Österreichische Artificial Intelligence-Tagung

Wien, September 1985

Herausgegeben von Harald Trost und Johannes Retti

Springer-Verlag Berlin Heidelberg New York Tokyo

Herausgeber

Harald Trost
Universität Wien
Institut für Medizinische Kybernetik und Artificial Intelligence
Freyung 6, A-1010 Wien, Österreich

Johannes Retti
Siemens AG Österreich
Göllnergasse 15, A-1030 Wien, Österreich

Tagungsleitung: Johannes Retti

Programmkomitee:

Harald Trost	Universität Wien
Wolfgang Bibel	TU München
Bruno Buchberger	Universität Linz
Ernst Buchberger	Universität Wien
Werner Horn	Universität Wien
Hermann Kaindl	Siemens AG Wien
Peter Raulefs	Universität Kaiserslautern
Ingeborg Steinacker	VOEST-Alpine AG Linz
Robert Trappl	Universität Wien
Wolfgang Wahlster	Universität Saarbrücken
Helmar Weseslindtner	TU Wien

Diese Tagung wurde von der Siemens AG Österreich sowie von der Österreichischen Studiengesellschaft für Kybernetik unterstützt.

ISBN 3-540-15695-X Springer-Verlag Berlin Heidelberg New York Tokyo
ISBN 0-387-15695-X Springer-Verlag New York Heidelberg Berlin Tokyo

Druck und Bindearbeiten: Weihert-Druck GmbH, Darmstadt
2145/3140-543210

The L-Language for the Parallel L-Machine
(A Parallel Architecture for AI Applications)

P. Hintenaus, B. Buchberger

Institut für Mathematik
Johannes-Kepler-Universität
A4040 Linz (Austria)

Abstract

The L-language, a language for programming parallel computer architectures especially suited for symbolic computation, is presented. The main goal of the language design is the explicit description of the interconnection structure of the system. Our aproach allows even recursive descriptions of the interconnection topology. The interconnection structures defined by L-programs can be realized on the parallel L-machine developed in the CAMP-LINZ working group.

Introduction

In recent years an increasing interest concentrates on possible parallel machine architectures for applications in artificial intelligence, in particular automated theorem proving, parallel evaluation of logic programs, symbolic computation and parallel organization of large knowledge bases. Parallel architectures for applications in these areas must meet objectives that are essentially different from the design objectives prevalent in numerical parallel computation. In /BiBu 84, Section 2/ a detailed analysis is given that derives the main design objectives for parallel architectures suitable for applications in artificial intelligence: It is argued that a suitable architecture should have a cellular homogeneous structure with asynchronous cooperation of many tightly coupled processor modules and a means for flexible interconnection topologies and synchronization strategies. The single modules and the parallel architecture as a whole should be universal machines.

The L-network project pursued at the working group CAMP of the University of Linz aims at the realization of a parallel architecture that meets the design objectives outlined above. The starting point of the project was the design and implementation of an "L-module", a flexible building block for the formation of parallel architectures ("L-networks") of arbitrary but fixed topology and arbitrary size (see /Bu 78/ and /Bu, Fe 78/). In the present state of the project a parallel architecture (the "L-machine") has been implemented that allows the creation of L-networks of arbitrary topology at compile time and even at execution time (see /Bu 83, 84/: see also /Bi, Bu 84/ and /As 85/ for two typical examples of applying the hardware components developed in the L-project for the realization of parallel inference machines and parallel functional evaluation.)

In this paper we give details of the syntactical realization and implementation of the L-language, a high level language that allows to program the L-machine in a natural and easy way and to fully exploit the features offered by the underlying parallel hardware. The essential two features of the L-language are:

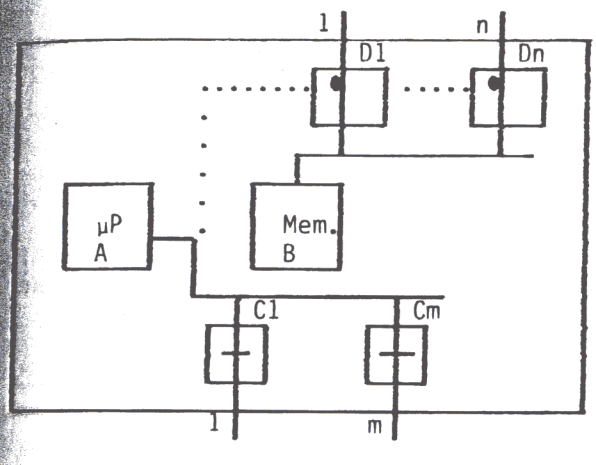
- In addition to the constructs of ordinary high level languages it incorporates the special instructions available on the L-modules (namely the "sensor bit" instructions and the "open/close" instructions for opening and closing certain interconnection buses between L-modules).

- It also allows to define L-network structures i.e. the topology of the interconnection between the L-modules. In particular, topologies with variable size parameters can be defined in this language (by recursion and iteration over the size parameters!) The potential of recursive and iterative definition of parallel network structures with variable size parameters is a feature that has not yet been included in other languages and, hence, is the major original contribution of the L-language.

The L-language has been introduced in /Bu 84/. The present paper reports on an ongoing implementation of the L-language, which forms the core of a diploma thesis of the first author under the supervision of the second author. In order to make the paper as self-contained as possible we start with a short review of the basic concepts of the L-project.

A Short Review of the L-Project

An L-Module is a module of the following structure:



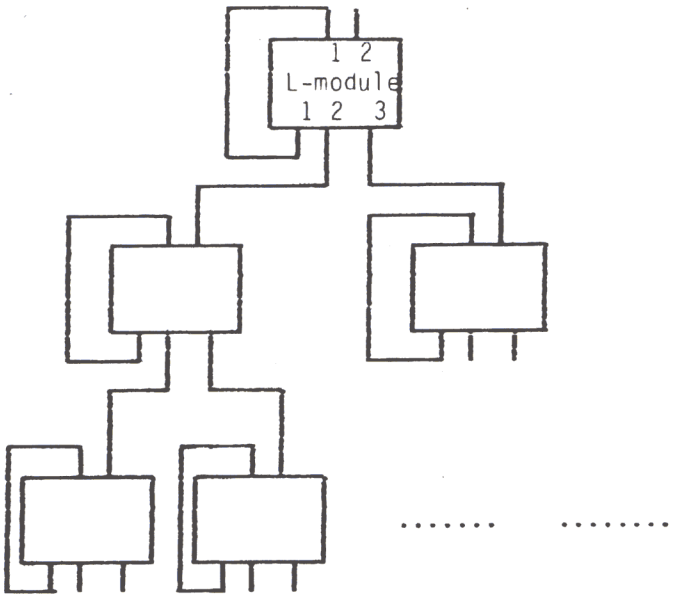
A: a microprocessor + private memory + some additional special circuitry.
B: a "shared" memory + some additional special circuitry.
C1,..,Cm: bus switches with an additional "open/close" facility. (The corresponding m bus branches are called "processor paths".)
D1,..,Dn: bus switches with a "sensor bit" ●. (The corresponding n bus branches are called "memory paths".)

In addition to the normal instruction set of a microprocessor, the component A can execute the following eight types of special instructions:

"open j", "close j" (for opening and closing switch Cj)
"set, (reset, load) local sensor j"
"set (reset, read) non-local sensor j".

The meaning of these instructions and the operation of the L-module are explained in detail in /Bu 83/. Arbitrarily many L-modules can be combined to form "L-networks" of arbitrary (but fixed) regular (or irregular) structure, by connecting a memory path of one module to a processor path of another.

As an example of a typical computation in an L-network we recall a solution of the tautology problem for boolean expressions on an L-network of binary tree topology (for details see /Bu, Fe 8/):



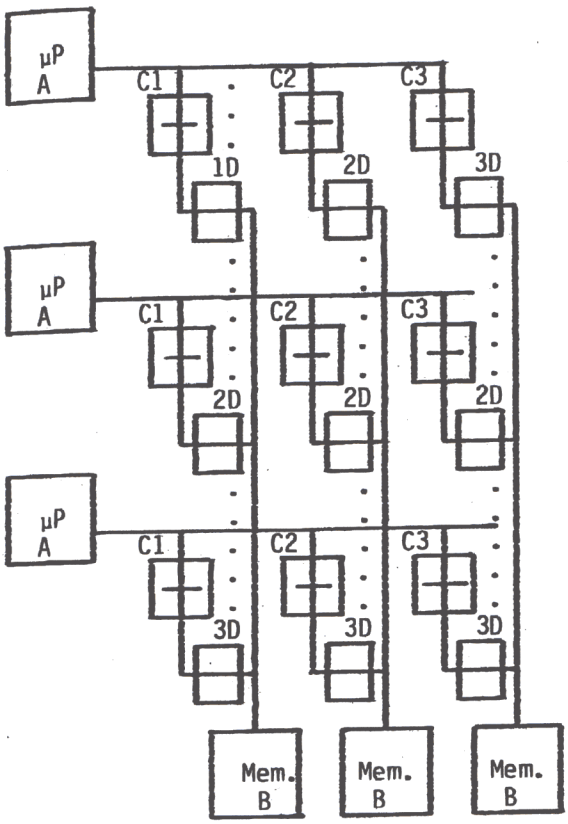
In each processor the following L-program has to be loaded and executed:

```
WHILE NOT SMə{2} DO END:
IF nə{1} = 0
THEN yə{1} := eval(tə{1})
ELSE
  tə{2} := subst(tə{1}, nə{1}, TRUE): nə{2} := nə{1} - 1:
  tə{3} := subst(tə{1}, nə{1}, FALSE): nə{3} := nə{1} - 1:
  SPə{2} := TRUE: SPə{3} := TRUE:
  WHILE SPə{2} OR SPə{3} DO END:
  yə{1} := yə{2} AND yə{3}
END
SMə{2} := FALSE
```

A reference to a variable followed by a ə and a number enclosed in braces means that this variable is located in a shared memory and is accessible via the processor path adressed by this number. SPə{i} and SMə{i} refer to the sensor bit that is located on the i'th processor path and i'th memory path respectively.

eval(t) is a procedure returning the truth value of the variable free boolean expression t. subst(t,n,b) returns the expression that results when the n'th variable of the expression t is replaced by the boolean value b.

n L-modules with n processor paths and n memory paths can be used to realize the "full graph" interconnection topology. In principle, this topology could be used to embed every other topology in a flexible and dynamical way. However, this arrangement would need n**2 switches of type C and D and n**2 connection buses between the switches. By a geometrical transformation that does not change the logical and physical properties of the device this arrangement can be replaced by the following crossbar configuration that does not need any interconnection buses (the details of this transformation are explained in /Bu 83/). A prototype of this crossbar arrangement with n=8 is in operation since November 1984. Note that exactly the same components A, B, C, D can be used for realizing special L-networks and the crossbar arrangement.



The dotted lines are the connections of the processors to the sensor bits in the components D, which the processors can access.

A Rough Sketch of the L-Language

An "L-program" written in the "L-language" /Bu 84/ consists of the description of the programs residing in the L-modules of an L-network and the description of the topology of the L-network. Significant extensions to ordinary programming languages are necessary in order to make such descriptions possible.

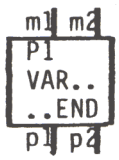
The nucleus of the implementation of the L-language described here is a small subset of MODULA-2 /Wi 82/. Programs written in the nucleus language are meant to reside in the private memory of a component A and are executed under the control of the processor in the component. In the following we call these programs PROCESSes. Variables declared local to these PROCESSes are placed in the private memories. PROCESSes can be combined to L-programs that describe the topology of an L-network and the PROCESSes residing in the L-modules of the L-network.

In order to place a variable in all shared memories it has to be declared at the outermost part of an L-program. A reference to a shared variable is made by giving the name of the variable followed (in braces) by the names of all those paths that should be used for this reference. The actual open and close instructions are generated by the compiler. The sensor bits are referenced by using the identifiers SP and SM for access via a processor path (called PPATH) or a memory path (called MPATH). We start with an example of an elementary PROCESS.

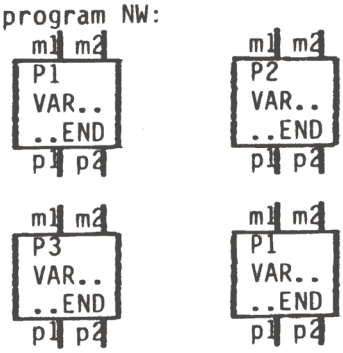
```
PROCESS P1 (p1, p2: PPATH: m1, m2: MPATH):
VAR p, q:
```

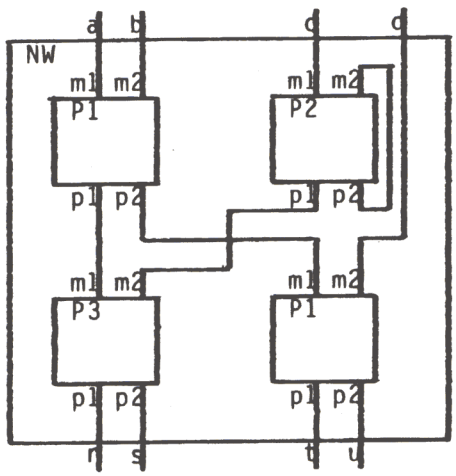
```
BEGIN
   ....
   some program text using, for example,
   v1∂{p1} := v1∂{p2}          IF SM∂{m1} THEN ....:
END:
```

This part of an L-program describes a PROCESS with two processor paths named p1 and p2 and two memory paths m1 and m2. This PROCESS will be assigned to an L-Module in a later part of the L-program. The component A of this L-module will contain the program VAR..BEGIN.....END in its private memory. The same information may also be expressed in the following graphical notation of the L-language:



It should then be immediately clear what is meant by the following graphical notation of an L-program NW:





Here, four L-modules are combined in order to form a more complicated L-network having three programs P1, P2, P3 stored in the private memories of four L-modules and realizing a particular interconnection schema between the L-modules. Some of the paths provide interconnections to the "outside" of the L-network.

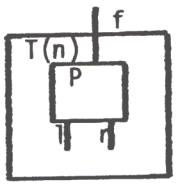
NW with processor paths r,s,t,u and memory paths a,b,c,d, again, can be used as a building block in hierarchically more complicated L-networks. The L-network NW in syntactically more conventional "linear" notation reads:

```
PROGRAM example:
```

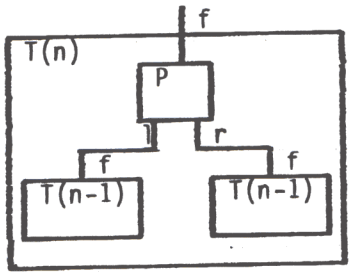
```
    declaration of shared variables (see below)

PROCESS P1(p1,p2:PPATH; m1,m2:MPATH): ....
PROCESS P2(p1,p2:PPATH; m1,m2:MPATH): ....
PROCESS P3(p1,p2:PPATH; m1,m2:MPATH): ....

NETWORK

SUBNET NW(VAR a,b,c,d: MPATH: VAR r,s,t,u: PPATH):
VAR   p11,   p12,   p21,   p22:   PPATH;   m22,   m31,   m32,   m41:   MPATH;
BEGIN
   P1(p11,p12,a,b);  P2(p21,p22,c,m22);
   P3(r,s,m31,m32);  P1(t,u,m41,d);
   CONNECT p11 TO m31; CONNECT p12 TO m41;
   CONNECT p12 TO m32; CONNECT p22 TO m22
END;

BEGIN
    actual network invocation (see below).
END.
```

A PROCESS is assigned to an L-module by giving its name in the NETWORK part of an L-program (at invocation time according to the specification in the SUBNETs). The names in the parameter list following the name of the process are used for referencing the paths of the special PROCESS-processor pair. SUBNETs serve as a collection of L-modules which will be used as building blocks for more complicated L-networks. SUBNETs can be used recursively in order to create networks of a regular structure. For example, the tree of the previous example can be defined as follows (assume we have a PROCESS P(f:MPATH:l,r:PPATH):):

```
SUBNET Tree (VAR f: MPATH; n:INTEGER);
VAR r,l:PPATH; s1,s2:MPATH;
BEGIN
  IF n=1 THEN
      P(f,r,l)
```




```
  ELSE
      P(f,r,l);
      Tree(s1,n-1);
      Tree(s2,n-1);
      CONNECT r TO s1;
      CONNECT l TO s2;
```




```
  END
END;
```

Design Objectives, Restrictions and Design Decisions for a Concrete Implementation of the L-Language

The L-language is a very general language concept that provides the natural environment for describing parallel algorithms on arbitrary L-networks and it does not depend on a particular syntactical notation. Our goal when implementing one special version of the L-language is to create a simple tool for programming the pilot reali-

zation of the L-machine described in /Bu 83, 84/. Because of its nice syntactical structure we chose MODULA-2 /Wi 82/ as the major guideline for the design of the grammar for the nucleus language. To keep the compiler simple the grammar is LL-1. Of corse we are interested in the possibility to use our compiler on prototypes with different microprocessors. Thus, we decided to generate intermediate code which will be interpreted on the actual machine. Since the present version of the compiler should only serve as an experimental tool we only implement INTEGER, BOOLEAN and ARRAYs of INTEGER and BOOLEAN as data types. However, for describing interconnection environments between L-modules that are changing before and during computation, SETs and ARRAYs of MPATHs and PPATHs are realized in addition. Our major emphasis lies on the NETWORK part in order to be able to experiment with various arrangements of processors. This feature of the language is the one that goes far beyond the possibilities of ordinary programming languages.

Our prototype compiler is developed in MODULA-2 on a Vicki microcomuter operating MSDOS V2,11. The intermediate code we generate is for a stack machine, augmented with instructions for operating the sensorbits and the switches to the shared memories. The adressing of the sensor bits and switches is done using translation tables similar to those the NS32000 architecture /NSC 83/ uses for reference to external objects, i.e. the physical addresses are stored in an array whose indices are the logical addresses.

The whole compiler operates on the host computer. The translation tables are constructed in the last pass of the L-compiler, which executes the network invocation. The resulting images are downloaded onto the physical processors of the prototype machine where the processes described in the program are executed.

Syntax of the Concrete Realization of the L-Language

An L-program has the following structure:

```
PROGRAM program name :

  CONST constant declarations
  SHARED VAR  declaration of shared variables
  Process declarations

NETWORK

  VAR declaration of variables used in the network invocation
  Subnet declarations

BEGIN
  network invocation
END.
```

A constant declaration has the following form:

```
identifier = constant expression ;
```

where constant expression is a well formed infix expression yielding a constant value.

The declaration of the shared variables has the syntactical form of a variable declaration (see below). However, only variables of the types INTEGER, BOOLEAN and ARRAY OF INTEGER or BOOLEAN may occur.

The syntactical form of variable declarations is: Variables are declared by giving their names followed by a colon and the type of the variables. Simple types are INTEGER, BOOLEAN, MPATH, PPATH. The other possible types are ARRAYs of all simple types and SETs OF PATHs. However the following restrictions must be respected:

```
-- SETs OF PATHs may only be used inside processes.
-- PATHs and ARRAYs of PATHs are allowed in the network part and in the
   parameter list of a PROCESS only.
```

Process declarations have the following form:

```
PROCESS process name ( parameter list ) :

  CONST constant declarations
  VAR variable declarations
  procedure declarations

BEGIN
  process body
END;
```

The parameter list contains the variables that will assume a value in the NETWORK invocation of the L-program. Constants and variables declared inside a PROCESS are only known inside this PROCESS. These variables will reside in the private memory.

Procedure declarations are like in MODULA-2 with the exception that nested procedure declarations are prohibited. The following control structures are available:

```
IF..THEN..ELSIF..THEN.....ELSE..END
WHILE..DO..END
REPEAT..UNTIL..
LOOP..END  (an infinite loop, which can be left by an EXIT statement).
```

Shared variables are acessed by giving their name followed by an a character and a set expression giving all the paths that should be used for this reference. Set expressions are either some path names enclosed in set brackets or the name of a SET OF PPATH.

Declaration of variables used in the network invocation: Every type besides SET OF PATHs is allowed for these declarations. The syntactical form is the same as described above.

A SUBNET is a collection of PROCESSes with some internal connections and some PATHs to the outer world. They serve as a building block for more complicated structures. The structure of a subnet declaration is as follows:

```
SUBNET subnet name ( parameter list ):

CONST constant declarations
VAR variable declarations

BEGIN
  subnet body
END;
```

tant

able
and

ving
are
mple
ted:

The parameter list serves as a means to connect a subnet to the outer world. If the value of a parameter in a subnet is changed and this should effect the value of the actual parameter in the calling network invocation then the formal parameter must be preceeded by the keyword VAR.

The variable declaration inside a subnet follows the same rules as that for the network invocation. However these variables are only existing as long as the SUBNET is active. In case of recursive invocation of SUBNETs, for each invocation a new set of these variables is used.

The subnet body and the network invocation are constructed using

```
IF..THEN..ELSIF..THEN....ELSE..END
WHILE..DO..END
REPEAT..UNTIL..
LOOP..END
```

The crucial point in the construction of networks is the creation of PROCESS processor pairs. This is done by giving the process name followed by a parameterlist. These actual parameters serve as names for the paths of this particular PROCESS processor pair. Via the CONNECT statement a PPATH is connected TO an MPATH. This is done by modifying the code of the associated processes in such a way that the desired connection is established (see below.)

The network is built up starting at the first line of the network invocation. Subsequent statements of the network invocation (including subnet calls) are executed until the end is reached. The output of this procedure will be the loadable code for all the physical processors of the L-machine.

Programming Example

Programming examples from the AI and symbolic computation areas are given in the papers listed as references. In this paper we only want to clarify the essential features of the parallel L-language. Therefore we choose an example whose problem specification does not need any specific prerequisites and still shows the use of the language features that are characteristic for the L-language.

The problem:

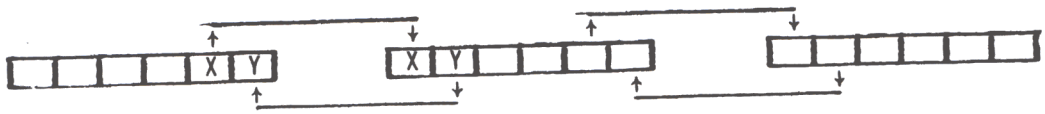
Given an array bar[0:800] we want to calculate the mean value

$$bar[i] := (bar[i+1] + bar[i-1]) / 2 \quad (\text{for } i := 1,\ldots,799).$$

This process will be iterated, say, hundred times. (As an interpretation, one could think about a bar of metal with some "temperature distribution" at the beginning whose temperature distribution is sought after a certain amount of time.)

A straightforward parallel algorithm for this problem:

We split the array into segments and use a processor for each such segment. Before a processor can start to compute all the mean values the first and the last value of the bar must be exchanged with those of the right and left neighbours in the following way:

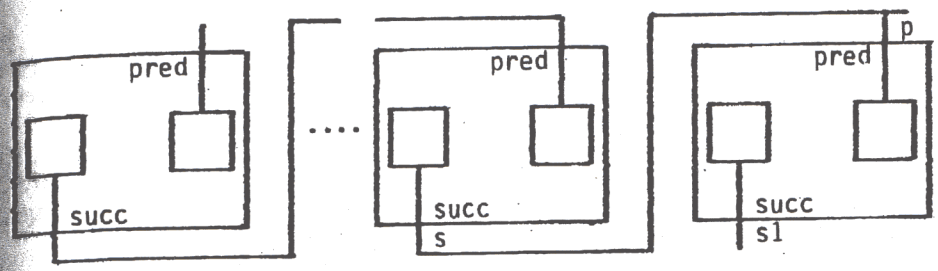


i.e. the elements at the margins of contiguous segments must be stored twice.

(In the program below an auxiliary array h will be used in order to store the new array during each iteration. After execution of one iteration the content of h is stored into bar.)

The network structure:

A network of the following structure is set up in the L-machine:



(The splitting of the array into segments, of course, could be done automatically in a preprocessor. For brevity, in the program below we do not show this preprocessing step and formulate the program for the case of a fixed number of 8 processors (segments)).

The L-program (description of the processes in the processors + description of the network structure):

```
PROGRAM Temperature;

SHARED VAR
  bar: ARRAY [0..100] OF INTEGER;

PROCESS PartOfBar(pred: MPATH; succ: PPATH; pos: INTEGER):
  (* pos specifies the number of a particular processor in the linear
  arrangement of processors.*)

VAR i, j: INTEGER;
  h: ARRAY[1..99] OF INTEGER;


BEGIN
  FOR j:=1 TO 100 DO  (* iterations *)
    IF pos>0 THEN SP ∂{pred}:=TRUE; WHILE SP ∂{pred} DO END END;
    IF pos < 7 THEN
      WHILE NOT SM ∂{succ} DO END;
      bar ∂{succ}[0] := bar ∂{MYSELF}[99];     (* exchange data *)
      bar ∂{MYSELF}[100] := bar ∂{succ}[1];
      SM ∂{succ} := FALSE
    END;
    FOR i:= 1 TO 99 DO
      (* compute mean values *)
      h[i] := (bar ∂{MYSELF}[i-1] + bar ∂{MYSELF}[i+1]) DIV 2
    END;
    FOR i:=1 TO 99 DO bar ∂{MYSELF}[i] := h[i] END
  END
END;

NETWORK

VAR p: MPATH; s, s1: PPATH; i: INTEGER;

BEGIN
  (* create a linear arrangement of processors *)
  PartOfBar (p, s, 0);   (* create first processor *)
  FOR i := 1 TO 7 DO
    PartOfBar (p, s1, i);
    CONNECT s TO p;
    s := s1
```

```
    END
  END.
```

(Some language specifications can be made more precise in this example:
+ The names of sensor bits are handled as boolean variables.
+ Each L-module has a memory path MYSELF and also a processor path MYSELF that are interconnected with each other.
+ A subnet invocation as, for example, PartOfBar(s,n1,i) assigns concrete physical paths to the path variables in the parameter list, for example to s,n1. Hence, PartOfBar(s,n1,i) overrides the assignment of s effected by PartOfBar(s,n,0) or the assignment of s effected by a previous PartOfBar(s,n1,i).)

Conclusions

The language implementation desribed in this paper is in progress. The first two phases (scanner and parser) of the compiler are completed, the code generation is near to completion. The interpreter for the network invocations has been designed. The implementation of the interpreter is in progress. The main purpose of having a concrete realization of the L-language is to have a practical means for easy experimentation with various network topologies. The know-how obtained from these experiments will be crucial for the isolation of special classes of topologies that are particularly suitable for certain classes of AI applications. The mathematical design and analysis of parallel symbolic algorithms for L-networks and the comparative study of other parallel architectures is currently pursued in various ph. d. and diploma theses in the frame of the L-project. References to the literature on various other parallel architecture projects may be found in /Bi, Bu 84/.

Acknowledgement

This work was supported by SIEMENS AG.

References

As 85
Aspetsberger, K., 85: Towards Parallel Machines for Artificial Intelegence: Realisation of the Alice Architecture by L-Components, this conference.

Bu 78
Buchberger, B., 78: Computer-Trees and Their Programming. Proc. 4th Coll. "Trees in Algebra and Programming", Univ. Lille, Feb. 16-18, 1978, pp. 1-18.

Bu, Fe 78
Buchberger B., Fegerl J., 78: A Universal Module for the Hardware-Implementation of Recursion. Univ. Linz, Inst. f. Mathematik, Report Nr. 106.

Bu 83
Buchberger, B., 83: Components for Restructurable Multi-Micro-Processor Systems of Arbitrary Topology. Proc. of MIMI 83 (Lugano), Acta Press Anaheim, pp. 67-71.

Bu 84
Buchberger, B., 84: The Present State of the L-Networks Project. Proc. of MIMI 84 (Bari), Acta Press Anaheim, pp. 178-181.

Bi, Bu 84
Bibel W., Buchberger B., 84: Towards a Connection Machine for Logical Inference. (Invited talk, Int. Symp. on Fifth Generation and Super Computers, Rotterdam, December 84)
To appear in: Future Generations Computer Systems, North Holland.

Wi, 82:
Wirth, N., 82: Programming in Modula-2. Springer-Verlag, Berlin Heidelberg New York.

NSC, 83:
National Semiconductor Corporation, 83: NS16000 Instruction Set Reference Manual.