A SIMULATION OF DATAFLOW COMPUTER MODELS AND A MODEL OF FAULT TOLERANT DATAFLOW COMPUTER

UDK 681.3.068

Milan Ojsteršek, Viljem Žumer, Peter Kokol, Anton Zorman
Tehnical Faculty Maribor

ABSTRACT: Dataflow architecture is accepted as the architecture for future computers because it can exploit potentialy all the parallelism in a program. This architecture is assumed to execute dataflow graphs. The nodes in the dataflow graphs represent asynchronous tasks. The arcs connecting nodes represent communication paths for the messages (tokens) generated by nodes or suplied from the external environment. Each node is executed (fired) when the required input becomes available. The dataflow arhitecture proposals can be classified as static and dynamic architectures. In a static architecture the nodes of a program graph are loaded into memory before the computation begins and at most one instance of a node is enabled for firing at a time. A dynamic architecture facillitates the firing of several instances of a node at a time and these nodes can be created at runtime. We have developed program language DFL1, simulator for the simulation dataflow computer models and their key mechanisms, improved hardware, software and set of instructions needed to support an efficient fault-tolerant dataflow computer.

1 INTRODUCTION

Over the last few years data flow computer architectures have been proposed and some computers have been built. A simple data flow execution model is based on a "pure" data flow program organisation: with an instruction being enabled when all its input tokens are available. Each instruction may have any number of inputs and any number of outputs. Discussing the U-interpreter, Arvind and Gostelow suggest that maximum parallelism can only be extracted from a program if an arc is allowed to carry more than a single token - a process achieved by carrying a label with each token that identifies the context of that particular token (ARVI 82). Packet communication machine organisation and high level computer language with exploiting inherent parallelism in data flow graph is the most significant for the data flow architecture. We have developed program language DFL1 (KOK 84) and simulator for the simulation of five various data flow computer models, which are suited for todays technology. During the simulation, simultaneous execution of a data flow program in distinct units (token transmission and matching, forming activities) and some important statistical parameters (i.e. an input stream, a business period, an idle period, a service time ...) are observed (OJST 84, ZUM 85-1, ZUM 85-2, OJST 86, OJST 87-2).

2 DESCRIPTION OF MODELS

Our models are based on the packet communication machine organisation with token matching and consist of an input section, memory sections, a global store, processor sections, an output section and units for communication among sections. The input section decomposes the data flow program and supports memory sections with input data. We have used two decomposition methods:

- Each instruction into its own memory section. If there are more instructions than memory sections, then more instructions are stored in one memory section. Instructions are randomly distributed.
- Each block into its own memory section. If there are more blocks than memory sections, then more blocks are stored in one memory section.

The memory section matches tokens into sets of tokens. When all of its input tokens having the same context are available, it forms activity (executable instruction which consists of a set of tokens with the same context and a copy of instruction) is formed and sent into an processor section. The processor section executes the activity and sends output results into memory sections. The output section collects final results of computation. The global store is used to prevent deadlock of the system. Communication units (networks, busses) transmit messages (tokens, sets of tokens, activities). They use post office interconnection principle for transmission messages. Communication units in our simulation are simulated as delay units.

2.1 Model 1

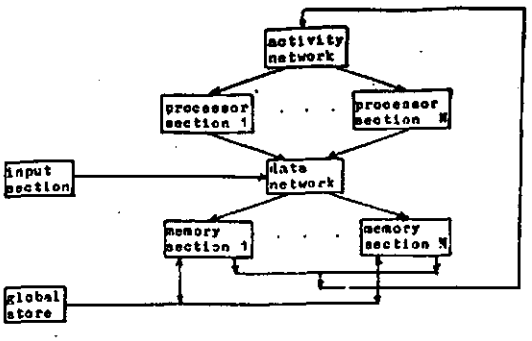


Figure 1:The model 1

The model 1 uses two networks: one for transmitting activities and one for transmitting data. It has separate memory sections and separate processor sections. The memory section consists of an input FIFO

queue, a matching store, an instruction store, an associative logic and an output FIFO queue. The processor section consists of an input FIFO queue,a processor, an output FIFO queue. This configuration is not suitable for super-systems, because networks cause too much delay time in transmitting messages. An advantage of this model is autonomic work of units. The model is suitable for small systems and for a subsystem of a large system.

2.2 Models Connected In Clusters

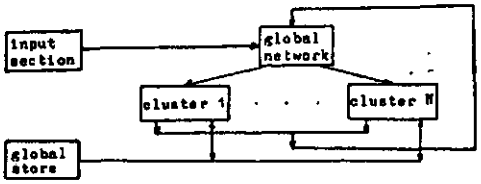


Figure 2:Models connected in clusters

Models 2,3,4,5 are very similar. Sections of models are connected in clusters.

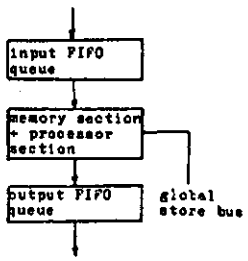


Figure 3

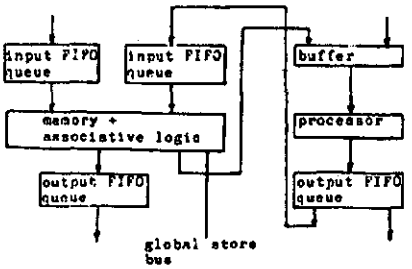


Figure 4

Figure 3: The cluster of model 2
Figure 4: The cluster of model 3

2.2.1 Model 2

The memory section and the processor are associated in one element (figure 3). The memory section matches tokens into sets of tokens, forms and executes activities. Output tokens which don't have their own instructions in the cluster are transmitted into output FIFO queue. Other tokens are matched in the cluster. Higher speed of forming and executing activities is the advantage of this model. There is no additional transmitting betwen memory section and processor. This model is very suitable for small and medium systems if the adequate decomposition method is used.

2.2.2 Model 3

The processor and the memory section are separated (figure 4). If the processor is idle and the memory section hasn't any activities, it can receive activities from other clusters. If the memory section has activities and the processor isn't idle, activities are transmitted into the output FIFO queue. The global network transmits it to the idle processor section.

2.2.3 Models 4 And 5

Models 4 and 5 are similar to model 3. In each cluster they have one or more processors. The memory logic establishes which processor is idle and adresses the activity to it. Model 4 is slightly better than model 5, because it needs less time for writing in FIFO queues.

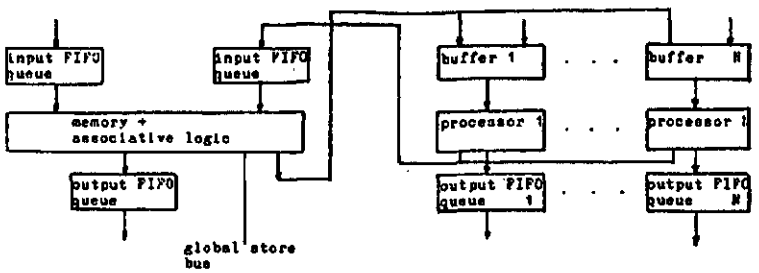


Figure 5:The cluster of model 4

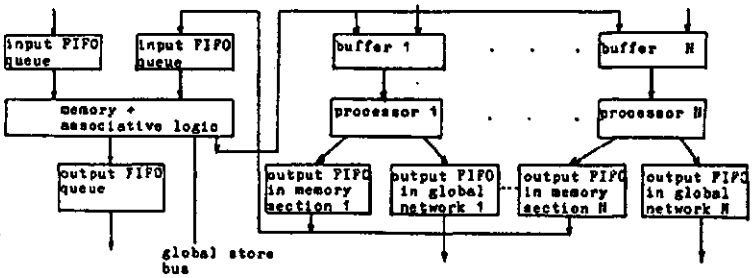


Figure 6:The cluster of model 5

3 SIMULATION RESULTS

98 different parameters can be varied in our simulation but it is reasonable to vary just some of them. Constant values are given to all others. Our models are simulated with four programs, the blocks of which are composed of data flow graphs shown on figures 7,8,9,10. Program blocks are independent and they have no common flow of data tokens.

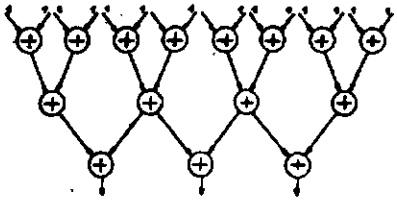


Figure 7

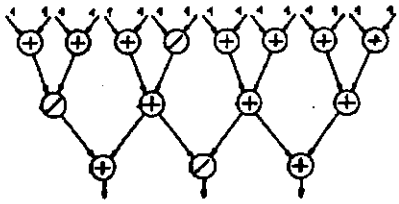


Figure 8

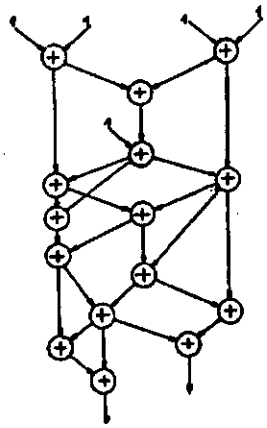


Figure 9

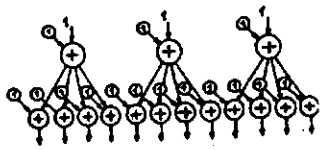


Figure 10

All data flow graphs have 15 instructions. Graphs shown on Figures 7 and 8 have many inherent parallelism and need many input data tokens. The graph shown on Figure 8 consists of instructions /, which are executed 30 times slower than instructions +. Such an instruction delays the execution of following instructions. The graph with few inherent parallelisms is shown on Figure 9. On Figure 10 the graph with many inherent parallelisms and three input data tokens is shown. Each instruction in the graph shown on Figure 10, has one input operand and one constant. This increases the speedup of firing enabled instructions. How are algorithms influenced to our models is described and evaluated.

Our simulation is executed with these constant input parameters:

- an instruction + is executed in 2 cycles
- an instruction / is executed in 60 cycles
- matching one token in a set of tokens is executed in 1 cycle
- activity forming is executed in 1 cycle
  FIFO queues have 5 elements
- writting or reading time to FIFO queues is negleable
- a network can transmit an infinite number of messages
- blocks of program are independent
- a model 1 has 5 memory sections

In the file of input parameters we have varied the following parameters:

- the selection of system (1 - 5)
- the decomposition method
- the delay time in transmitting messages through the network (0, 1, 2 cycles)
- the capacity of memory section and a capacity of global store
- the input algorithm

The program consists of an equal number of blocks and memory sections.

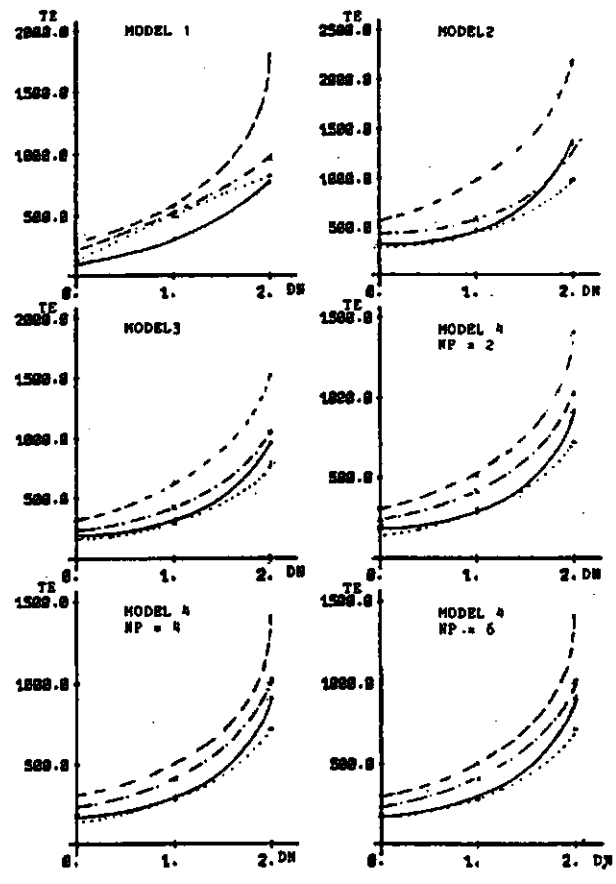
We have reduced the number of observed output parameters to the minimum following ones: a program execution time requested for execution of 200 instructions, an average number of busy processors, an utilization of processors, an average number of busy memory logic in memory sections, an average number of sets of tokens, which are matched in memory section, an utilization of a matching store, an average number of sets of tokens, which are matched in global store, and an utilization of a global store.

The following abbreviations are used in diagrams:
UP - the utilization of processors (%)
DN - the delay time in transmitting messages through the network (cycles)
NP - number of processors
TE - the program execution time (cycles)

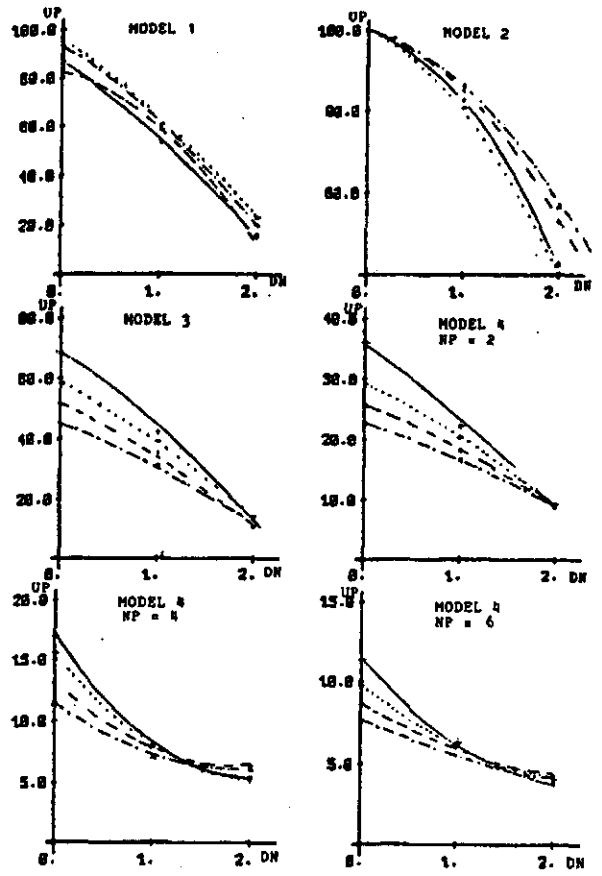
———	algorithm shown on Figure 7
- - - -	algorithm shown on Figure 8
—·—·—	algorithm shown on Figure 9
·····	algorithm shown on Figure 10

For the model 1 the decomposition method "each instruction into its own memory section" is better than "each block into its own memory section" , so we have made diagrams which evaluate the model 1, only for this method. For models 2,3,4 we have made diagrams only for "each block into its own memory section" decomposition method because this method is better.

The program execution time dependent on the delay time in transmitting messages through the network



The utilization of processors dependent on the delay time in transmitting messages through the network



The algorithm shown on Figure 10 is the fastest one (it has a maximum degree of inherent parallelism and a minimum number of input data). Proportions in executing time of program and an utilization of units are changed by increasing the delay time in the transmission through the network. An executing time of program is dependent on an inherent parallelism of algorithm and a number of input data needed for the execution of this algorithm. The executing time of algorithm is decreased to some limit and then it keeps constant value (all inherent parallelisms are used), if the number of memory sections or the number of processor sections are increased or delay time in transmission through the communication units is decreased. Owing to the fact that the processor section and the memory section of model 2 form one element, execution time of model 2 is almost twice as long compared to model 3; but its advantage is the best utilization of units. Because delays of reading and writing into FIFO queues were not considered, the results of models 4 and 5 are identical. With this considerations we have found that model 4 is better than model 5. The biggest average number of sets of tokens in the memory section is in the algorithm shown on Figure 9, because instruction / delays the execution of following instructions. On model 1 we have simulated the deadlock of a system. If matching stores are full, then other tokens must match in the global store. This causes bad utilization of units in a model (a lot of time is wasted for communication between memory sections and the global store). We have simulated the deadlock with decreasing capacity of matching stores. A *good* decomposition method can prevent deadlock or bad utilization of a system.

4 FAULT TOLERANT DATAFLOW COMPUTER

Results of simulation and our knowledge about dataflow computers are shown that there are many problems to be solved. In the design of an actual dataflow computer the main problems are:

- construction of a large and fast matching memory at a *reasonable* cost,
- construction and management of a structure memory,
- network construction,
- interruption, error and exception handling,
- inefficiency due to insufficient parallelism,
- developing efficient control, partitioning and scheduling algorithms.

We have developed key mechanisms, improved hardware, software and set of instructions *needed* to support an efficient fault-tolerant dataflow computer (OJST 87-1, OJST 87-3).

Proposed dataflow architectures are very inefficient on regular structures because of fine granularity of their operations. When data is structured (vectors, matrices, *records*) the *control* and data flow is very regular and predictable and there is no need to pay high overhead for scheduling. These architectures don't have mechanisms for interruption, error and exception handling, a mechanism which reassigns nodes to another unit if faults have been detected, a mechanism which stops sending tokens to the faulty processor *and* a mechanism which destroying read/write requests in memory after fault. Contents of data buffer in the output port of the fail processor are lost.

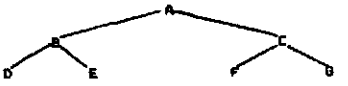


Figure 11: Task Invocation Structure

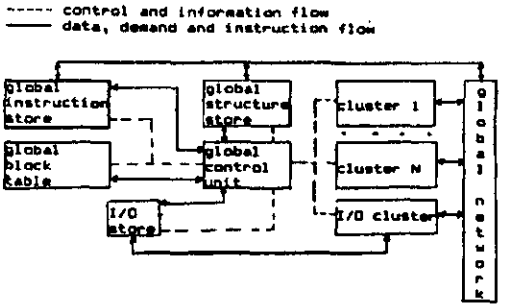


Figure 12: Global level of Fault Tolerant Dataflow Machine (FTDFM)

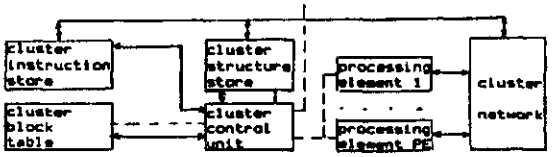


Figure 13: Cluster level of FTDFM

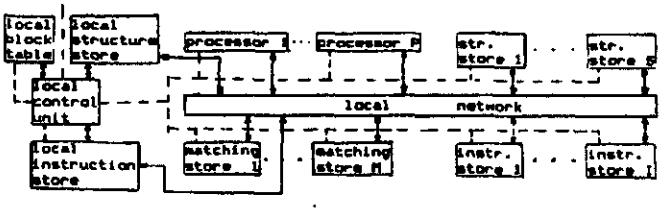


Figure 14: Processing Element of FTDFM

In our computer model (figures 12,13,14) the combination of control flow, data flow and demand driven are used. We adapt to the granularity of the data structure and create large structures as one object. We *reduce scheduling* overhead by combining together as many scalar operations as possible and executing them as one

object.

Hierarchical decomposition method (ARVI 85) are used. This method is based on the concept of resource bounded graphs (RBGs). The RBG is a program fragment for which a bound on the resource requirement of the fragment can be derived at compile time as a simple function of a collection of parallelism parameters. A program is viewed as a collection of RBGs.

The execution of a program can be represented by a tree of task invocations (figure 11). This method use breadh-first partitioning algorithm until sufficient parallelism is generated and then it use depth-first partitioning algorithm. With depth-first partitioning algorithm deadlock are avoided.

Our system is a dynamic architecture for task level dataflow with facilities to support node reassignment when processors fail. It uses tagging scheme similar to the MIT dynamic architecture. It have three hierarchical levels. The purpose of using three hierarchical levels is to execute programs efficiently by utilizing principles of locality. Each hierarchical level have an instruction store, a block table, a structure store and a control unit. The instruction store holds copies of the instructions which are executed in this level. The structure store holds input data structures (DS) of RBG which are executed in this level. The block table contains adresses of units where RBGs are executed. The control unit adresses RBGs and its DS to units in this level and generates block table. If hardware faults occur in one of the units, control unit reassigns RBGs and its DS to healthy units. With this mechanism we achieve that only the RBG and their DS which is assigned in the faulty unit must be reexecuted. Control unit also deletes RBGs and DS when the computations of them are finished. It also controls the amount of available memory and the utilization of units. If deadlock is occured the control unit reassigns RBGs, its sets of tokens and DS from too busy units to idle units.

Our proposed model has high bandwidth (HB) and low bandwidth (LB) communication paths. HB paths are intended for transmaiting RBGs, DS, activities and tokens betwen units. The LB paths are intended for transmaiting status, diagnostics, control and measurement information. Our model has I/O instructions, instructions which are executed in exception condition, instructions implementing the special operators for dynamicaly creating instances of node resulting from recursive nodes in dataflow graphs, loop and streams, table-oriented instructions (for readind an entry, writing into an entry and modifying parts of an entry), structure-oriented instructions (for selecting an element from a structure, appending an element to a structure and testing for emptiness), string manipulation instructions (to search for a substring and a lenght of a string to compare strings, and concatenate strings), stream oriented instructions, fixed-point instructions, logical/shift instructions, floating-point instructions, compound instructions (for reducing token movement).

5 CONCLUSION

Today a vast collection of single-board computers are available which offers roughly 1 MIPS at low cost: these are touted as building blocks for multiprocessors. Can dataflow machines compete? It is not clear if a single dataflow processor can achieve the performance of a von Neumann processor at the same hardware cost. The dataflow instruction-scheduling mechanism is clearly more complex than incrementing a program counter. An engineering effort substantially beyond any of the current dataflow projects is required to make fair comparison. The SIGMA-1 (SHIM 86) project is an important step in this direction. The question becomes more interesting when we consider machines with multiple processors, where the dataflow scheduling mechanism yields significant benefits. The dataflow approach can be viewed as an extreme solutions to the memory latency problem - the processor never waits for responses from memory: it continues processing other instructions. Instructions are scheduled based on the availability of data, so memory responses are simply routed along with the tokens produced by processors. It is our belief, that dataflow architectures together with improved hardware, software, set of instructions needed to support an efficient fault tolerant dataflow computer and with powerfull high level functional languages, will show the programming generality, performance and cost effectiveness nedded to make parallel machines widely applicable.

6. References

(ARVI 82) Arvind, Gostelow K. P.:
"The U-interpreter"
IEEE Computer, February 1982, pp. 42-48.

(ARVI 85) Arvind, Culler D. E.:
"Managing Resources in a Parallel Machine",
Proc. of the IFIP TC-10., Conference on
Fifth-generation Computer Architecture,
Manchester U.K., July 1985, pp. 103-121.

(KOK 84) Kokol P., Stiglic B., Zumer V.:
Pretvorba aplikativnega jezika v grafe
pretoka podatkov", ETAN, XXVIII
Jugoslavenska konferencija Split,
4.- 8. juna 1984 Split,
pp. IV. 547 - IV 554, - in slovene.

(OJST 84) Ojsteršek M.:
"Simulacija podatkovno vodenega računalnika"
Tehniška fakulteta Maribor,
Maribor 1984 - diplomsko delo -in slovene.

(OJST 86) Ojsteršek M., Zumer V., Kokol P.:
"Data Flow Computer Simulation",
in Proceedings of the 8th International
symposium Computer at the University,
Cavtat, pp. 2.07 1 - 2.07 10, May 1986.

(OJST 87-1) Ojsteršek M., Zumer V., Kokol P.,
Zorman A.: "Izboljšana strojna in
programska oprema ter nabor instrukcij,
ki omogočajo delovanje učinkovite in na
napake neobčutljive podatkovno vodene
arhitekture", XI Bosanskohercegovački
simpozium iz informatike "Jahorina 87",
zbornik radova, knjiga I, Sarajevo,
april 1987, pp. 150 1-8 - in slovene.

(OJST 87-2) Ojsteršek M., Zumer V., Kokol P.:
"Dataflow Computer Models",
COMP EURO 87, VLSI and Computers, Hamburg,
may 1987, pp. 884-885.

(OJST 87-3) Ojsteršek M., Zumer V., Kokol P.,
Zorman A.: "An Overview and Comparison
of Data Flow Computer Systems", Proceedings
of the 9th International Symposium Computer
at the University, Cavtat, Yugoslavia,
pp. 2R.06 1 - 2R.06 6, May 1987.

(SHIM 86) Shimada T., Hiraki K., Nishida K.,
Sekiguchi S.: "Evaluation of a Prototype
Data Flow Processor of the SIGMA-1 for
Scientific Computations", Proc. of the 13th
Annual International Symposium on Computer
Architecture, Washington, USA, 1986,
pp. 226-234.

(ZUM 85-1) Zumer V., Kokol P., Ojsteršek M.:
"Modeli podatkovno vodenih sistemov in ocena
zanesljivosti", IX Bosanskohercegovački
simpozium iz informatike "Jahorina 85",
zbornik radova, knjiga I, Sarajevo,
april 1985, pp. 168-1-8, - in slovene.

(ZUM 86) Zumer V.,Kokol P., Ojsteršek M., Zorman A.:
"Parallel Computer Architectures,
Programming Languages and Algorithms,"
Tehniška Fakulteta Maribor,
Maribor 1986, Technical Report - in slovene.