatically, but the programmer can replace the default load-balancing strategy with an application-specific one to optimize performance.

4. Implementing parallel constraint filtering

Constraints are used to describe conditions on variables of a problem description (Figure 1a)). They determine which combinations of variable values are admissible. Constraints can be given by finite sets of tuples (like in CONSAT [2]) but also by function definitions, which are evaluated when constraints are processed. By variables used in multiple constraints a net is constructed, where the size depends on the problem size. Solving a constraint net means finding a value for each variable so that no constraint is violated.

To reduce the effort to find a solution, the method of constraint filtering is often used ([7]). This process consists of the repeated application of a filtering operation, removing all locally inconsistent values from the domain of possible values of the variables. In CONSAT a variation of this filtering scheme is also used to compute global solutions.

We used the structure type relaxation net to implement parallel constraint filtering, implementing variables as value nodes, constraints as function nodes and constraint nets as relaxation nets. We tested this implementation on our local workstation cluster and a constraint net for the 3-dimensional interpretation of line drawings [7]. First experiments showed that there is slight overhead related to the abstractions used, but much related to distribution (i.e. computing a partition) and communication (i.e. moving objects around). In the near future experiments with constraints solving a simulation task for analog electrical circuits will be carried out.

5. Conclusions

Our work presents the first approach for high-level parallel programming on a workstation cluster in COMMON LISP. Using structure types made it easy to parallelize

important AI techniques on a workstationcluster. Extending Allegro COMMON LISP enables us to integrate them into complex Applications. At the moment we use them to parallelize central parts of a diagnosis application for fork list trucks.

REFERENCES

1. M. Dixon, J. de Kleer. Massively Parallel Assumption-based Truth Maintenance. *Proceedings of the AAAI 88*, 199–204, 1988.
2. H. W. Guesgen. CONSAT: A System for Constraint Satisfaction. *Notes in Artificial Intelligence*, 1989.
3. K. Ho. *High-Level Abstractions for Symbolic Parallel Programming.* PhD thesis, University of California at Berkeley, 1994.
4. L. Hotz and G. Kamp. Programming the Connection Machine by using the Metaobject Protocol. In *Parallel Computing, Trends and Applications*, North Holland, 1994. Elsevier Science Publishers.
5. L. Hotz. An Object-Oriented Approach for Programming the Connection Machine. In H. Kitano, editor, *Second International Workshop on Parallel Processing for Artificial Intelligence, PPAI'93.* Elsevier Science Publishers, 1993.
6. G. Kiczales, D. G. Bobrow, and J. des Rivieres. *The Art of the Metaobject Protocol.* MIT Press, Cambridge, MA, 1991.
7. D. L. Waltz. *Generating semantic descriptions from drawings of scenes with shadows.* Technical Report AI-TR-271, MIT Laboratory for Computer Science, Cambridge, MA, 1972.
8. A. Yonezawa, J. Briot, and E. Shibayama. Object-Oriented Concurrent Programming in ABCL/1. *SIGPLAN Notices*, 21(11):258–268, 1986.
9. C. K. Yuen. *Parallel Lisp Systems.* Chapman & Hall, 1993.