1984-06-05A

MIMI '84 Bari

ISBN 0-88986-058-0

MINI AND MICROCOMPUTERS AND THEIR APPLICATIONS

Editor: G. Mastronardi

A PUBLICATION OF
THE INTERNATIONAL SOCIETY FOR MINI AND MICROCOMPUTERS — ISMM

ACTA PRESS
ANAHEIM * CALGARY * ZURICH

Proceedings of the ISMM International Symposium: MINI AND MICROCOMPUTERS AND THEIR APPLICATIONS, Bari, Italy, June 5-8, 1984.

ACTA PRESS
P.O. Box 2481
Anaheim, CA
92804, U.S.A.

ACTA PRESS
P.O. Box 3243, Stn. B
Calgary, Alberta
Canada, T2M 4L8

ACTA PRESS
P.O. Box 354
CH-8053 Zurich
Switzerland

ACTA PRESS CODE: 065

THE PRESENT STATE OF THE L-NETWORKS PROJECT

B. Buchberger

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

ABSTRACT

L-networks are networks of L-modules. An L-module consists of a processor (with private memory), a shared memory, a data and instruction path branching into m different paths under the control of the processor, and a data path collecting n data paths having access to the shared memory (m, n ... natural numbers). L-networks of arbitrary interconnection topology can be flexibly formed from L-modules. Typically, L-networks are used to implement parallel algorithms of a cellular, asynchronous type. At MIMI 83, Lugano, we described a hardware implementation of L-networks that allows the dynamic configuration of L-networks of arbitrary topology before or even during the execution of a parallel algorithm. The topology can be chosen such that it optimally corresponds to the interconnection topology inherent in the given parallel algorithm. In this paper, the structure of the "L-language" is described. The L-language is a high-level language that allows the formulation of L-algorithms, i. e. parallel algorithms for L-networks. The decisive innovative feature of the L-language is that, in addition to the L-programs to be stored in the L-modules, also the interconnection schemas of L-networks can be expressed in the L-language. More precisely, the L-language allows the recursive formulation of L-networks whose size and structure depends on parameters. For example, rectangles with m rows and n columns, rings with n nodes, trees with n levels etc. (m, n ... parameters) may be described in the L-language and can then be configured automatically.

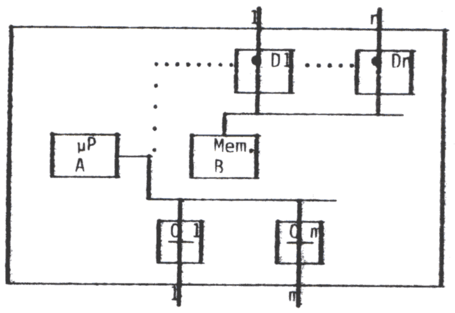
INTRODUCTION

L-networks have been introduced in /Buchberger 78/ as a hardware concept for the execution of arbitrary parallel algorithms, in particular, of parallel algorithms of a cellular, asynchronous type. At MIMI 83, Lugano, we described a hardware implementation of L-networks that allows the flexible configuration of L-networks of arbitrary topology before or even during the execution of parallel algorithms, see /Buchberger 83/. Further details of the L-network approach and a comparison of L-networks with other hardware systems for parallel algorithms may also be found in the references given in /Buchberger 83/. A recent overview on multi-microprocessor systems is /Paker 83/.

In this paper, only a very brief repetition of the concept of L-module and L-network is given in order to make the paper self-contained. The main emphasis of this paper is the description of the "L-language". This is a high-level language designed for fully exploiting the potential of the L-network implementation described in /Buchberger 83/: L-programs, i. e. programs written in the L-language, do not only contain a description of the programs stored in the L-modules of the L-network considered, but describe also the actual topology of the L-network. The description of the network topology in an L-program, typically, is recursive. This implies that topology descriptions of trees, rectangles, rings, pipelines etc. may depend on several parameters that describe the "size" of the respective structures. As far as we know, this feature of the L-language is new. It is essential for the description of parallel algorithms that need networks of variable size for their execution, i. e. networks whose size depends on the size of the input. For example, Todd's algorithm /Todd 78/ needs (log n) processors for sorting sequences of length n. This innovative feature of the L-language is fully supported (and not only "simulated") by the implementation of L-networks described in /Buchberger 83/. A language of this type needs also an extended concept of "program correctness". The main extension is that inductions have to be carried out also on the parameters that describe the network topology. First ideas for a formal proof system for the correctness of L-algorithms have been developed in /Buchberger, Aspetsberger 81/. Some examples of L-algorithms are given in /Aspetsberger 84/.

L-MODULES, L-NETWORKS, L-ALGORITHMS (REVIEW)

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

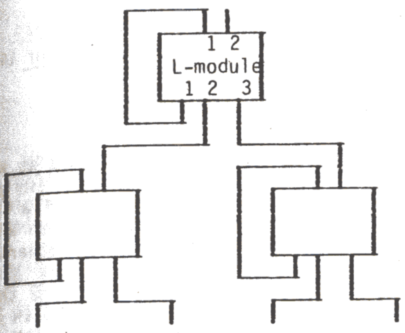


A: a microprocessor + private memory + some additional special circuitry.
B: a "shared" memory + some additional special circuitry.
C1,...,Cm: bus switches with an additional "open/close" facility. (The corresponding m bus branches are called "processor paths".)
D1,...,Dn: bus switches with a "sensor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

"open j", "close j"
"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 is explained in detail in /Buchberger 83/. Arbitrarily many L-modules can be combined to form "L-networks" of arbitrary (but fixed) regular (or irregular) structure, for example, to a binary tree:

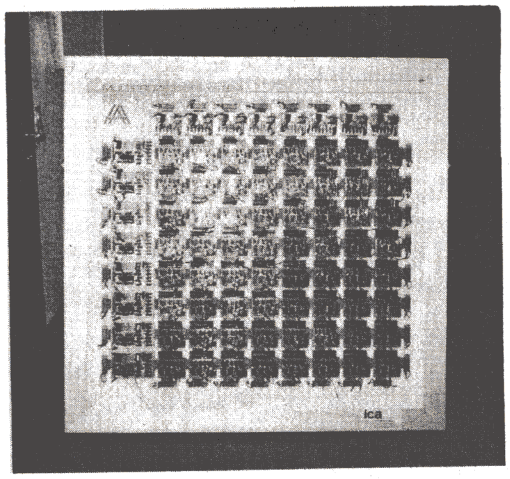


The numbers m and n must be chosen appropriately for the L-modules used for realizing a particular topology. (For example, m=3, n=2 in the binary tree).

THE PRESENT STATE OF THE L-NETWORK PROJECT

In /Buchberger 83/ the design of a hardware implementation is described that allows to configure L-networks of arbitrary topology from L-modules with arbitrary parameters m and n from a pool of components A, B, C, D. Note that the configuration and reconfiguration of L-networks is totally dynamic in this implementation. This means that the bus interconnections between the L-modules have not to be established mechanically by plugging cables but can be chosen under the control of the programmer before or even during the execution of a particular L-algorithm.

Meanwhile a pilot implementation based on the MC6502 consisting of a pool of 8 components A, 8 components B, 64 components C and 64 components D has been successfully finished. Note that the design is totally independent of the particular microprocessor chosen. Furthermore, the design is deliberately structured in such a way that it lends itself to a future VLSI implementation. With present-day VLSI technology, implementations incorporating several thousand components of type A seem to be possible. A rough impression of the structure of the pilot implementation using conventional technology is given by the following picture (the size of the device is approximately 1 m x 1 m):



The next steps in the L-network project are:
1. Implementation of a rudimentary loader.
2. Implementation of an extended assembler.
3. Implementation of a confortable monitor.
3. Implementation of the L-language.
4. Systematic study of parallel algorithm types.

The rudimentary loader has already been finished. It has the following tasks:
a. It accepts programs and data for the L-modules in the system written in hexadecimal code and stores the programs to the private memory of components A with specified physical numbers. The programs may contain arbitrary special instructions of the eight types listed above.
b. It simultaneuosly starts the programs.
c. It allows to read off the memory contents of shared and private memories after termination.

The extended assembler will be an extension of the MC6502 assembler with the addition of the special instructions of the eight types described above in mnemonic presentation.

The confortable monitor, essentially, will have the same objectives as the rudimentary loader. However, it will provide more convenient man-machine communication.

The objective and structure of the L-language will be described in the present paper. The implementation of the L-language will be the next main software goal within the project.

The systematic study of parallel algorithm types seems to be the most important research goal. Our main emphasis is on parallelism in symbolic (non-numerical) computation. It is generally accepted that vector and array structures are well suited for parallel numerical computation. However, much more systematic knowledge of practically useful network structures is necessary for making parallelism relevant for symbolic computation. As a result of our investigations so far, we believe that algorithms of time complexity $O(n^2)$ (or a similar complexity) are an interesting candidate for practically useful parallelizations (on L-networks): Such algorithms have a chance of allowing a parallel L-algorithm of, say, time complexity O(n) using O(n) L-modules. Thus, the speed-up can be drastic while the increase in hardware complexity is still practically manageable. The parallelization of Dijkstra's single source shortest path algorithm is an example of such an algorithm. The parallelization of other algorithms does not seem to be of that much practical importance. For example, $O(2^n)$ algorithms often have an easy O(n) parallelization using 2^n L-modules, see /Buchberger 78/. It is clear that such parallelizations, though resulting in a dramatic speed-up, are of no practical value because of the hardware complexity involved. On the other hand, there are also examples of O(n.log n) algorithms that allow an O(n) parallelization using O(log n) L-modules (consider, for example, Todd's sorting algorithm /Todd 78/). Although practically realizable, such parallelizations do not really result in a drastic speed-up.

THE L-LANGUAGE

An "L-program" written in the "L-language" consists of the description of the programs residing in the L-modules of an L-network <u>and</u> the description of the topology of the L-network. Some extension to ordinary programming languages are necessary in order to make such descriptions possible.

The <u>nucleus</u> of the L-language is an ordinary high-level language. Since, in this paper, we are not interested in syntax we use a "Pidgin-PASCAL" for notation. 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. A variable v denotes a storage region in all those shared memories that are "attached" to A via the "open" switches of type C (note that our use of "open" and "close" is just reverse to the normal use of these words in switching theory.)

The first extension to the nucleus is the possibility of declaring <u>private</u> variables. A variable declared private denotes a storage region in the private memory of a component A.

The next extension is the addition of the <u>special</u>

instructions of the eight types mentioned above in some convenient notation. (For example, in /Aspetsberger 84/

"S_j:= 1" is used for "set local sensor j" and

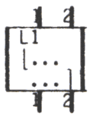
"a(j):= b(k)" is used for the sequence
"open k; p:= b; close k; open j; a:= p; close j",

where p is some variable declared private.) The parameter j in these instructions may be a variable or a constant. It denotes the name of one of the "processor paths" or "memory paths".

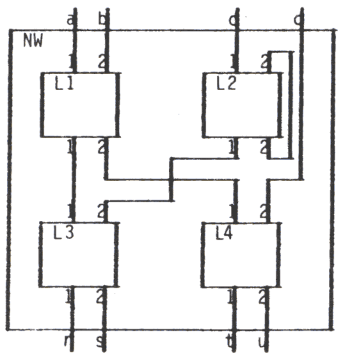
The names of the paths available in a particular L-program must also be declared. L-programs residing in one L-module, thus have the following typical structure:

```
net L1: elementary
       [private p, q;
        ....
        some program text using, for example,
           close 1; p:= a; open 2; if S2 = 1 then ....;
        ...]
        processor paths 1, 2;
        memory paths 1, 2.
```

This L-program describes an "elementary" L-network consisting of a single L-module with two processor paths named 1 and 2 and two memory paths 1 and 2. The processor component A of this L-module contains in its private memory the program [.....]. The L-module together with its program [....] is given the name L1 for later reference, in particular, for later use as a building block in more complicated L-networks. The same information may also be expressed in the following graphical notation of the L-language:



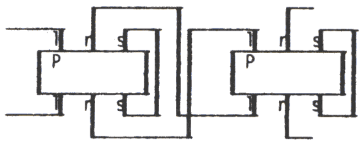
It should, then, be immediate, what is meant by the following graphical notation of an L-program NW:



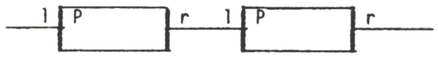
Here, L1,...,L4 are elementary L-programs of identical structure. They are combined in order to form a more complicated L-network having four (different) programs stored in the four private memories of the L-modules L1,...,L4 and realizing a particular interconnection schema between the L-modules. Some 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-program NW in syntactically more conventional "linear" notation reads:

```
net L1: ....; net L2: ....; net L3: ....; net L4: ....;
net NW: compound
      [L1; L2; L3; L4]
      processor paths
      r:= 1 of L3; s:= 2 of L3; t:= 1 of L4; u:= 2 of L4
      memory paths
      a:= 1 of L1; b:= 2 of L1; c:= 1 of L2; d:= 2 of L4
      connections
      1 of L1 with 1 of L3; 2 of L1 with 1 of L4;
      1 of L2 with 2 of L3; 2 of L2 with 2 of L2.
```

For simplification of notation, in most cases we will make the following assumption: in every L-network (in particular, in elementary L-networks) the name x of a path will always occur both as a name of a processor path and a name of a memory path. Furthermore, if not otherwise stated, if the processor path x of the L-network N1 is connected with the memory path y of the L-network N2 then, also, the processor path y of N2 is connected with the memory path x of N1. Also, the processor path with the special name "s" in an L-module is always connected to the memory path s of the same L-module (i. e. we assume that every L-module will always have access to its own shared memory via the path with the special name s). Thus, the following (part of an) L-program



may be replaced by the much simpler representation



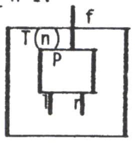
Note that we do not have to pay attention any more, where a path leaves a rectangle in order to distinguish between processor and memory paths: every path symbolizes a processor and a memory path.

RECURSIVE DEFINITIONS OF L-NETWORKS

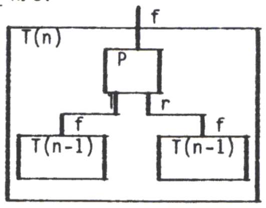
Typically, L-networks of highly regular structures are used for the parallelizations of algorithms and, in addition, the "size" of these regular structures depends on the value of the inputs for the algorithm. For example, for graphs with n nodes, n/(log n) L-modules are needed for the parallel shortest path algorithm in /Aspetsberger 84/. In order to define such variable size L-programs the names of L-programs must allow "parameters". As an example, consider the definition of an L-network with tree structure (compare the section on L-networks above and the parallel shortest path algorithm in /Aspetsberger 84/). In semigraphical notation the definition is:

net T(n): compound

case n=1:



case n>1:



Here, P is assumed to be some (elementary) L-network with paths 1,r,f. In linear notation the same definition is:

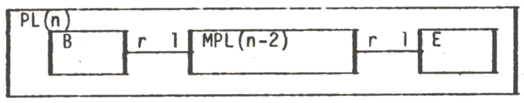
```
net T(n): compound

   case n=1:  [P]
              paths f:= f of P;
```

```
case n>1:   [ P; 2 copies of T(n-1) ]
            paths f := f of P;
            connections
            l of P with f of copy 1 of T(n-1);
            r of P with f of copy 2 of T(n-1).
```

We next give the definition of an L-program PL ("pipeline") with n L-modules in the L-language (semigraphical notation; the reader should now be able to translate it into the linear notation by himself). We present the definition "top-down" starting with the "main program".

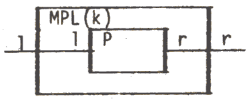
net PL(n): compound

case n>2:

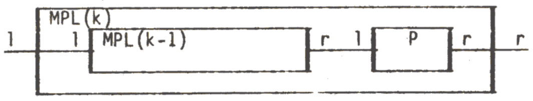


net MPL(k): compound

case k=1:

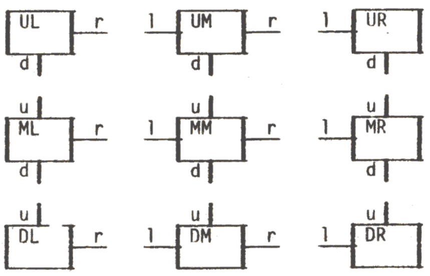


case k>1:

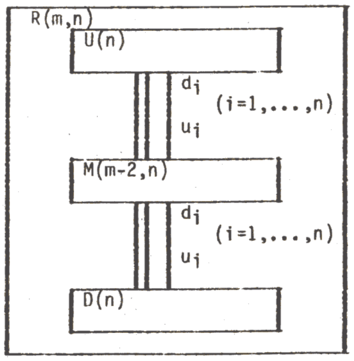


B, P, and E are assumed to be (elementary) L-programs.

Finally, we present the definition of an L-program R(m,n) ("rectangle" with m rows and n columns). This definition needs a new language feature: a variable number of paths u_i and d_i must be handled. We use the following elementary L-programs:



R(m,n) is then defined as follows:



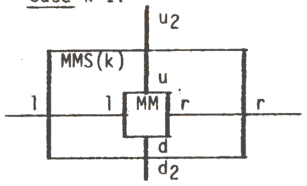
A linear notation for the path connections used in this definition is:

```
connections
for i:= 1 to n do
      d_i of  U(n) with  u_i of  M(m-2,n)
      d_i of  M(m-2,n)  with  u_i of D(n).
```

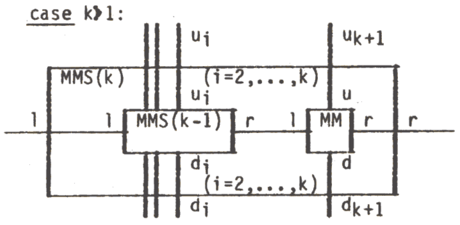
The definition of the rows U(n) and D(n) is similar to the definition of the pipeline. However, an increasing number of paths u_i must be handled. M(k,n), as a sequence of rows, is similarly defined. We demonstrate the language constructs by the definition of the middle part MMS(k) in one row of M(m-2,n):

net MMS(k):

case k=1:



case k>1:



CONCLUSIONS

We extended the concept of L-networks by the definition of the L-language, which is a high-level programming languages for programming L-networks. The implementation of a compiler for the L-language is the next major goal in the L-network project. A pilot hardware implementation of a system for realizing arbitrary L-networks has just been finished. Together with the L-language implementation this implementation will provide a flexible tool for experimenting with the design and implementation of parallel algorithms of arbitrary inherent interconnection topology of processing elements. The accumulation of know-how in the design of practically useful parallel algorithms should eventually lead to the design of an L-network hardware system with very large numbers of processing elements for parallel symbolic computation of a cellular and asynchronous type.

REFERENCES

- Aspetsberger, K., 84: Some Examples of Parallel Algorithms for L-Networks. These proceedings.
- 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.
- Buchberger, B., 83: Components for Restructurable Multi-Microprocessor Systems. Proc. MIMI 83, Lugano, Acta Press, Zürich, 67-71.
- Buchberger, B., Aspetsberger, K., 82: Proving the Correctness of Programs for Networks of L-Modules. Internal Technical Report, CAMP-81-3.0, Universität Linz, Institut für Mathematik.
- Paker, Y., 83: Multi-Microprocessor Systems. Academic Press, London, New York, 1983.
- Todd, S., 78: Algorithm and Hardware for a Merge Sort Using Multiple Processors. IBM J. Res. Develop., 22/5, Sept. 1978.

ACKNOWLEDGEMENT: This work is supported by the Austrian Research Fund (Project Nr. 3896).