Proc.6th Internat.Symp.on
Mini- and Microcomputers
Budapest 1980

A UNIVERSAL VARIABLE-TOPOLOGY MULTI-MICROPROCESSOR-SYSTEM

by
B.Buchberger,K.Aspetsberger

Johannes-Kepler-Universität
A-4040 Linz, Austria

ABSTRACT

The logical structure of a multi-microprocessor-system is described whose essential features are:

1. The system consists of identical microprocessor-modules.

2. The data exchange between the modules is effected by a crossbar communication network.

3. A complete system of sensor bits for transmitting control information between the modules is provided.

4. Each module operates autonomously.

5. Arbitrary types of interconnection schemata may be realized.

6. Typically, identical programs are loaded into the modules in order to realize parallel algorithms of "cellular" types, in particular recursive algorithms.

1. INTRODUCTION

The starting-point of the project described in this paper has been the desire for a hardware-realization of recursive algorithms in order to exploit the parallelism that is naturally inherent in certain recursive schemata. Actually, the design of a suitable hardware system resulted in the concept of a modular multi-microprocessor system that is universally applicable for realizing arbitrary parallel algorithms.

In the first stage of the project a hardware module (microcomputer + special interface) has been developed that may be used as a cellular building-stone for the formation of networks of arbitrary,but fixed topology and unlimited size (1,2,3,4).

In the present stage of the project a system is implemented that consists of modules (microcomputers) + special equipment for the formation of networks of arbitrary topology. The topology may be adapted to the structure of the algorithm by software means prior to the execution of the algorithm.

In a future stage of the project the system will augmented by a feature that permits the adaptati and variation (including extension and contractio of topology during the execution of the algorithm.

The projects described in (5 - 14) have simil design objectives as our own one. However, there a significant differences in the accents on the vario subgoals as well as in the details of realizatio The distinctive features of our concept are:

1. Extreme modularity of the logical structure of t system: for the programmer the system appears as co posed of identical modules ("cells") that have t full potential of cooperating autonomously with the neighbour modules.

2. Clear structure of the hardware: data processin data transfer and transmission of control informatic are the three functions that determine a natura decomposition of the hardware structure.

3. Extreme parallelization and flexibility topology by the use of a new type of crossbar system.

2. ANALYSIS OF A RECURSIVE ALGORITHM

As a motivation for the approach described in thi paper consider the following recursive algorithm fo the evaluation of arithmetical expressions:

```
eval(t,a): =
if   addterm(t)
then eval(opd1(t),a) + eval(opd2(t),a)
else a_ind(t).
```

For simplicity, only expressions t consisting of th operator "+" and variables are considered (for example $t:= (x_1 + (x_2 + x_3))$). The algorithm should be self-explanatory.

A typical computation according to the above algo rithm is shown in Figure 1.

The graphical representation of the computatio suggests a tree-like interconnection of processin modules in order to optimally exploit the parallelis that is apparently inherent in such a computatio

(parallel calls of "eval" in the procedure body of "eval").

3. A UNIVERSAL MODULE

The module shown in Figure 2 ("L-module") is proposed as a universal building-stone for computational net works of the above and, in fact, arbitrary topologies.

μP is a (micro)processor that has read/write access to its "private memory" PM and to the "shared memory" SM. Typically, the private memory contains the program. The shared memory may also be accessed by other modules via $I_1,\ldots,I_m$, however for writing only. Only one of the access paths to the shared memory may be active at a given time instant. The "sensor bits" $S_1,\ldots,S_m$ provide a means to transmit control information that allows the coordination of accesses to the shared memory SM by software means. $S_1,\ldots,S_m$ my be set (reset) from outside the module and read only from the μP inside the module. Complementary to the paths $I_1,\ldots,I_m$ for access from outside the module, the paths $O_1,\ldots,O_n$ are used to transfer data from the given module to the shared memory of neighbouring modules. Every path O_i is accompanied by a sensor-line V_i that enables the microprocessor to transmit control information to the sensor bits of its neighbours.

The above module is identical to the module described in (1,2) except that it is restricted to write-accesses via $I_1,\ldots,I_m$. This facilitates the hardware realization of the variable topology version (see Section 5). The universality of the module is not influenced by this restriction. Recently a similar module has been proposed in (7).

4. A NETWORK FOR THE SAMPLE ALGORITHM

The above sample algorithm can be executed on the network of L-modules shown in Figure 3.For realizing the above recursive algorithm, in each of the modules in the network the follcwing program should be executed, which results from a straight-forward modification of the procedure body of the recursive algorithm:

```
if   ¬ S3 wait;

if addterm(t)

then t':=opd1(t); a':=a; switch':=1;
     t":=opd2(t); a":=a; switch":=2;

     V1:=true; V2:=true;
     if ¬ S1 wait;  V1:=false; if S1 wait;
     if ¬ S2 wait;  V2:=false; if S2 wait;

     V3:=true; if S3 wait;
     if switch = 1
       then b"':= b+c
       else c"':= b+c;
     V3:=false;

else V3:=true; if S3 wait;
     if switch = 1
       then b"':= a ind(t);
       else c"':= a ind(t);
     V3:=false;
```

In the formulation of the program the following rul has been used:

Address Modification Rule:

If the identifier a' is used in module M it addresse a storage region in the shared memory of the neigh bour module M_1 of M, which is interconnected with I via path O_1 and sensor-line V_1. The neighbour M_1 ma access the same storage region by the identifier a The analogous rule applies to a", a"',...

By the concurrent execution of the above identica programs in the modules a "computational wave" i generated in the modules that exactly corresponds t the computation described by the recursive algorithm The instruction sequences involving the sensor bit realize the handshakes necessary for the synchroni zation of the processes going on in the modules, i particular for the coordination of the accesses t the shared memories. The value of the variabl "switch" tells a module whether it is the "left o right son" of its father. (A detailed explanation o the above computational process and various furthe examples may be found in (1) and (4). The reader i advised to carry out the computation with paper an pencil.)

5. REALIZATION OF VARIABLE TOPOLOGY

The interconnection topologies for the modules tha optimally exploit the parallelism of algorithm display a great variety (trees of various shapes, pipe-lines, rectangular nets, hexagonal nets, double-trees, etc., see (8) for a systematic treat ment. In (15) various typical parallel algorithm have been formulated for networks of L-modules.)

Our objective, therefore, is to construct a hardwar system, which permits to compose L-modules i networks of arbitrary topology by simply loading acc special storage cells prior to execution of th algorithm. For the programmer the appearance of th system should not be altered by this additiona feature. (In particular the autonomous character o the L-modules should be preserved). Also, durin execution of the algorithms, the performance of th networks should be the same as if the L-modules wer connected by fixed data-paths and sensor-lines, i.e. the additional hardware-feature should not act as a "hidden sequentializer". (For instance, a common bus would sequentialize parallel processes).

These severe design constraints lead to the solutio shown in Figure 4 whose feasibility, using present day hardware-components, is demonstrated in th companion paper (16). The system consists of thre main parts: a pool of L-modules M_i, a crossbar syste

for data transfer and a network for the transmission of control information to and from sensor bits.

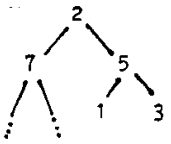
Logically, the L-modules in the pool with physical numbers 1,...,k are identical to the L-modules of Section 3. As a part of the above system, however, they have the structure shown in Figure 5. The module has only one data path I and one data path O. In compensation, it has a special memory for the address mapping AM. The address "a" of a store instruction "STORE a" is decomposed in two parts a = (n,r), where n is the logical number of the neighbour and r is the relative address of the storage cell in the shared memory of the n-th neighbour. By means of the address mapping AM the logical number n is translated into a physical number n. n and r (together with the data to be transfered) are transmitted via O to the crossbar system.

Using the physical number n the crossbar system conducts the data to the destination module, where it is stored at address r (note that this realizes the address modification rule of Section 4). In the most extreme case all modules in the system may transfer data via the crossbar system simultaneously. The crossbar system is not responsible for the solution of access conflicts to the shared memories of the modules. Instead, the programmer is exclusively responsible for avoiding access conflicts by an appropriate use of the sensor instructions (see the sample program).

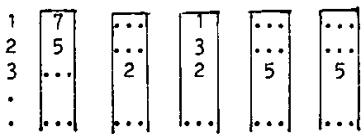
The unidirectional character of data transfer, the fact that the crossbar system does not have to take care of access conflicts, and the idea (16) of a special serial realization of data transfer seem to make the above crossbar system an interesting solution also for large numbers of modules ($k \geq 100$).

Logically, the sensor-bit array must provide a sensor bit for every module in every module with the interconnection pattern shown in Figure 6 (for k = 3). The sensor instructions V_n:=true, V_n:=false, if S_n then ..., where n is the logical number of a neighbour, have also to be relativized by means of the address mapping AM. If module M is connected to module M' by connecting O_i of M with I_i of M' and module M' should also be connected with module M, then O_i of M' should be connected with I_i of M. By following this rule only one address mapping is needed in each of the modules. In (16) an elegant method for practically realizing the above array without connecting every module with every module is given.

As an example for the use of the address mappings consider the network in Section 4. Suppose that it should be realized by using the modules with physical numbers 2, 7, 5, 1, 3 in the following positions:



Then the address mappings of these modules must contain the following values respectively:



In this example one also sees that the variation of network topology during the execution of the algorithm is a design objective of extreme importance: If one wants to execute the sample program for arbitrary terms (of a maximal "depth" d) then one has to provide a binary tree of depth d. For a concrete input term t, however, only a small part of this network will be used. On the other hand, a tree t of depth $d' > d$ is not manageable any more though it may contain less than 2^d "nodes".

The variation of network topology during execution of the algorithms may be achieved in the proposed system simply by adding a control processor to the system that may change the address mappings of the modules during execution. Simultaneous calls of the control processor by different modules must, of course, be sequentialized and thereby will destroy the extreme parallel character of the present system. This feature will be added to the system in the next stage of the project.

For this final system the sample program could have the following form:

```
if ¬S3 wait;
if addterm(t)
   then generate(1,3); t':= ...
        generate(2,3); t":= ...
        V1:= true; ... V3:= false;
        delete
   else ...
```

Generate(1,3) calls the control processor for starting a new module whose address mapping must be loaded such that, logically, it is the "left son" of the calling module. Delete returns the calling module to the pool of "free" modules.

6. A FURTHER EXAMPLE

The conjunctive normal form C(t) of a boolean expression t may be generated by the following recursive rules:

$$C(1) = 1 \quad \text{(if 1 is a literal)},$$
$$C(\neg\neg t) = C(t),$$
$$C(\neg(t_1 \wedge t_2)) = C(\neg t_1 \vee \neg t_2)$$
$$C(\neg(t_1 \vee t_2)) = C(\neg t_1 \wedge \neg t_2)$$
$$C(t_1 \wedge t_2) = C(t_1) \wedge C(t_2) \qquad (*)$$
$$C(t_1 \vee t_2) = \text{Multiply}(\, C(t_1), C(t_2)\,)$$

Multiply is a procedure that "multiplies" two conjunctions of clauses by applying the distributive law. The structure of the algorithm C is essentially the same as that of the algorithm in Section 2. A tree-like network of L-modules will exploit the

parallelism contained in line (*). One can achieve a further parallelization in this example by adding a "multiplication network" to the tree (shown for depth 3 and a "balanced" tree in Figure 7).

Here, C_i are (modules containing) clauses (= disjunctions of literals) and D_j are the results of the "multiplication"

$$(C_1 \wedge C_2) \vee (C_3 \wedge C_4)$$

in the form

$$D_1 = C_1 \vee C_3, \quad D_2 = C_1 \vee C_4,$$
$$D_3 = C_2 \vee C_3, \quad D_4 = C_2 \vee C_4.$$

Note that a very "dense" network is necessary to optimally exploit this parallelism. It is essential in such examples that a huge number of data transfers between the different modules may be carried out simultaneously without the interference of a central management. This is a strong motivation for reconsidering crossbar systems as it is done in our project.

The parallelization of recursive algorithms normally leads to asynchronous computations. The multi-microprocessor-system considered in this paper, however, is well suited for synchronous computations as well, for instance computations of the array-type and of the pipeline-type see (15).

7. CONCLUSION

A multi-microprocessor-system has been described that may realize arbitrary interconnection topologies between L-modules in order to optimally fit the system to the computational structure of parallel algorithms. For the programmer the system appears as a network of autonomous modules. It should be clear from the discussion that, in particular, also data-driven (7,12) and result-driven (11) language-constructs may be implemented on the system in a natural way. The variation of topology during execution of the algorithms appears to be the next natural step in the project. It may be realized simply by making the address mappings accessible to a control processor.

REFERENCES

(1) B. Buchberger.
Computer-Trees and Their Programming. Proc. 3rd Coll. "Les arbres en algebre et en programmation", Lille (1978), 1-18.

(2) B. Buchberger, J. Fegerl.
A Universal Module for the Hardware-Implementation of Recursion. T. Report Nr. 106, Univ. Linz, Inst. f. Mathematik (1978).

(3) B. Buchberger, J. Fegerl, F. Lichtenberger.
Computer Trees: A Concept for Parallel Processing. Microprocessing and Microsystems, 3/6 (1979), 244 - 248.

(4) F. Lichtenberger.
Speeding up Algorithms on Graphs by Us: Computer Trees. Graphs, Data Structures, Al rithms (M. Nagl ed), Hanser Verlag, Münc (1979), 65 - 79.

(5) R. Kober, C. Kuznia.
SMS 201 - A Powerful Parallel Processor w: 128 Microprocessors. Euromicro Jour 5(1979), 48 - 52.

(6) W. Händler, F. Hofmann, H. Schneider.
A General Purpose Array with a Broad Spect of Applications. Workshop GI "Comput Architecture", Erlangen (1975).

(7) H. Schreiber, V. Sigmund.
Functually Structured Computer Architect Using Plug-in Micro-Processing Modules. Pr IMMM79, Geneva (1979), 204 - 212.

(8) H. T. Kung.
The Structure of Parallel Algorithms. Report, Carnegie-Mellon Univ., Dptmt.Co Scie. (1979).

(9) S. Fuller, J. Ousterhout, L. Raskin, P. Rubinfeld, J. Pradeep, R. Swan.
Multi-Microprocessors: An Overview and Worki Example. Proc. IEEE 66/2 (1978), 216 - 228.

(10) J. Syre, et al.
Pipelining, Parallelism and Asynchronism the LAU System. Proc. Int.Conf. "Paral Processing" (1977), 87 - 92.

(11) P. Treleaven, G. Mole.
A Multi-Processor Reduction Machine for Us Defined Reduction Languages. T. Report, Uni Newcastle, Comp.Lab. (1980).

(12) J. Dennis, D. Misunas.
A Preliminary Architecture for a Basic Dat Flow Processor. Proc. 2nd Ann.Symp. "Comput Architecture", MIT (1975).

(13) B. Sullivan, T. Bashkov.
A Large Scale Homogeneous, Fully Distribut Parallel Machine I,II. Proc. 4th Ann.Sym "Computer Architecture", CACM-SIGARCH, 5 (1977), 105 - 117, 118 - 124.

(14) R. Albrecht
Konzept eines Mehr-Processor-Parallel-Reche werks. T. Report, Univ. Innsbruck, Inst. Informatik (1980).

(15) K. Aspetsberger.
Algorithmentypen für Multi-Mikrocompute Systeme. Diplomarbeit, Univ. Linz, Inst. Mathematik (1980).

(16) B. Quatember.
A Hardware-Realization of a Variable-Topolo Multi-Microprocessor-System. These proceedin

Acknowledgement: The project is supported by t Austrian Research Fund (Project Nr.3896). We al gratefully acknowledge valuable discussions with Dr. F. Lichtenberger.

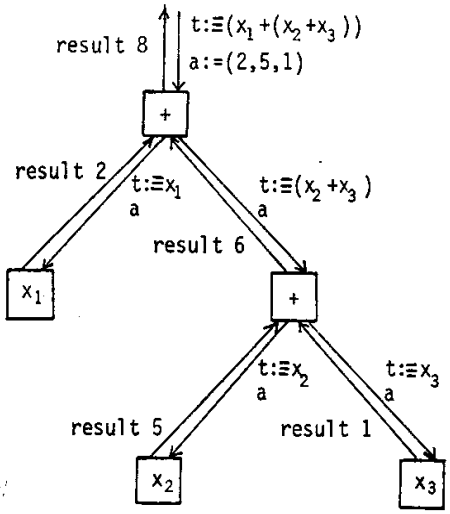


Figure 1

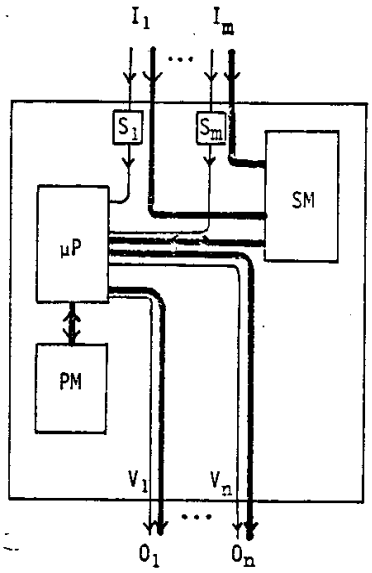


Figure 2

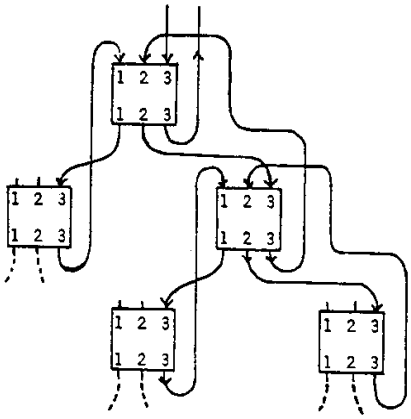


Figure 3

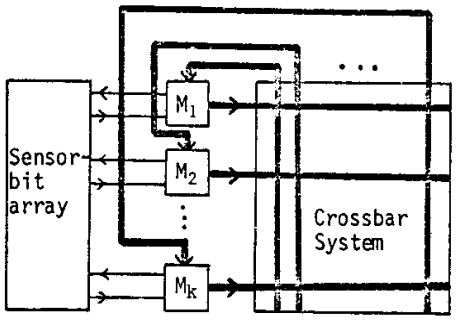


Figure 4

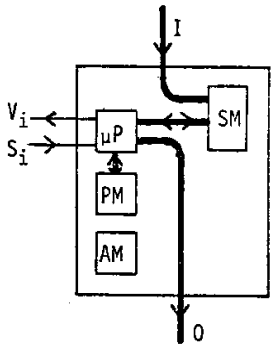


Figure 5

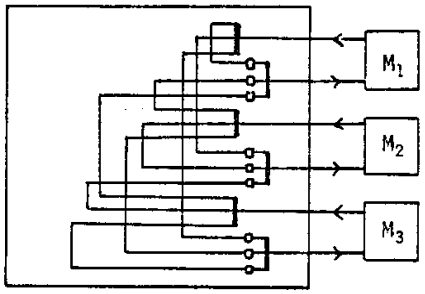


Figure 6

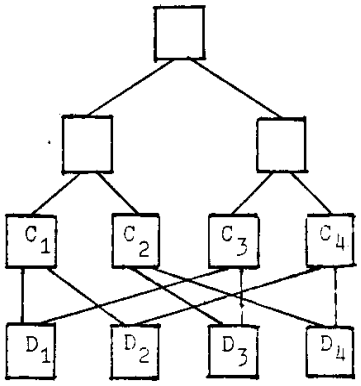


Figure 7