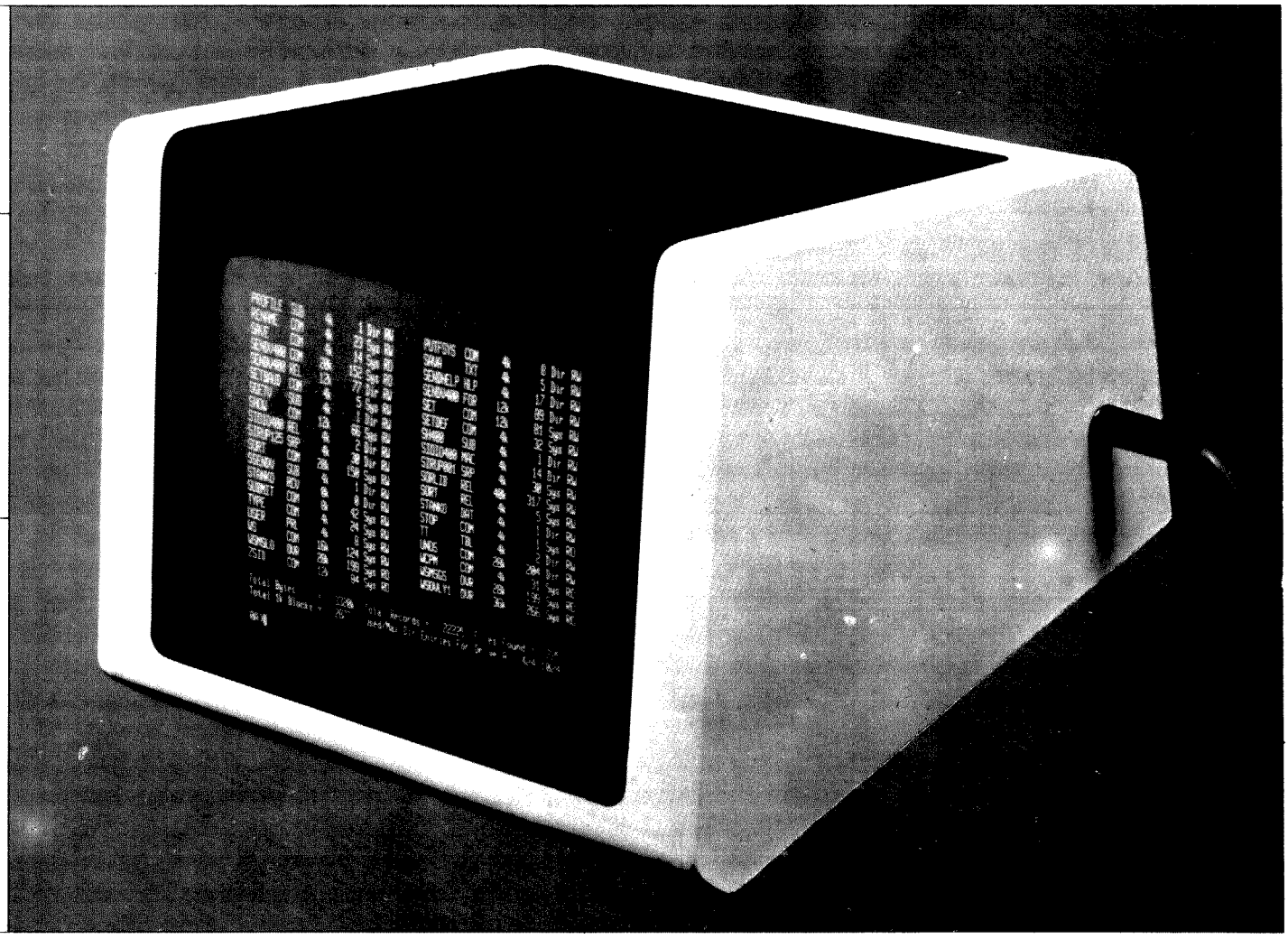
86 informatica 3

YU ISSN 0350-5596

SISTEM ZA ŠALTERSKO POSLOVANJE V BANKAH IN NA POŠTAH



računalniški sistemi delta®

Sistem za šaltersko poslovanje je sodobna računalniška oprema za delo v bankah in na poštah, opremljen z ustrezno programsko opremo.
Sistem omogoča samostojno ažurno poslovanje – od posameznih operativnih del na šalterjih do zajema podatkov za nadaljnjo obdelavo. Deluje lahko povsem samostojno ali v povezavi z glavnim računalnikom (prenos informacij je mogoč prek stalno najetih ali navadnih telefonskih linij). Delovanje sistema tudi ni odvisno od razpoložljivosti računalniških kapacitet glavnega računalnika.
Sistem nadomešča raznovrstno opremo, ki se uporablja pri šalterskem poslovanju – od klasičnih mehanografskih strojev, pisalnih strojev do kalkulatorjev in deloma mikročitalni-

Sistem za šaltersko poslovanje je savremena računarska oprema za rad u bankama i poštama, opremljen sa odgovarajućom programskom opremom.
Sistem omogućava samostalno ažurno poslovanje – od pojedinih operativnih poslova na šalterima do zahvata podataka za dalju obradu. Može da radi sasvim samostalno, ili da komunicira sa glavnim računarom (prenos informacija je moguć preko stalno iznajmljenih ili običnih telefonskih linija). Rad sistema je takođe nezavisan od raspoložljivosti računarskih kapaciteta glavnog računara.
Sistem zamenjuje raznovrstnu opremu, koja se upotrebljava u šalterskom poslovanju – od klasičnih mehanografskih mašina, pisaćih mašina do kalkulatora i delimično čitača

informatica

ČASOPIS ZA TEHNOLOGIJO RAČUNALNIŠTVA
IN PROBLEME INFORMATIKE
ČASOPIS ZA RAČUNARSKU TEHNOLOGIJU I
PROBLEME INFORMATIKE
SPISANIE ZA TEHNOLOGIJA NA SMETANJETO
I PROBLEMI OD OBLASTA NA INFORMATIKATA

YU ISSN 0350–5596

LETNIK 10, 1986 – ŠT. 3

Časopis izdaja Slovensko društvo Informatika, 61000 Ljubljana, Parmova 41, Jugoslavija

Uredništvo in uprava:

Informatica, Parmova 41, 61000 Ljubljana, telefon (061) 312 988; teleks 31366 YU Delta.

Letna naročnina za delovne organizacije znaša 5900 din, za zasebne naročnike 1590 din, za študente 490 din; posamezna številka 2000 din.

Številka žiro računa: 50101-678-51841

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije

Na podlagi mnenja Republiškega komiteja za informiranje št. 23-85, z dne 29. 1. 1986, je časopis oproščen temeljnega davka od prometa proizvodov

Tisk: Tiskarna Kresija, Ljubljana

Grafična oprema: Rasto Kirn

V S E B I N A

informatica

JOURNAL OF COMPUTING AND INFORMATICS

Published by Informatika, Slovene Society for Informatics, Parmova 41, 61000 Ljubljana, Yugoslavia

Headquarters:

Informatica, Parmova 41, 61000 Ljubljana, Yugoslavia. Phone: 61 31 29 88. Telex: 31366 yu delta

Annual Subscription Rate: US$ 22 for companies, and US$ 10 for individuals

Printed by: Tiskarna Kresija, Ljubljana

Design: Rasto Kirn

YU ISSN 0350–5596

VOLUME 10, 1986 – No. 3

C O N T E N T S

PARALLEL PROCESSING: ARCHITECTURE OF THE 1990'S (*)
(A SURVEY OF RECENT DEVELOPMENTS)

INFORMATICA 3/86

Peter Brajak
Inst. „J. Štefan", Ljubljana

UDK: 681.3:519.7

ABSTRACT - Two observations regarding the evolution of computer systems have, during the past decade or so, become so commonplace. First, the cost of digital hardware has dropped to the point that processors need no longer be considered a scarce resources. Second, the cost of computer software is increasing, both in absolute terms and in comparison with the cost of hardware on which it executes. The availability of large numbers of inexpensive processing elements quite naturally suggest the possibility of constructing highly concurrent systems capable of very rapid executions.

This paper briefly surveys recent results in the field of parallel processing. The results show that the rapid progress has been achieved and that the parallel processing can yield high performance. They also show that the major research issues still remain to be addressed.

1. Introduction

We are entering the era of fundamental changes in the field of computer architecture. In particular, a large part of these changes is represented by a transition from the serial to parallel processing. This transition is stimulated by at least four reasons:

1) the cost of digital hardware has dropped to the point that processors need not be considered a scarce resource

2) improvement of performance of monoprocessors is technologically limited,

3) the scope of problems whose algorithmic complexity exceeds the capacity of the today most powerful computers is very large,

4) the formulations of these problems naturally suggest their parallel realization.

Parallel processing can not guarantee to produce all desired performance levels in speed, reliability, accuracy, availability etc. Its limits are far from well known yet. On the other hand, it is certain that parallel processing requires large changes as regards:

1) ways of conceiving basic algorithmic and programming problems,

2) developing new software tools (languages, compilers, operating systems, communication mechanisms etc.),

3) developing non Von Neuman computer architectures.

For these reasons the basical question is whether parallel processing can reach high levels of the desired performance. In other words, whether it is worth such changes.

This paper tries to explain certain problems regarding these changes and tries to present some of the research work that has been accumulated in recent years. Primarily, the paper is concerned with the problems due to changes in developing new parallel architectures.

First, we are presenting a short discussion of measures and metrics used in evaluating the performance of the parallel systems. Afterwards, we will give an overview of some characteristics of the most typical examples.

(*) originally presented as an invited paper at MIPRO86, Symposium on New Generation of Computers, section Comp. Arch., Opatija, May 14-16 1986

2. Taxonomies

It is safe to say that all computers made today use concurrency to improve their speed. This parallelism is manifested on several levels:

1) inside the arithmetical and logical units:
   - transfer by words rathar than bits,
   - I/E overlaps,
   - Carry Look Ahead, etc.,

2) inside the central processor
   - pipelining,
   - multi functional units,

3) inside a monoprocessor
   - I/O,
   - cache,
   - interleaving, ettc.,

4) and principally - inside multiprocessor system.

We think that there are no fully developed taxonomies that can cover all characteristics of parallel computer architectures. The widely used one based on SIMD, MIMD modes of operation is not appropriate, since it does not show diversity of algorithmic and application features that modern computers use in order to explote concurrency. Therefore, we show three different taxonomies, first the one based on modes of operations, second the one based on applications they usually run and third the one based on types of algorithms they exploit.

Based on the mode of operation parallel computer could be in one of the following groups:

1) VECTOR PROCESSORS
   - vector supercomputers
     - CDC STAR-100
     - TI ASC
     - Cray-1,Cray-2,Cray-XMP
     - Burroughs BSP
     - Cyber 205
     - Fujicu VP100, VP200
     - Hitachi 8810, 8820
   - attached vector processors.
     - FPS 164
     - CSFI MAP6430

2) MULTIPROCESSORS
   - highly parallel multiprocessors
     - NYU ULTRA
     - CHoPP
     - IBM RP3
     - CEDAR
   - microprocessor based multiprocessors
     - BBN butterfly
     - INTEL iPS
     - CalTech HyperCube

3) SPECIAL PURPOSE PARALLEL PROCESSORS
   - sistolic
   - comparative-associative
   - reconfigurable
   - language based
     - Erlangen EMSY85
     - Connection Machine
     - TRAC
     - NON-VON, DADO
     - MPP
     - LISP mafine
     - CHIP

4) DATA FLOW
   - MIT Data Flow
   - Manchester Data Flow
   - SIGMA-1
   - TI DDP
   - DDM1

Based on type of algorithms they execute parallel processor could be explicit or implicit. This division comes from the definition of an algorithm as a partial order of operations, where partial orders are determined by the data dependencies between operations. A parallel algorithm can therefore be:

1) explicit: if data dependancies are well-defined, known a priori,

2) implicit: if data dependence is not a priori known, and must be determined through data analyses.

Based on type of applications, parallel systems can be divided in two groups:

1) supercomputers:
   - numerical applications,

2) fifth generation computers:
   - artificial intelligence,
   - expert systems,
   - relational data bases,
   - computer vision,
   - speech recognition.

3. Metrics

It can be affirmed that there are no good metrics that measure the performance of the system as a function of parallelism and offer at the same time some degree of comparability between different systems.

Common speed metric, MIPS (million instructions per second) fully defines the speed of the single processor systems and enables comparability between them. Furthermore, it is possible, knowing the

speed (rate) of the system, and the number of instructions in the program to predict a priori the execution time of new programs.

MIPS, however, is inadequate for vector type of parallel systems that operate in single instruction multiple data mode of operation. They achieve concurrency performing a number of different operations with a single instruction.

Slightly better, though still far from good metrics are MFLOPS (million floating point operations per second) and MLOPS (million logical operations per second). MFLOPS is used as speed metric for supercomputers while MLOPS is preferred with the fifth generation computers. The problem with these units is that they provide valuable results only for computations with dominant arithmetical or logical operations.

Another problem is that it is not possible to define a priori the speed of new, unfamiliar programs. In other words, it is impossible to say that the machine with X MFLOPS rate is going to solve a problem in the M/X seconds knowing that the specific problem has M number of computer instructions.

For multiprocessors, data flow machines and other parallel systems that operate asynchronously on the given sets of data, the problem is even harder. MFLOPS is not an adequate metric since these systems do not exploit concurrency based on one instruction multiple data principle. MIPS, on the other side is inadequate since it does not take into account the degree of concurrency.

Very often we use terms such as speedup and efficiency as the metrics that measure the performance as a function of parallelism/1/.

Let

T(1) = the time required to execute on a single processor,

T(p) = the time required to execute the same job using p processors,

then

$$\text{Speedup: } U(p) = \frac{T(1)}{T(p)}$$

and

$$\text{Efficiency: } E(p) = \frac{U(p)}{p} = \frac{T(1)}{pT(p)}.$$

Another, very useful definition of efficiency is

$$\text{Efficiency: } E(p) = \frac{A}{p}$$

where A represents the number of usefully active processors.

This definition gives a notion of concurrency, that is, a quality of a given algorithm.

Multiprocessors can, at best, reach a linear speedup. That is, N of processors can be at best 1/N time the speed of the monoprocessor. Among the factors that effect the non linear speedup are: synchronization, algorithm serialization, inter-processor communications, common resources, input/output etc./2/.

4. Vector computers

The vector computing is the most frequent form of parallelism with modern computers. The reason simply is the comparatively simple architecture which enables a simultaneous processing of only the regularly ordered vector elements. Furthermore, the additional price of the vector processor compared to the already existing sequential one is relatively small.

Although the vector computers do not belong to real parallel systems they have to be included in each grouping of such systems since they are the only ones commercially used for quite some period of time. For these, the largest number of sample cases have been written, new algorithms developed and comparisons made. Such, they are of high importance for the development of future systems.

Vector processing is based on employing two aspects of parallelism: on pipelining/4/ and on employing multi functional units/5/. In pipelining the speed increase is obtained through instruction overlapping. Mostly it is based on execution of a single instruction on vector elements that uniformly stream through the pipeline. The major characteristics are:

- synchronization is simple: based on data structure of operands; once all operands are in order and pipe activated, relative order of results is assured,
- the results come out of the pipeline by the speed determined by the speed of its slowest segment,
- increasing the number of segments, the total pipe speed is increased as well,
- better for operations on long vectors (relatively smaller start-up time),
- degradation is enlarged with jumps interrupts, scalar operations,
- pipelinig can be successfully and easily applied to arithmetic operations.

In vector computers with multi functional units each unit independently executes predetermined operations simultaneously with respect to other units. Most frequently vector computers use units for addition, multiplication, addresses generation, normalization etc. Multi functional units have the following characteristics:

- synhronization is more complicated: based on data availability,
- instruction dependency is usually resolved using additional coordinating unit,
- there must be a mechanism for instruction queuing: stack,

- technique is better for problems that can be organized in short fast instruction loops.

The largest problem with vector computers is not the complexity of hardware, but the algorithms that can be found embodying parallel code segments. The Amdahl's law theoretically shows the potential degradation of vector computer operations; the experience shows how hard it is to adapt algorithms to vector computers /6,7/. The dynamics nature of non numerical programs: data dependancy, jumps, interrupts break the regular data flow through vector units and so reduce the parallelism. This is evident from the following table which displays speeds of fastest contemporary vector computers on samples of 14 test programs from Lawrence Livermore National Laboratory/8/.

Cray 2 Civic comp June 1985	FACOM VP-50 V10L20 Aug 1985	Siemens VP-200 V10L10 May 1985	FACOM VP-400 V10L20 Aug 1985	HITAC S810/20 HAP V02-30 March 1986	NEC SX-2 Fortran SX Rev 022 March 1986
Mflop/s	Mflop/s	Mflop/s	Mflop/s	Mflop/s	Mflop/s
[illegible]	[illegible]	[illegible]	[illegible]	156.3	783.3
64.5	94.[illegible]	[illegible]	[illegible]	259.1	423.7
[illegible]	[illegible]	[illegible]	[illegible]	216.9	360.6
3.8	63.2	98.1	106.1	66.7	124.4
6.7	10.7	10.0	10.5	2.4	11.1
4.4	10.2	9.5	10.5	3.2	13.1
360.0	[illegible]	420.0	492.4	290.9	827.7
10.4	56.0	40.4	46.6	8.8	156.8
[illegible]	[illegible]	257.4	231.4	263.2	315.6
47.4	52.6	54.8	[illegible]	67.1	98.0
2.0	5.1	4.8	5.1	9.9	24.8
52.3	42.7	14.3	156.8	110.8	276.5
2.6	6.5	6.2	7.0	5.0	8.7
5.8	11.8	13.8	15.1	8.5	24.2
47.[illegible]	55.[illegible]	131.5	166.1	104.9	289.2
7.3	12.8	19.7	21.5	[illegible]	53.[illegible]

Amdahl's law says that when a computation is performed in two modes of operation the net speed is mostly determined by the computation speed of the slower mode.

if:

- f - percentage of scalar operation
- p - vector processor quality: pipe speed related to the scalar one,

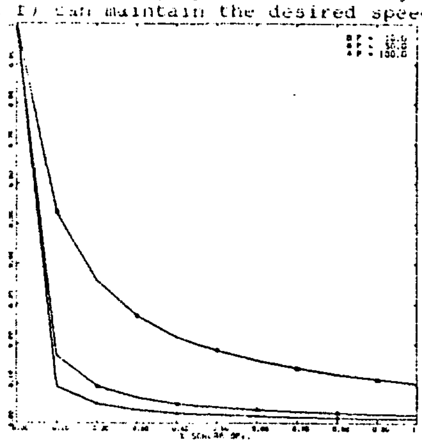
then

$$T(p) = f*T(1) + (1-f)*T(1)/p$$

that is

$$Speedup(p) = 1/(f+(1-f)/p)$$

From the displayed diagram it is evident that for vector computers the quality of algorithms is of utter importance and that only the highly paralellized algorithms (small f) can maintain the desired speed.



Vector computers are still today the fastest computers, but not due to their parallel capabilities but due to the fastest components installed (memory, scalar processor).

5. Multiprocessors

In the last few years a large progress was achieved in the field of parallel processing as concerns multiprocessor systems. Several years ago multiprocessors were only a topic of academic research works (C.mmp, CM*, Pluribus, HEP/10,11,12,13/) while today undoubtedly there are several hundreds of smaller and larger systems based on multi processing architectural concepts/14/. Enthusiasm arises from the facts that:

- speed can be obtained by a larger number of smaller, cheaper processors,
- speed of vector and other SIMD computers is bounded by percentage of scalar operations,
- special purpose computers are not general and cannot can be used on larger class of problems.

Most of today available multiprocessors are neither in theoretical concepts nor in practical performance level characteristics that should be expected from a truly multiprocessor system. These machines have not obtained the desired results due to:

- memory contentions arising as each task accesses common memory,
- too much time is required to synchronize the exchange of data between tasks,
- the scheduling of tasks is time consuming,
- the algorithms shoud be well developed that would enable the activity of all processors.

The ultimate multiprocessor based on Schwartz's "Paracomputer" /15/ would be a general purpouse computer, fast as well as any special purpouse parallel computers, with unlimited number of processors and conflict free access to the common memory. Such systems are presently the subject of reasearch work with two dominant academic projects (CHOPP, ULTRA)/16,17/. Most other commercially available multiprocessors are based mainly on primitive synchronization and communication mechanisms with networks that do not support a large number of processors.

Different from vector processors where architecture is brought to its optimum level, multiprocessors still have no defined architectural standards. We can say that there are as many different concepts and solutions as there are multiprocessor systems. There are two basic questions to be put regarding this topic:

- what processor connection is most convenient for a certain number of processors?
- are the systems with a processor access to the common memory "better" than those with message passing and local memories?

5.1 Processor connections

The answer to the first question is well known and can be obtained by a simplified calculation of network quality which is defined as a quotient of its performance and price.

performance = P/T

price = V + E

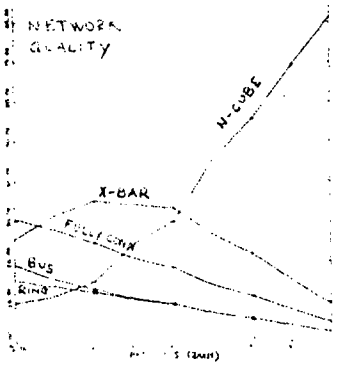
where

P - number of messages in the network
T - round trip delay
V - number of links
E - number of network switches

NETWORK	MESSAGES	DELAY	PERFORMANSE
bus	1	2	1/2
ring	N	N	1
n-cube	N/2	2logN	N/(4logN)
X-bar	N	4	N/4
f.conn.	N	2	N/2

NETWORK	LINKS	SWITCHES	PRICE
bus	1	N	N+1
ring	N	N	2N
n-cube	N+NlogN	NlogN/2	N+3NlogN/2
X-bar	2N	NN/4	NN/4+2N
f. conn.	(NN-N)/2	NN	N+(NN-N)/2

NETWORK	QUALITY
bus	1/(2N+2)
ring	1/2N
n-cube	1/(4+6logN)
cross-bar	1/(N/2+4)
fully conn.	1/(N+1)



From the quality diagram we can see what interprocessor communication is the most suitable one for a given number of processors. For a small number of processors (<=4) it is most convenient if all processors are fully connected; for up to 64 processors the most convenient is to connect processors using cross-bar switch; and for those above 64 processors the best quality show the permutation networks such as n-binary cube.

5.2 Common vs. Local Memory

The choice between common or local memory system is the most challenging task and depends on price, generality, speed, application and number of processors an architect sets while choosing a design for his multiprocessor. Schwartz /15/ showed that in applications where thousands of processors cooperatively solve large problems, the systems supporting high speed concurrent access to the common memory are more efficient and less complicated for programming. For a programer it is very important that the interprocesor network geometry is hidden and that each processor independently performs message routing along the network. In systems with local memories, routing is needed on the programming level. Undoubtedly, the increasing number of processors mounts the price of common memory systems while the quality falls because of memory requirements collision. This can be observed from the flooding of multiprocessors developed on Hypercube principle /18/, a network that supports message passing communication between local memories. It has a very simple geometry: in a m-processor system each processor has its number from 0 to m-1. The I processor is connected to log(m) processors whose number differs from I in only one bit of its binary representation.

The proof that the increasing of processor number does not cause an increase of collisions in common memory systems is the "Conflict Free Memory" - U.S. Patent 254583/19/ which proves that the access to the common memory does not depend on the processor number and that synchronization can be performed using network switching elements. It's a very original idea, apllied in CHOPP, in ULTRA and in some other projects /20/. It is characterized by the following features:

- all processors can read any memory location simultaneously,
- all processors can read the SAME memory location simultaneously,
- all processors can write into the same location simultaneously.

This can be achieved using "intelligent" network switches, so that they intercept simultaneous requests for a single memory location and let only one request be present at the memory bank containing that location. Each node in the network must maintain a list of memory requests that have passed through the node to avoid sending out more than one request for a single memory location.

However, multiple accesses to different locations in the same memory bank still conflict at that particular bank. Conflicts

can be avoided by a statistical interleaving of memory locations and by program specified interleaving of code. Each processor must implement a fixed , one to one, onto mapping of memory requests to the physical addresses in the N memory banks. The mapping must be fixed and random.

There is another requirement for an optimal use of such system. Programs must be presented to the processor in terms of tasks. In order to garantee full processor utilization there must be sufficient number of tasks active at all the times to overcome the latency of the interconnection network.

It is evident that in the near future we can not expect commercially available systems that would support the above mentioned concepts for a large number of processors. Not only because of weight on the architectural level (expensive and complicated switches) but because of the software equipment that stagnates considerably. However, these concepts, plus the solution for software synchronization, could be used as standards we should follow in building general purpose, highly parallel multiprocessors.

6. Special Purpose Parallel Systems

A very popular way to increase speed is to optimize the communication structure among processors so that it corresponds to specific problems that should be performed on it. The idea flourished in the late 70's with the development of VLSI technology /21/ and the belief that communication is the basic factor in the price of parallel processing. Such systems, for the price of speciality give optimal solutions in performance levels of the parallel computations.

Sistolic processors /22/ are examples of processors based on neighbour to neighbour principles. Communicationally, they are optimized for solving specific numerical problems. A large number of mathematical operations, like matrics operations, linear equations solutions, FFT can be hardwared to a corresponding processor geometry, giving almost theoretically optimal results /23/. Sistolic processes are time synchronized and operate on the principle of overlapping (like in pipelining) with difference that all processors perform identical functions. Data elements stream regularly from one neighbouring processor to another. Such flow through processors is constant, without interrupts and optimal since all processors are active.

From this technology more complicated and general architectures were developed which are often used in fields of artificial intelligence, relational databases, computer vision, and in some other non-numerical applications. Simply saying, 5th generation computer architectures had emerged, based on the ideas from sistolic processor (putting algorithm on interprocessor geometry) and ideas from the 60-ies when first associative computers were developed: PEPE, STARAN, ALAP, ECAM /25/. The idea is that memory values are not referenced by address but by a comparison of the very value (Content Addressed Memory - CAD).

A specific parallel system for production (expert) systems could serve as an example of such a system. It is based on the binary tree interprocessor geometry, which is ideal for searching problems.

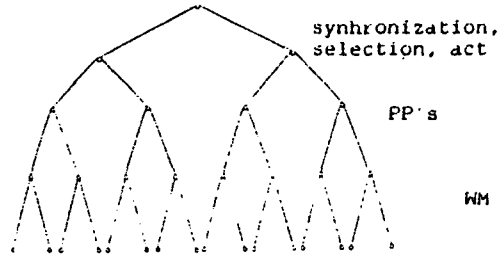
As we know, the production system is defined as a set of rules, productions, and a database, called working memory (WM). Each production consists of left hand side (LHS) and right hand side (RHS):

```
( ( =x ON = y ) (- =z ON =x)
        ==>  (- =x ON =y) (=x ON =z) )
```

The following cycle of operation is repeatedly executed:

1. Match: for each rule, determine whether the LHS matches the current environment of WM,
2. Select: choose one of the matching rules according to some predefined criterion,
3. Act: add or delete from WM all assertions specified in the RHS of the selected productions.

Parallelism can be achieved at all three steps, especially at the first one where the need for a CAM system is stressed. The idea is that each production is stored on one processor element of a binary tree and all processors that are subtrees to this processor are part of its WM. Example of a functional distribution in a simplified system (4 productions only) would be the following:



Parallelism of this system is manifested so that LHS of each production could be simultaneously compared to the content of the corresponding WM. Each production processor (PP) sends to its subtree pattern elements of LHS. After KlogM steps where K is the number of elements of LHS and RHS and M is the number of WM processors, all PPs that matched LHS obtain the appropriate elements of the RHS. A selective step chooses one of the RHS in 2logN steps, where N is the number of productions (a way to the top of the tree and back to PP). The third step deletes or adds the RHS of the chosen production to each WM in KlogM steps.

Similar systems could be applied for relational data bases, expert systems, that is, for all forms of knowledge based retreivals /26,27,28,29/.

A very interesting concept of specialized

parallel systems is, by no doubt, the idea of reconfigurably connected processors. The primary idea was initiated by CHiP project /30,31/ and nowadays lots of researchers work in this field /32,33,34/. It is a network of processors connected with switches such that before the program execution each switch performs a short program which defines the form of connectivity among the processors. As an example, if a data searching algorithm is concerned, processors configurate themselves in a binary tree geometry; if the matrix multiplication is concerned, processors are connected as a hexagonal array.

The power of these systems is that it is possible to achieve the generality of multiprocessors and the speed of specialized parallel systems. Problems encountered by such systems are similar to those encountered by multiprocessors:

- complicated synchronization,
- information propagation time is not equal for all processors,
- memory conflicts.

7. "DATA FLOW" computers

Data flow systems have the potential to become the most prospereous arhitecture of parallel processing systems. What makes them so attractive is that:

- they perform data dependency analysis at run time, thus representing convenient way of exploiting implicit parallelism,
- the number of processors can be increased without changing applications code,
- computation is deterministic,
- parallelization is automatic.

The first and the third of these points are particularly appealing because on many parallel systems computation is not necessarily repeatable. Since implicit parallelism includes explicit parallelism, dataflow is considered the most general operational principle for parallel processing. Although dataflow arhitectures enjoy popular support among scientists, their greatest disandvantage is the considerable large overhead that comes from the self-synhronizing manner of their parallel operations. It is especially evident in applications where the parallelism is primarly inherent in the data structure objects, or in simple repeatable concepts. This is the reason why relatively few systems are operational, and relatively little experience available.

8. Conclusion

There are numerous encouraging results that prove that the rapid progress is being made in the field of parallel processing. In this paper we give a broad overwiev of some of the most important classes of parallel processors, we show their positive and negative aspects, future trends and possible difficulties.

Experiments and experience suggest the following:

1) significant gain in speed can be achievied using concepts of parallel processing, even for systems with small number of processors,
2) a broad spectrum of applications can be applied to parallel processing,
3) simple structures could be adequate to support parallel computation (this is true even for highly parallel systems).

These are the encouraging results, which however, should not diminish major research problems that remain to be addressed:

1) software and software tools:
   - decomposition and analysis of parallel formulation of algorithms,
   - debugging,
   - data dependency,
2) system problems:
   - I/O,
   - interactive graphics,
   - local networks,
3) highly parallel arhitectures:
   - measure and performance of highly parallel systems with thousands of processors.

ACKNOWLEDGMENT

Author wishes to thank prof. A. Zeleznikar for many useful comments. Without him this work would not even be conceived.

10. References

/1/ Buzbee B.L., Applications of MIMD Machines, Proc. in Vector and Parallel Processors in Comp. Sci., Oxford 1984

/2/ Jones A., Schwarz, Experience Using Multiprocessor Systems-A Status Report, ACM Surveys, June 1980

/3/ Amdahl G., The Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, AFIPS 1967

/4/ Kogge P., The Arhitecture of Pipelined Computers, 1981

/5/ J.E. Thornton, Parallel operation in the CDC 6600, AFIPS 1964

/6/ D.J. Kuck, Parallel Processing of Ordinary Programs, Advances in Computers, Academic Press, 1976

/7/ Rodrigue G., Giroux D., Pratt M., Perspective on Large Scale Scientific Computation, IEEE Computer, Oct. 1980

/8/ Steen A., Results on the Livermore Loops on some new supercomputers, Supercomputers, March 1986

/9/ Hollenberg J. The Cray-2 Computer System, Supercomputer, 8/9 1985

/10/ Wulf W. A., Bell C. G., C.mmp: A Multi-Mini-Processor, AFIPS 1972

/11/ Katsuki at. all, Pluribus-An Operational Fault Tolerant Multiprocessor, Proc. IEEE, 1978

/12/ Swan R.J, Fuller S.H., Sieworek D.P., CM*: A Modular Multiprocessor, AFIPS Conf. Proc 46, 1977

/13/ Snelling D. HEP applications, Proc. in Vector and Parallel Processors in Comp. Sci., Oxford 1984

/14/ Lineback R., Parallel Processing: Why a Shakeout Nears, Electronics, 1985

/15/ Schwartz T. J, Ultracomputer, ACM Trans. of Prog. Lang. 1980

/16/ A Gottlieb at. all, The NYU ULTRAcomputer - *Designing an MIMD Shared Memory Parallel* Computer, Trans. on Comp., 1983

/17/ Sullivan H, Bashkow T, A Large Scale Homogeneous, Fully Distributed Parallel Machine, 4-th Ann. Sym. on Comp., 1977

/18/ Seitz C. The Cosmic Cube, Communications of ACM, 1/85

/19/ Sullivan H., Bashkow T. Cohn L., private communications

/20/ Brajak P., A Design of a Fully Distributed, Parallel System With a Common Memory, Jahorina 1985

/21/ Mead C. Conwey L., Introduction to VLSI Systems, 1980

/22/ H.T. Kung, C.E. Leiserson, Systolic Arrays (for VLSI), Sparse Matrix Proc., 1978

/23/ H.T. Kung, The structure of Parallel Algorithms, Dept. of Comp. Sci., Carnegie-Mellon U., 1979

/24/ Lee C. Y., Paul M. C., A Content Addressable Distributed Logic Memory with Applications to Information Retrieval, Proc of IEEE, June 1964

/25/ Yau S.S., Fung H.S., Associative Processor Architecture A Survey, ACM Syrveys, 9/1977

/26/ Shaw D., NON-VON Supercomputer, Dept. of Comp. Sci., Columbia University 1983

/27/ Stolfo S., DADO AI machine, Dept. of Comp. Sci., Columbia University 1983

/28/ Fahlman S. The Hashnet Interconnection Structure, Dept. of Comp. Sci., Carnagie-Mellon U., 1979

/29/ Forgy C., Note on Production Systems and Illiac IV, Dept. of Comp. Sci., Carnagie-Mellon U., 1980

/30/ Snyder H, Programming Processor Interconnection Structures, Dept. of Comp. Sci. Perdue U., 1981

/31/ Snyder H, Intro. to the Configurable Highly Parallel Computer, IEEE Computer, 1982

/32/ S.P. Kartashev, S.I. Kartashev, Efficient Internode Communications in Reconfigurable Binary Tree, IEEE Trans. on Comp. 1984

/33/ S. Yalamanchili, K. Aggarwall, Reconfiguration Strategies for Parallel Arhitectures, IEEE Computer 1985

/34/ D. Hillis, The Connection Machine, MIT Press, 1986

Petar Brajak was born in Rijeka, Yugoslavia on June 19, 1957. He receved the B.Sc. and M.Sc. in computer science form Columbia University, 1981 and 1983 respetevely, where he worked on two parallel processing projects: NON-VON and CHoPP.
From 1984 he works at the Reactor Centre of "J. Stefan" Institute, Ljubljana. His work includes instalation of large programs on vector processor CSPI MAP6410, and developing software tools for vectorization of serial programs.
His research interests are in the field of higly parallel multiprocessor systems, especialy synchronization protocols on software levels.

ON CHOOSING A PLAN FOR THE EXECUTION OF DATA FLOW PROGRAM GRAPH

INFORMATICA 3/86

Borut Robič, Jurij Šilc
Jožef Stefan Institute, Ljubljana

UDK: 681.519.7

ABSTRACT - In the paper we present a generalized analisys of static data flow program graphs. These graphs are allowed to have nodes that use more than one unit of time for their execution. Such graphs are more realistic then graphs with nodes that execute in one unit of time. We restrict our consideration to graphs with integer execution times of their nodes. In the paper we first briefly describe the data flow concept of computation. Next we describe the basic data flow architecture and a common way of the execution of a graph on it. We show that this way of the execution has a drawback. In the next sections we introduce the notion of a static data flow program graph and describe the state of the execution of such graph. The state consists of a few time depending sets of nodes. We define a plan for the execution of a program graph which is the result of the analysis of the graph made before its execution. There are two special plans for the execution. Information, obtained by these two plans is used for defining the third special plan, which we call the heuristic plan for the graph execution. The execution of a graph according to this plan may minimize the number of processors needed, without lengthening the total execution time of the graph. Finally, we informally describe the algorithm for obtaining plans for the execution.

O IZBIRANJU NAČRTA ZA IZVRŠEVANJE PODATKOVNO PRETOKOVNIH GRAFOV - V delu je podana posplošena analiza statičnih podatkovno pretokovnih grafov. Točke takšnih programskih grafov se lahko izvršujejo poljubno celo število časovnih enot. Uvodoma je opisan koncept podatkovno pretokovnega računanja. Sledi opis značilne podatkovno pretokovne arhitekture ter enega izmed možnih izvrševanj podatkovno pretokovnega grafa na njej. Prikazana je slabost takšnega načina izvrševanja. Po opisu statičnega programskega grafa sledijo definicije množic točk, ki sestavljajo stanje izvrševanja grafa. Definiran je načrt izvrševanja programskega grafa, ki je rezultat njegove predhodne analize. V splošnem obstaja več načrtov za izvrševanje. Izvrštve, ki ustrezajo posameznim načrtom, se razlikujejo po uporabljenem številu procesnih elementov in ne podaljšujejo minimalnega časa, potrebnega za izvršitev programskega grafa, v kolikor je na voljo dovolj procesnih elementov. Obstajata dva posebna načrta za t.i. takojšnje in leno izvrševanje, ki v splošnem ne minimizirata števila potrebnih procesnih elementov. Ker sistematično pregledovanje vseh možnih načrtov vodi v kombinantorično eksplozijo, je v delu predlagan hevristični postopek za izbiro načrta izvrševanja, ki teži k minimizaciji števila procesnih elementov.

1. INTRODUCTION

A data flow system comprises a user-oriented high-level language, a low-level base language, and a highly-parallel computer architecture. User programs are written in the high-level language and are translated into corresponding programs in the base language. A base language program is a graph composed of nodes interconnected via directed arcs. Each internal node is an operation and represents a separate processing elemet capable of accepting, processing and emitting value tokens travelling along the graph arcs. Each operation executes only when all tokens, carrying operands required by that operation have arrived. At that point the operands are consumed by the node and new tokens, carrying results, are placed on the output arcs [COM82]. This fundamental principle permits the graph to be mapped onto a computer architecture consisting of a very large number of independent processing elements and switches, able to connect any two processing elements, making a data path between them. Separate data paths can cross the switch simultaneously [KFG584]. For example see Fig.1.

2. SIMULATION OF DATA FLOW ARCHITECTURE

In general not all operations (nodes of a graph) need to be assigned to processing elements at the same moment since not all operations have available all input operands at that moment. To make the data flow concept possible even without a very large number of processing elements data flow computers are

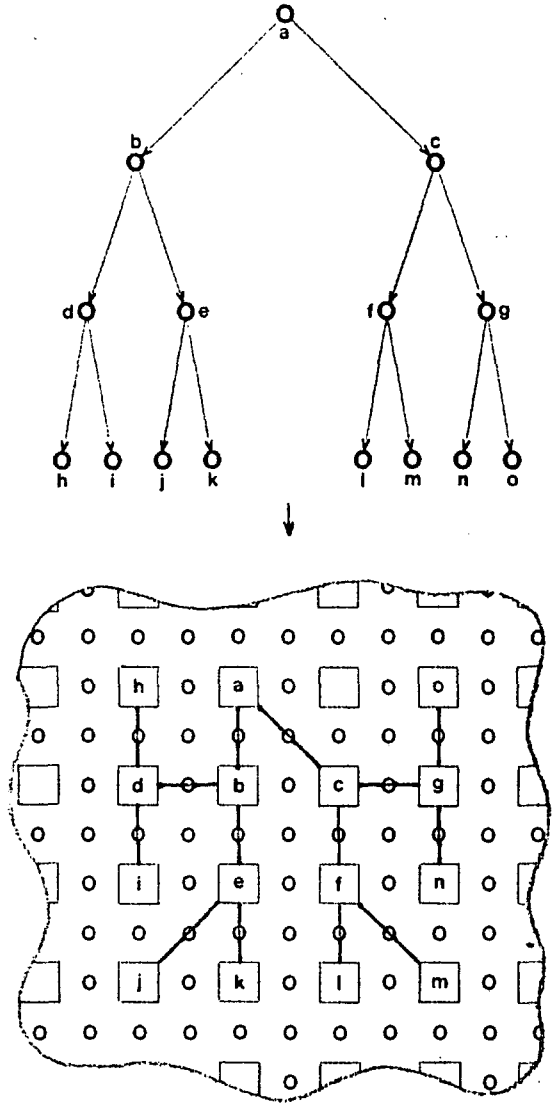


Fig.1: In the switch lattice the squares represent processing elements, circles represent switches, lines represent data paths.

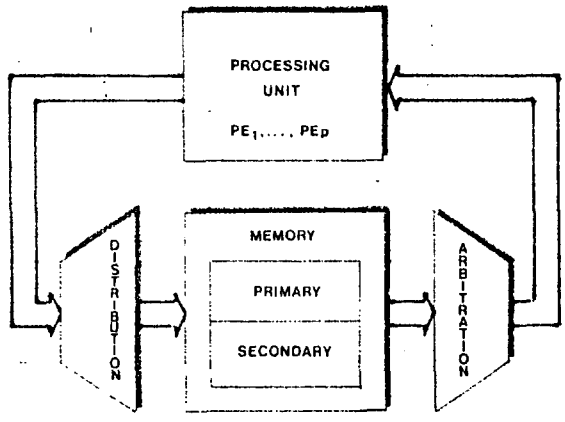


Fig.2: The model of data flow machine.

usually based on a packet communication machine organization, consisting of a circular instruction execution pipeline of resources. The resources are memory, arbitration, processing and distribution unit. The memory unit is divided into the primary and secondary part, the former beeing faster. The processing unit consists of a number of processing elements (Fig.2).

Nodes, having none of their input operands arrived reside in the secondary memory. These are noncreated nodes. At the moment when the first input operand has arrived the node is created, i.e. moved to the primary memory to wait for the other input operands. A created node becomes executable at the moment when the last input operand has arrived. Executable node is ready for the execution and may be transferred (fired) to the processing unit where a processing element starts to execute it. It is the arbitration unit that decides which of the nodes are executable and which processing elements are to be allocated to. The place in the memory unit which was occupied by that node is now set free. While the node is beeing executed the distribution unit finds out where the nodes waiting for the result are. When the result is produced, the distribution unit sends it to all nodes waiting for it, creating some of them if neccessary. The node that has been executed is now deleted.

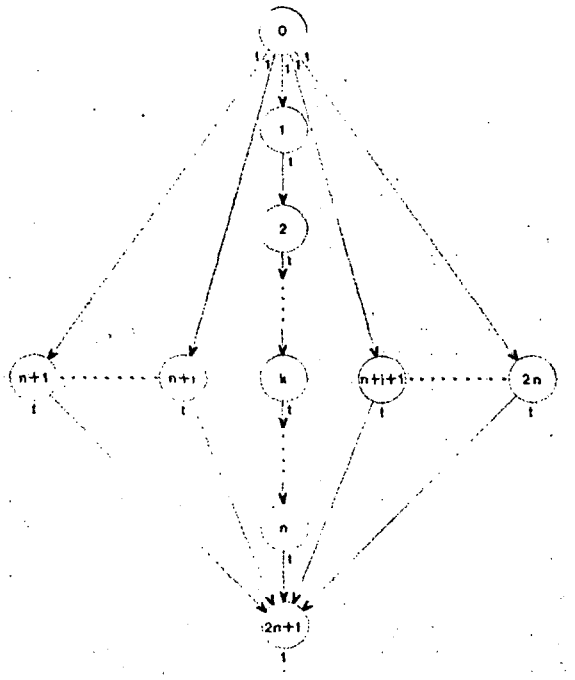
Such execution may need more processing elements than there are in the processing unit. The problem where there are more executable nodes than processing elements must be solved during the execution of a graph. All these nodes are executed in several steps implying the lengthening the total minimum execution time of a graph. Note that after each such step again the same problem may appear.

An anaysis of the graph before its execution may prevent the problem discussed above. The basic observation is that in general not all executable nodes have to be fired immediately, since some of them may wait a period of time in the memory without lengthening the total minimum execution time of a graph. Analysing the graph we obtain several plans for its execution, each execution having its own characteristic. Information obtainded by the plan is used by the architecture components during the actual execution of a graph in deciding which executable nodes should be retained. The execution according to a properly chosen plan may result in minimization of some resources needed, such as the number of processing elements. We point out three special plans for the execution. Execution according to the first of them is essentially the immediate execution, described above. The execution according to the second plan is the opposite of the immediate execution, while the execution according to the third plan often uses minimum number of processing elements. Executing graphs according to this plan we may avoid the problems discussed above.

3. STATIC DATA FLOW PROGRAM GRAPH

There are two ways of envisaging a data flow program: as a static or as a dynamic data flow program graph. We limit our discussion to static graphs. In short, static data flow program graphs are acyclic, while the dynamic are not.

We define a static data flow program graph G = (V,A), in further discussion program graph, to be a directed, acyclic, and simple (no multiple arcs of the same direction between two nodes) graph. The set V of nodes is partitioned into three disjoint sets V_S ,V_I , V_F of starting, internal and final nodes, respectively. The starting nodes have no input archs while the final nodes have no output archs. Furthermore, there must always exist a path to any internal or final node from some starting node and, similarly, a path from any starting or internal node to some final node. Starting nodes are used to carry the input values while the final nodes store the results. Internal nodes carry operations [Ro586]. The time of the execution of a node n is t_n, where t_n = 1 for all starting and final nodes n.



$\mu_{imm} = \mu_{lazy} = n+1 \qquad \mu_{heu} = 2$

Fig.3: Static data flow program graph.

The length of a path is defined to be the sum of the execution times of the nodes along the path. The path from a set of nodes A to a set of nodes B is defined to be any path that starts at some node of the set A and ends at some node of the set B. The longest path from the set A to the set B is a path that has among all the paths from the set A to the set B maximal length. The length of the longest path from the set A to the set B is described by l(A,B). When the set consists of only one element we substitute the set by its element. For example, $l(n,V_F)$ is the length of the longest path from the node n to the set of the final nodes. Note that since the program graph is acyclic by assumption, the lengths l(m,n) for any pair of nodes can easily be computed using one of well known algorithms [Law76]. The evaluation of l(A,B) is then trivial for any two disjoint sets of nodes A and B.

If a node n is beeing executed at the moment j we define $l'(n,V_F)$ to be the sum of the length of the longest path from the set of its sons to the set of the final nodes and, the time neccessary for the node n to finish its execution.

4. THE STATE OF THE GRAPH EXECUTION

The state of the execution of a graph at the moment j is the quintuple $\sigma_j = (N_j, C_j, E_j, F_j, D_j)$ where N_j ,C_j ,E_j ,F_j and D_j are the sets of noncreated, created, executable, executing and deleted nodes at the moment j, respectively. There are also few other sets used for computing the sets mentioned. The sets are described bellow:

- F_j^* ... (executed nodes) is the set of all the nodes fired before the moment j that have finished their execution at the moment j and put their results on all output arcs,
- C_j^{new} ... (new created nodes) is the set of all those nodes that received at the moment j the first input operand,
- C_j^{old} ... (old created nodes) is the set of all those nodes that had been created at any of the moments before the moment j yet did not fire until the moment j-1 including,
- C_j ... (created nodes) is the union of old and new created nodes,
- E_j^{new} ... (new executable nodes) is the set of all those nodes that have received the last input operand at the moment j,
- E_j^{old} ... (old executable nodes) is the set of all those nodes that had become executable before the moment j but have not fired until the moment j-1 including,
- E_j ... (executable nodes) is the union of the old and new executable nodes,
- E_j^* ... (unconditionally executable nodes) is the set of all those executable nodes that must be fired at the moment j so as not to lengthen the total execution time;
- E_j^v ... (conditionally executable nodes) is the set of all those executable nodes that are fired at the moment j although they could be fired later withot lengthening the total execution time of a graph,
- F_j^{new} ... (new executing nodes) is the set of all the nodes that started executing at the moment j,
- F_j^{old} ... (old executing nodes) is the set of all those nodes that had been fired before the moment j yet have not finished their excution till the moment j including,
- F_j ... (executing nodes) is the union of the old and new executing nodes,
- N_j ... (noncreated nodes) is the set of all those nodes that have not created till the moment j including,
- D_j ... (deleted nodes) is the set of all those nodes that executed till the moment j including,
- E_j^{cr} ... (critical nodes) is the set of all those nodes that allways become unconditionally executable at the same moment j regardless of the plan of the execution.

5. THE PLAN FOR THE EXECUTION

The plan of the execution is a supervisor that controls the execution of a program

graph. Consequently, the plan implyes a certain degree of control flow in data flow architecture and is realized by a time control vector associated to each node of a program graph.

The plan for the execution of a graph is determined by the rule which selects the sets E_j^v. There are two trivial plans for the execution of a graph. These are a plan for immediate and a plan for lazy execution. The plan for immediate execution is determined by choosing E_j^v to consists of all those executable nodes at the moment j that could be fired even later. The plan for the lazy execution is determined by forcing E_j^v to be empty at all moments j. Consequently, the immediate execution fires the nodes as soon as possible while the lazy execution defers firing to the last possible moment. In general, none of these two executions minimizes the number of processing elements needed. The plan for the execution that uses minimum number of processing elements could be found by sistematic examination of all possible rules for the choosing the sets E_j^v. Since we want to avoid the combinatorial explosion we try to find a heuristic rule such that the execution according to the associated heuristic plan would use the number of processing elements as close as possible to the theoretical lower bound. For example, execution of the graph on Fig.3 according to the immediate plan needs n+1 processing elements, since at the moment j = 1 the node 1 is fired simultaneously with the nodes n + i, 1≤i≤n. Similarly, the lazy plan implies the execution that involves n+1 processing elements, too. Namely, at the last step the node n is fired together with the nodes n + i, 1≤i≤n. On the other hand, the heuristic plan offers an execution with only two processing elements for at each step only the pair of nodes k, and n+k are beeing executed.

The plan for the heuristic execution will be discussed below.

6. THE TIME CONTROL VECTOR

Every node n is associated with a time control vector $\tau_n = (\tau_{C,n}, \tau_{E,n}, \tau_{F,n})$.

$\tau_{C,n}$, $\tau_{E,n}$ and $\tau_{F,n}$ are the moments when the node n becomes created, executable and fired. Time control vectors are computed for each node while the plan for the execution is beeing constructed. Since all the sets described above are affected by the rule for choosing the set E_j^v, the time control vectors depend on a plan constructed. Every plan for the execution has its own set of time control vectors. To point out that the time control vector of a node n is computed according to the plan for immediate or lazy execution we write τ_n^{imm} or τ_n^{lazy}, respectively.

7. THE PLAN FOR HEURISTIC EXECUTION

7.1. CRITICAL NODES

Let $\tau_n^{imm} = (\tau_{C,n}^{imm}, \tau_{E,n}^{imm}, \tau_{F,n}^{imm})$ be time control vectors of a node n according to immediate and $\tau_n^{lazy} = (\tau_{C,n}^{lazy}, \tau_{E,n}^{lazy}, \tau_{F,n}^{lazy})$ time control vectors of a node n according to lazy execution. Then we say that an internal node n is critical if $\tau_{F,n}^{imm} = \tau_{F,n}^{lazy} = \tau_{F,n}^{cr}$.

THEOREM 1.

Every critical node n is fired at the moment $\tau_{F,n}^{cr}$, regardless of the plan of the execution.

PROOF: $\tau_{F,n}^{imm}$ is the first possible moment when the node n can be fired, regardless of the plan of the execution of a program graph. On the other hand, $\tau_{F,n}^{lazy}$ is the last moment when the node can be fired without lengthening the total execution time, taken over all possible plans of the execution. Consequently, if n is a critical node, then $\tau_{F,n}^{imm} = \tau_{F,n}^{lazy} = \tau_{F,n}^{cr}$ which means, that the moment $\tau_{F,n}^{cr}$ is the only moment when the node n can be fired in all possible plans of the executions. Q.E.D.

LEMMA 1.

There is at least one critical node in the program graph.

PROOF: Let be p the longest path from the set of starting nodes to the set of final nodes. Then, in any plan of the execution the son of starting node on the path p must be fired at the moment j=1 so as not to lengther the total execution time. Consequently, the son of the starting node is critical. Q.E.D.

THEOREM 2.

A node is critical if and only if it is on a longest path from the set of starting nodes to the set of final nodes.

PROOF: Let p be some longest path from the set V_S to the set V_F. By Lemma 1 the son of the starting node on the path p is critical. Now suppose that all first $k > 0$ internal nodes $n_1, n_2, \ldots, n_k$ on the path p are critical. Consequently, the first moment at which the node n_{k+1} can be fired is the moment $1+\Sigma_{i=1}^{k} t_{n_i}$. Since the node n_{k+1} is on the longest path from the set V_S to the set V_F this is also the last moment, when it can be fired, so as not to lengthen the total execution time of the graph implying that the node n_{k+1} is critical. Conversely, suppose a node n is critical. Consider the longest path p of the all paths from V_S to V_F, passing the node n. We define the subpath p' of the path p to be the path consisting of all the nodes from V_S to the father of the node n. Similarly, the subpath p" of the path p consists of all the nodes from the son of the node n to the set V_F. We define $l(p')$ and $l(p")$ to be the lengths of the paths p' and p", respectively. Note, that since p is the longest path from V_S to V_F through the node n, the path p' must be the longest of the paths from V_S to the set of fathers of n. Similarly, the path p" must be the longest path from the set of sons of the node n to the V_F. Suppose the relation $l(p')+t_n+l(p") < l(V_S, V_F)$ holds. Then there must be some node m on the path p, that can be fired at at least two different moments. The node m must be either on the path p' or on the path p", since the node n is critical, by assumption. Were the node m on the path p' the node n could be fired at

at least two different moments contradicting the assumption that it is critical. If the node m were on the path p", a similar argument would result. Consequently, relation $l(p')+t_n+l(p'') = l(V_S, V_F)$ is true, implying that the path p is one of the longest paths from V_S to V_F. Q.E.D.

Let E_j^{cr} be the set of all critical internal nodes at the moment j. We call the value

$\zeta = \max |E_j^{cr}|$, where $1 \le j \le t^*$, the maximal critical parallelism of a program graph, where $t^* = l(V_S, V_F) - 1$.

Constructing the longest paths from V_S to V_F [Law76] and considering their nodes, the critical internal nodes are easily computed. We could determine the critical nodes also by computing the time control vectors associated to the plan for the immediate and lazy execution and comparing their third components, for each internal node.

7.2. UPPER AND LOWER BOUNDS ON μ_{min}

Let us define μ_{min} to be the minimum number of processing elements needed for the execution of a program graph, taken over all possible plans of the execution. We want to bound μ_{min} as much as possible.

Let μ_{imm} and μ_{lazy} denote the number of processing elements needed in the immediate and lazy execution of a program graph, respectively. We define the average parallelism of a program graph to be $\pi = t_{seq}/t^*$, where $t_{seq} = \sum_n t_n$, $n \in V_I$, and maximal parallelism to be $\lceil \pi \rceil$*. We also define $\alpha = \max(\lceil \pi \rceil, \zeta)$, and $\beta = \min(\mu_{imm}, \mu_{lazy})$. Then the following theorem illustrates the upper and lower bounds on μ_{min}.

THEOREM 3.

Minimal number of processing elements needed is bounded by $\alpha \le \mu_{min} \le \beta$.

PROOF: In case of $\zeta \ge \lceil \pi \rceil$ the proof is evident since there exists a moment j when at least ζ critical nodes must be fired. Suppose now that $\zeta < \lceil \pi \rceil$. Let μ_j, $j = 1,2,\ldots,t^*$ denote the number of internal executing nodes at the moment j. Then $\sum_{j=1}^{t^*} \mu_j = t_{seq}$. Suppose that $\mu_j < \pi$, for all $j=1,2,\ldots,t^*$. Then we have $t_{seq} = \sum_{j=1}^{t^*} \mu_j < \sum_{j=1}^{t^*} \pi = t_{seq}$, a contradiction. Consequently, $\mu_{j'} \ge \pi$, for some j', $1 \le j' \le t^*$. Since the number $\mu_{j'}$ is an integer, we also have $\mu_{j'} \ge \lceil \pi \rceil$, and a fortiori $\mu_{min} \ge \lceil \pi \rceil$. The right side of the nonequation is also evident. Q.E.D.

If we get $\alpha = \beta$ then at least one of the plans for immediate or lazy execution is also an optimal one. If $\alpha < \beta$, which is much more possible, we can try to push the upper bound β down by additional computations where we chose another plan of the program graph execution. For example, we may try to choose the sets E_j^v on each step in such a way, that $|F_j - V_F|$, $1 \le j \le t^*$, is as close as possible to α. The details are discussed in next section.

* $\lceil r \rceil$ is the lowest integer, greater or equal to r.

7.3. THE PLAN FOR HEURISTIC EXECUTION

Since the lower bound on the minimum number of processing units needed is given by α, we choose the sets E_j^v in such a way, that the number $|F_j - V_F|$ is on each step j, $1 \le j \le t^*$, as close as possible to α. To do this we first consider the number $c = |E_j^* - V_F| + |F_j^{old}|$ of the processors that are already needed at the moment j. If this number is greater than or equal to α, nothing is put into the set E_j^v. On the other hand, if $c < \alpha$, we consider the executable nodes, that need not to be fired at the moment j, i.e. the set $E_j - E_j^*$. If there are at most $m = \alpha - c$ such nodes we put all of them into the set E_j^v. However, if there are more than m such nodes n, we choose m of them, having the highest values $l(n, V_F)$, and put them into the set E_j^v (Fig.4).

```
c := |E*j - VF| + |F^old_j| ;
if c ≥ α then E^v_j := ∅
else
  begin
    m := α - c ;
    if |Ej - E*j| < m  then  E^v_j := Ej - E*j
    else Choose m nodes n from Ej - E*j having
         the longest values l(n,VF) and put
         them into the set E^v_j ;
  end;
```

Fig.4: Heuristic choosing the sets E_j^v.

8. ALGORITHM FOR OBTAINING EXECUTION PLANS

In this section we describe the algorithm for obtaining a plan of the execution. Different plans may be obtaind according to the rule by which the subsets E_j^v are chosen (see step 15).

At the moment j = 0 the sets C_j, C_j^{new}, E_j, E_j^{new}, F_j, and F_j^{new} are initialized to the set V_S of starting nodes. The set N_j is initialized to the set $V - V_S$. All the other sets are empty.

The plans for immediate and for lazy execution are computed by choosing $E_j^v := E_j - E_j^*$ and $E_j^v := \emptyset$, respectively, on each step (15). The plan for the heuristic execution is obtaind by choosing the sets E_j^v on each step (15) according to the sequence described on Fig.4.

The algorithm is implemented on LSI-11/23 computer under RT-11 operating system and tested on several program graphs. The Fig.7 describes an analysed program graph and the resulting time control vectors obtained.

(1) Evaluate $l(m,n)$ for all nodes m,n by computing the transitive closure of the program graph ;

(2) initialize sets ; $j := 0$;

(3) <u>while</u> $D_j \neq V$ <u>do</u>

<u>begin</u>

(4) $j := j + 1$;

(5) $F_j^* := \{ n \mid n$ finished execution at the moment $j \}$

(6) $C_j^{old} := C_{j-1} - F_{j-1}^{new}$;

(7) $C_j^{new} := \{ n \mid n$ received at the moment j its first input operand $\}$

(8) $C_j := C_j^{old} \cup C_j^{new}$;

(9) $E_j^{old} := E_{j-1} - F_{j-1}^{new}$;

(10) $E_j^{new} := \{ n \mid n$ received at the moment j its last input operand $\}$

(11) $E_j := E_j^{old} \cup E_j^{new}$;

(12) $F_j^{old} := F_{j-1} - F_j^*$;

(13) $l'(F_j^{old}, V_F) := \max l'(n,V_F)$, where $n \in F_j^{old}$;

(14) $E_j^* := \{ n \in E_j \mid l(n,V_F) = l(E_j,V_F) \geq l'(F_j^{old},V_F) \} \cup (E_j \cap V_F)$;

(15) Choose a subset E_j^v of the set $E_j - E_j^*$;

(16) $F_j^{new} := E_j^* \cup E_j^v$;

(17) $F_j := F_j^{old} \cup F_j^{new}$;

(18) $D_j := D_{j-1} \cup F_j^*$;

(19) $N_j := V - (C_j \cup F_j \cup D_j)$;

(20) Adjust the components of time control vectors for the nodes in C_j^{new}, E_j^{new} and F_j^{new} ;

<u>end</u> ;

Fig.5: Algorithm for obtaining execution plans of a program graph.

9. CONCLUSION

What is the time comlexity of the algorithm on Fig.5? The time complexity of the step (1) is $O(|V|^3)$, since this is the time complexity of the computing the longest paths between all nodes in a graph [Law76]. The time complexity of the steps (2) to (20) is $O(|V|^2)$, since the loop (3) is entered $O(|V|)$ times, each of the steps (4) to (20) having time complexity $O(|V|)$. Consequently, the overall time complexity of the algorithm on Fig.5 is is $O(|V|^3)$.

The algorithm was tested on 400 randomly generated program graphs with at most 64 nodes. In only 0.5% of analysed graphs $\mu_{heu} > \beta$ was obtained. In all other cases μ_{heu} was less than or equal to β. Heuristic plan improved both immediate and lazy execution in 50.25% of cases. Furthermore, many (55.75%) of heuristic executions were also proved to be optimal, since the associated numbers of the processing elements needed equaled the values α, as was the case on the Fig.7. Note, that this number is pesimistic since there were very probably graphs that could not be executed using only α processing elements. The results are depicted on Fig.6.

It is to point out that the algorithm on Fig.5 can be used for minimization of some other resources such as the number of primary memory locations needed .

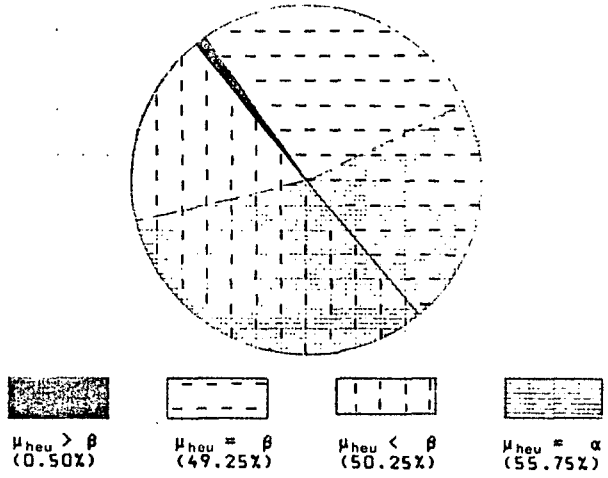
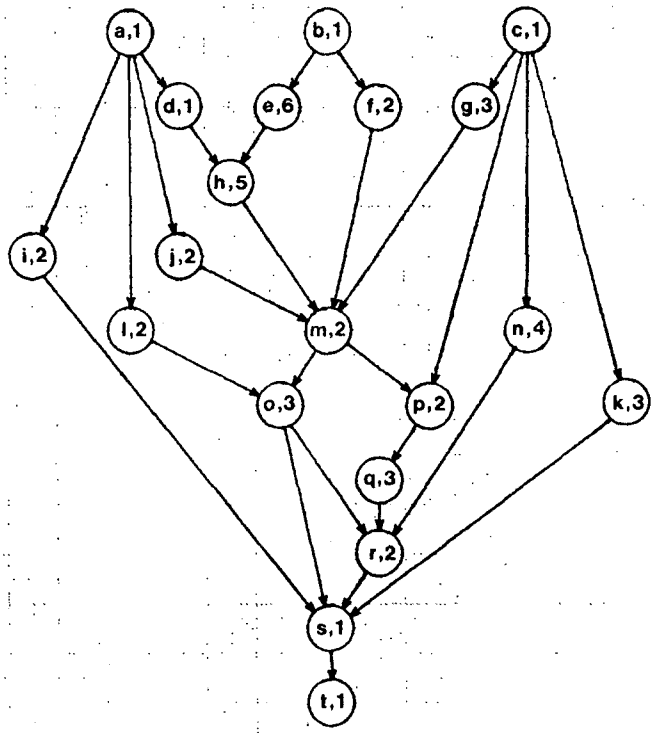


Fig.6: The behavior of the heuristic plan.

v	$\tau^{imm}_{F,v}$	$\tau^{lazy}_{F,v}$	$\tau^{heu}_{F,v}$
a	(0, 0, 0)	(0, 0, 0)	(0, 0. 0)
b	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
c	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
d	(1, 1, 1)	(1, 1, 6)	(1, 1, 1)
e	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)
f	(1, 1, 1)	(1, 1,10)	(1, 1, 4)
g	(1, 1, 1)	(1, 1, 9)	(1, 1, 1)
h	(2, 7, 7)	(7, 7, 7)	(2, 7, 7)
i	(1, 1, 1)	(1, 1,19)	(1, 1, 9)
j	(1, 1, 1)	(1, 1,10)	(1, 1, 2)
k	(1, 1, 1)	(1, 1,18)	(1, 1, 6)
l	(1, 1, 1)	(1, 1,14)	(1, 1, 4)
m	(3,12,12)	(12,12,12)	(4,12,12)
n	(1, 1, 1)	(1, 1,15)	(1, 1, 6)
o	(3,14,14)	(14,16,16)	(6,14,14)
p	(1,14,14)	(1,14,14)	(1,14,14)
q	(16,16,16)	(16,16,16)	(16,16,16)
r	(5,19,19)	(19,19,19)	(10,19,19)
s	(3,21,21)	(19,21,21)	(9,21,21)
t	(22,22,22)	(22,22,22)	(22,22,22)



$\mu_{imm} = 9$ $\mu_{lazy} = 4$ $\mu_{heu} = 3$

$|\pi| = 3$ $\zeta = 1$ $\alpha = 3$

In this case the heuristic plan is also optimal!

Fig.7: Plans for immediate, lazy and heuristic execution of a given program graph.

10. LITERATURE

[COM82] COMPUTER, Soecial Issue On Dataflow Systems, Vol.15, No.2, Feb.1982

[KFGS84] Kapauan A., J.T.Field, D.B.Gannon, L.Snyder: 'The Pringle Parallel Computer', Proc. 11th Int'l Symp. Comp. Arch., IEEE Press, New York, 1984, pp.12-20

[Law76] Lawler E.: Combinatorial Optimization: Networks and Matroids, Holt, Rinehart&Winston, New York, 1976

[RoŠ86] Robič B., J.Šilc : 'Analysis of Static Data Flow Program Graphs', to be published in Elektrotehniški vestnik, (in Slovene)

[TJS83] Tokoro M., J.R.Jagannathan, H.Sunahara: 'On the Working Set Concept for Data-flow Machines', Proc. 10th Int'l Symp. Computer Architecture, IEEE Press, New York, 1983, pp.90-97

[ŠiR85] Šilc J., B.Robič : 'Basic Principles of Data Flow Systems', Informatica 2/85, pp.10-15 (in Slovene)

Saša Prešeren, Rudolf Murn, Dušan Peček
Institut Jožef Stefan
Ljubljana, Jugoslavija

UDK: 681.3:514.1

Procesiranje slik je povezano z obdelavo velikega števila podatkov v kratkem času. Ker so številne obdelave slikovnih podatkov, ki jih izvajamo na zajeti sliki med seboj neodvisne in iterativne, omogočajo uporabo paralelnega pristopa pri procesiranju slike. Članek podaja pregled, kako se papalelni procesorji uveljavljajo pri procesiranju slik. Seznanimo se s principom delovanja mrežnega procesorja, to je najbolj razširjene paralelne arhitekture za procesiranje slik. Na koncu članek nakazuje smer razvoja paralelnih računalniških arhitekturpri procesiranju slik.

PARALLEL IMAGE PROCESSING. Image processing requires real time processing of large number of data. Because many operations on image data are independent and iterative, one can apply parallel approach in image processing. This paper discusses parallel processor architectures in image processing. Most popular parallel processor arhitecture for image processing is array processor The principle of array processor architecture is given and future trends in parallel processor arhitecture for image processing is discussed.

1. UVOD

Za procesiranje slik je značilna obdelava velikega števila podatkov v kratkem času. Podatke dobivamo preko sistemov za zajemanje slik, ki večinoma temeljijo na CCD kamerah (PRE84, PRE85a). Ti podatki so največkrat simetrično urejeni v strukture imenovane podatkovni vektorji.

Procesiranje slik zahteva obsežne numerične operacije na vektorjih podatkov. Te operacije so preprosto seštevanje in odštevanje podatkovnih vektorjev, ter tudi zahtevnejše matematične operacije kot so konvolucija, Fourierjeva transformacija, računanje gradientna ali celo sintaktična analiza. Če te algoritme implementiramo direktno v materialni opremi dobimo hiter sistem za procesiranje slik, ki pa je ozko namenjen izbrani alikaciji. Lahko pa procesiranje slik izvajamo tudi na splošnem računalniku, pri čemer pa na račun nižje cene sistema dobimo počasnejši sistem za procesiranje slik. Številni proizvajalci sistemov za procesiranje slik nudijo danes računalnike, kjer uporabnik s posebnim programskim orodjem za procesiranje slik razvija svojo aplikacijo. Vse funkcije in operacije na slikovnih elementih določi uporabnik programsko z ukazi, ki jih ima v knjižnici funkcij in operatorjev. Ti ukazi pa sprožijo posamezne obdelave, ki so vezane bodisi na hitre in optimalne programske algoritme ali pa celo na aktiviranje določene materialne opreme za hitro obdelavo slikovnih informacij (PRE85b). Zato zahteva procesiranje slik hitre matematične procesorje in veliko pomnilnika. Pri problemih procesiranja slik v realnem času se poslužujemo prijemov paralelnega procesiranja, kjer centralno procesno enoto razbremenimo številnih iterativnih obdelav, ki so povezane s procesiranjem slik. V ta namen v zadnjem času uporabljajo paralelne arhitekture računalnikov in sicer artitekture, kjer enemu ukazu ustreza večkratni podatek (SIMD - single instruction multiple data) ter architekture, kjer večkratnim ukazom ustreza večkratni podatek (MIMD - multiple instruction multiple data) (STO80), (BRA86). V tistih aplikacijah, kjer sta struktura slikovnega polja in način obdelave slikovne informacije fiksno določeni, prevladujejo SIMD arhitekture. V teh primerih zmanjšamo fleksibilnost sistema za procesiranje slik, pridobimo pa pri hitrosti delovanja računalnika. Pri procesiranju slike z računalnikom vrste MIMD, pa večje število procesorjev deluje hkrati ali pa neodvisno na istih ali različnih podatkih pri procesiranju slike.

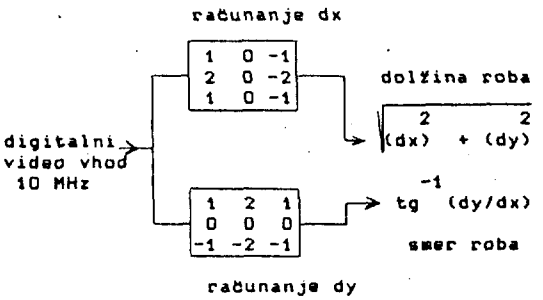
Prvi in še vedno najbolj razširjene vrste računalnikov s katerimi rešujemo probleme hitrega procesiranja slik pa so mrežni (array) procesorji (WIL86).

2. PARALELIZEM OBDELAV

Procesiranje slik zajema vrsto obdelav slikovnih informacij, ki se lahko izvajajo hkrati. Paralelizem obdelav je možen tako pri predprocesiranju slik kot tudi pri delovanju različnih operatorjev na sliko. Pri procesiranju slik lahko govorimo o štirih nivojih paralelizma in sicer:

- paralelizem operatorjev,
- slikovni paralelizem,
- paralelizem sosedov in
- bitni paralelizem svetlobnih elementov - pikslov.

Paralelizem operatorjev zajema hkratno delovanje različnih neodvisnih operatorjev za izločanje značilnosti na zajeto sliko v slikovnem zajemalniku. Delno spominja paralelizem operatorjev na cevno procesiranje. Primer paralelizma operatorjev za izračun Sobel-ovega operatorja v realnem času kaže slika 1. Paralelizem je dosežen s hratnim delovanjem dveh različnih operatorjev dimenzije 3x3 na slikovne elemente in sicer enim za x smer ter drugim za y smer. Rezultat te obdelave je dolžina roba in njegova smer ali naklon kar se uporablja pri relaksacijskih algoritmih pri robljenju slik.



Slika 1.: Paralelizem pri računanju Sobelovega operatorja v realnem času.

Slikovni paralelizem se uporablja takrat, ko različni procesorji sinhronizirano obdelujejo okolice različnih pikslov. Primer take obdelave je konverzija slike s 512x512 elementi v sliko s 1024x1024 elementi v realnem času. Slikovni paralelizem se velikokrat izvaja z računalniki vrste SIMD.

Paralelizem sosedov zajema paralelni dostop do sosednjih slikovnih elementov. Ta paralelizem izvajajo sistolični procesorji. Primer take obdelave je izvajanje vseh funkcij, ki zahtevajo za izračun vrednosti sosednjih elementov.

Bitni paralelizem pikslov se uporablja pri procesiranju slik takrat, ko je potreben hkratni dostop do vrednosti vseh bitov posameznega piksla.

Vse te paralelizme, ki so značilni za procesiranje slik se danes izvaja z različnimi vrstami paralelnih računalniških arhitektur.

3. PARALELNE ARHITEKTURE

Pri procesiranju slik v realnem času vedno pogosteje nadomeščamo klasične von Neumanove računalnike vrste SISD (single instruction single data) z arhitekturami, ki imajo neko obliko paralelizma.

Paralelne računalnike , ki se uporabljajo pri procesiranju slik lahko klasificiramo v naslednje razrede:

- SIMD ali vektorski računalniki, ki so lahko realizirani kot:
  * mrežni procesorji
  * cevno procesiranje
- MIMD ali večprocesorji:
  * večprocesorji s preklopnim vezjem za povezavo vsakega procesorja z vsakim pomnilniškim modulom
  * večprocesorji z ločenim lokalnim pomnilnikom

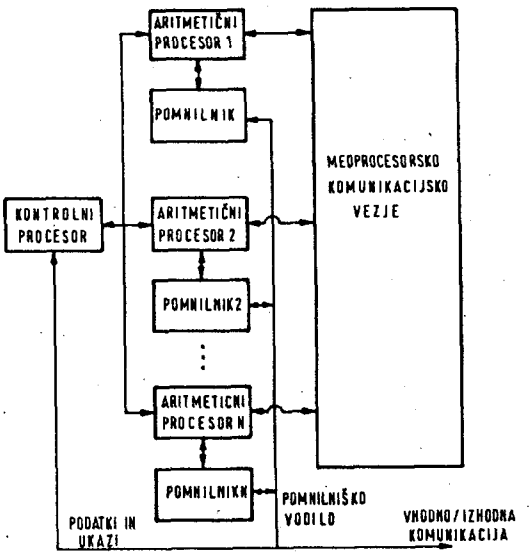
Računalnik s SIMD arhitekturo je vektorski procesor v smislu, da vsak ukaz deluje na vektorju podatkov ne pa na posameznem operandu. Vektorski procesor je lahko konstruiran kot mrežni procesor, pri katerem je vsak podatek v podatkovnem vektorju obdelan z drugim aritmetičnim procesorjem. Vektorske procesorje je mogoče implementirati tudi na drugačen način na primer s cevnim procesiranjem ali pa z arhitekturo "waveform array", ki temelji na VLSI polju mrežnih procesorjev. Pri mrežnem procesorju ima vsak procesor svojo aritmetično enoto in lokalni pomnilnik, ni pa tak procesor samostojen serijski računalnik ker nima svojega lastnega toka ukazov.

Arhitektura vrste MIMD je pravi večprocesor, ki sestoji iz N procesorjev, tako, da je lahko vsak procesor samostojen. Procesorji so med seboj povezani in med obdelavo slikovnih podatkov sodelujejo. Večprocesor z N neodvisnimi procesorji ima N ukaznih tokov in N podatkovnih tokov, torej en podatkovni tok na en ukazni tok.

3.1. Mrežni procesorji

Arhitektura mrežnih procesorjev je posebej prilagojena izvajanju ponavljajočih se operacij na pravilno urejenih strukturah podatkov. Te operacije izvajajo paralelno in zato mnogo hitreje kot klasični von Neumanovi splošni računalniki. Zaradi te regularnosti lastnosti je programiranje mrežnih procesorjev relativno preprosto v primeri s programiranjem ostalih paralelnih procesorjev in jih lahko uporabimo pri večini problemov iz področja računalniškega vida. Najboljše rezultate pri hitrosti procesiranja slik dobimo v primeru, ko je arhitektura mrežnega procesorja popolnoma usklajena z organizacijo podatkov, ki jih mrežni procesor obdeluje.

Ker so pri procesiranju slik najbolj zastopani mrežni procesorji, poglejmo organizacijo takega mrežnega računalnika.



Slika 2.: Princip organizacije mrežnega računalnika za procesiranje slik.

Na sliki 2 vidimo permutacijsko vezje, ki omogoča izmenjavo informacij med procesorji. Permutacijsko vezje je prikazano kot povezava med vsemi procesorji, lahko pa je tudi take vrste, da omogoča povezavo med lokalnimi pomnilniki. Kontrolna enota na sliki 2 je računalnik s svojim lokalnim pomnilnikom, aritmetično enoto in hitrimi registri. Bistvena razlika med kontrolnim procesorjem in ostalimi procesorji v mrežnem računalniku je dejstvo, da kontrolni procesor lahko izvaja ukaze za pogojne skoke in na ta način določa vrstni red izvajanja ukazov. Aritmetični procesorji pa nimajo te lastnosti ker morajo biti vedno sinhronizirani in torej ne morejo izvajati različnih operacij po ukazu za pogojni skok.

Tako kot v klasičnih von Neumanovih računalnikih so tudi pri mrežnem računaniku ukazi shranjeni v centralnem pomnilniku skupaj s podatki. Glavni pomnilnik v sistemu na sliki 2 je

skupni pomnilnik vseh N procesorjev. Vsak ukaz je lahko kontrolni ukaz ali pa vektorski ukaz, ki se izvaja v polju procesorjev. Tok ukazov pa je precej podoben klasičnem serijskem toku ukazov, tako, da se ukazi izvajajo zaporedno. Primarna funkcija kontrolnega procesorja je določitev, kje naj se posamezni ukaz izvede, bodisi v sami kontrolni enoti ali pa v polju procesorjev.

Pri delu z mrežnimi procesorji naletimo na probleme pri programiranju mrežnega procesorja. Tisti, ki so hitri zahtevajo programiranje v mikrokodi. Pri ostalih, ki dovoljujejo uporabo visokonivojskega jezika ali imajo za uporabnika pripravljeno knjižnico ukazov, pa zasledimo počasnejše delovanje. Najhitrejši so tisti mrežni procesorji, ki so optimirani za en sam format podatkov in specifično vrsto operacij kot na primer hitro Fourierjevo transformacijo - FFT.

Obstajajo tudi posebne metode za avtomatko transformacijo serijskega programa v paralelni program za izvajanje na mrežnem procesorju (ZEL86). Taka avtomatska transformacija je učinkovita za tiste vrste programov, ki imajo močno iterativno strukturo - take pa so obdelave pri procesiranju slik.

Na žalost pa ta avtomatska transformacija ne delujejo za vse programe, ki imajo visoko stopnjo iterativnosti. Prav tako pa ne obstaja univerzalna metoda za pretvorbo seriskih programov v paralelno obliko, če ti programi nimajo iterativne strukture.

Danes so mrežni procesorji sestavni del splošnih računalniških sistemov. Z računalnikom komunicirajo mrežni procesorji po vodilu kot na primer multibus ali unibus ali pa celo preko DMA prenosa.

Nekatere SIMD arhitekture sistemov za procesiranje slik dovoljujejo procesiranje med skaniranjem slike ter s tem močno zmanjšajo velikost slikovnega pomnilnika za zajemanje slike. S tem se tudi poveča hitrost procesiranja slike ter zmanjša obseg strojne računalniške opreme.

Ker številni proizvajalci računalniških vezij vidijo bodočnost procesiranja slik v SIMD arhitekturah, se na tržišču pojavljajo nova hitra vezja za izvajanje posameznih funkcij za procesiranje slik. Texas Instruments je na primer razvil vezje (chip) za Sobel-ov detektor robov, ki obdela sliko velikosti 512 x 512 x 8 bitov 30 krat na sekundo. ITT razvija vezje imenovano celični mrežni procesor vrste SIMD RISC za procesiranje slik. To vezje bo imelo med 16 in 20 procesorjev vrste 2903 v enem chipu.

3.2. MIMD arhitekture pri procesiranju slik

Za današnji razvoj računalnikov vrste MIMD, ki so namenjeni procesiranju slik so značilne tri smeri razvoja:

- sočasno delo več mrežnih procesorjev,
- več procesorjev na isti plošči
- večprocesorski chip.

Proizvajalci mrežnih procesorjev vidijo uspeh posameznih mrežnih procesorjev pri procesiranju slik in želijo izboljšati performanse sistemov za procesiranje slik z MIMD arhitekturo, ki vključuje sočasno delo več mrežnih procesorjev. Ti mrežni procesorji so organizirani bodisi paralelno ali zaporedno.

Druga smer razvoja MIMD arhitektur za procesiranje slik zajema sočasno delo dveh procesorjev na isti plošči. Čeprav je ideja o uporabi dveh procesorjev podobna ideji o dveh mrežnih procesorjih, pa z dvema procesorjema na isti plošči znatno zmanjšamo obseg materialne opreme, ki je potreben za izdelavo sistema za procesiranje slik.

Tretja smer pa zagovarjajo večprocesorski chip, ki predstavlja polje procesorjev za hitro obdelavo slikovnih podatkov.

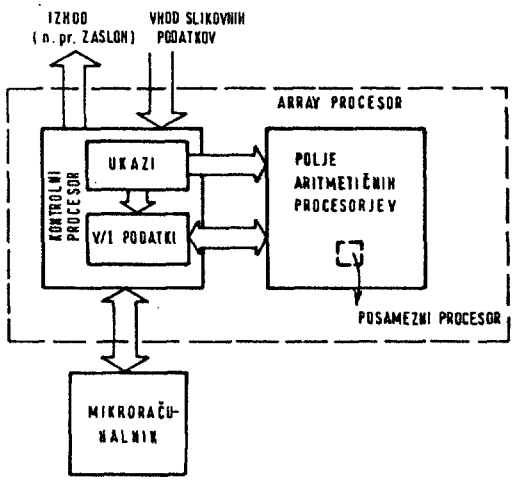
Vse te oblike MIMD arhitektur se za procesiranje slik že dobijo na tržišču.

3.3. Procesorji za procesiranje pikslov

Sodobni poskusi pri razvoju RISC MIMD arhitektur imajo svoj odziv tudi na področju procesiranja slik. S kombiniranjem več procesorjev v enem samem chipu je danes mogoče uresničiti dolgo trajajoče sanje teoretikov - to je pikselni procesor. Čeprav pri tem obstaja še veliko nerešenih problemov in je mnenje mnogih, da še ni prišel čas za pikselni procesor, pa bomo verjetno kmalu slišali poročila o uporabi pikselnih procesorjev za izvrševanje določenih funkcij pri procesiranju slik. To bodo verjetno tiste funkcije, ki zahtevajo manj komunikacije med procesorji. Za aplikacije, ki zahtevajo FFT ali podobne transformacije pa bi zaradi velike komunikacije med procesorji prišlo do zasičenja in zmajšanja hitrosti obdelave slikovnih podatkov.

4. APLIKACIJE

Uporaba mrežnega procesorja se je uspešno izkazala v mnogih aplikacijah. Kot primer si oglejmo vizualno kontrolo kvalitete pri industrijski proizvodnji kozarcev z otroško hrano. Konfiguracija sistema je prikazana na sliki 3 in temelji na procesorju 68000. Zajemalec slike ima velikost 512x512 pikslov. Prvotno sistem ni imel mrežnega procesorja, ampak se je za obdelavo slike uporabljal procesor 8086. Čas potreben za optično kontrolo enega izdelka je bil eno sekundo, kar ni zadovoljevalo hitrosti tekočega traku. Izračuni so pokazali, da bi za dosego zahtevane hitrosti, ki jo narekuje tekoči trak, bilo treba uporabiti 10 do 15 procesorjev 8086. Razvijalci so se namesto tega odločili za mrežni procesor, ki se je izkazal za cenejšo varianto, hkrati pa je



Slika 3.: Sistem za optično kontrolo kvalitete v industriji, ki temelji na povezavi mrežnega procesorja in klasičnega von Neumanovega računalnika.

hitrost optične kontrole povečala na 15 izdelkov na sekundo. Ta primer kaže, da mrežni procesor v kombinaciji s klasičnim računalniš kim sistemom lahko pri procesiranju slik močno zboljša performanse celotnega sistema za optič no kontrolo.

5. ZAKLJUCEK

Želje po procesiranju slik, to je obdelavi ogromnega števila podatkov v realnem času, rastejo hitreje kot sposobnosti računalnikov za procesiranje slik. Zato se pri procesiranju slik še vedno borimo s časom in iščemo posebne rešitve v materialni opremi, ki so pa tem hitrejše čim bolj so prilagojene fiksnem formatu podatkov.

Poudariti moramo, da so posamezni pristopi in arhitekture procesiranja slik primerni samo za nekatere aplikacije. Ker ima vsaka računalniš ka arhitektura svoje dobre in slabe lastnosti, najdemo sisteme za procesiranje slik, ki so sestavljeni iz različnih arhitehtur računalnikov in s tem prilagojeni posamezni aplikaciji. Tako kot se vsaka aplikacija razlikuje, moramo pri načrtovanju sistema za procesiranje slik upoštevati vse zahteve posamezne obdelave in ustrezno izbrati bodisi cenen sistem računalniš kega vida, ki temelji na osebnih računalnikih, bodisi mrežni procesor ali pa celo MIMD arhitekturo.

Obstaja pa ogromno aplikacij, kjer zadostuje za procesiranje slik običajen PC računalnik, opremljen z video modulom. Tak PC računalnik z video modulom zajema sliko resolucije 256x256x4 bite v 1/60 sekunde. Za razliko od paralelnih sistemov za procesiranje slik, ki so v cenovnem razredu nekaj 100.000 dolarjev, pa je sistem za procesiranje slik, ki temelji na PC računalniku v cenovnem razredu nekaj tisoč dolarjev.

Mrežni procesorji zasedajo danes nedvomno vodilno vlogo pri sistemih za paralelno procesiranje slik. Ker so hitri in tudi fleksibilni jih lahko uporabljamo za številne aplikacije. V področje procesiranja slik prodirajo sistemi s SIMD in MIMD arhitekturami, ki so posebej primerni za izvajanje nekaterih funkcij pri procesiranju slik. Zato pričakujemo, da bodo bodoči sistemi za procesiranje slik organizirani kot MIMD arhitekture z več procesorji, ki bodo vključevali tudi več mrežnih procesorjev. Pričakuje se, da bodo višje generacije računalniških arhitektur izredno ugodne za procesiranje slik v realnem času.

6. LITERATURA

(BRA86) P. Brajak: Paralelno procesiranje - arhitektura 90-tih godina, MIPRO'86, Opatija 1986.

(PRE84) S. Prešern: Optična kontrola plošč tiskanega vezja, Informatica 4/1984.

(PRE85a) S. Prešern: Quality control based on feature extraction for CCD linear image inspection system, Robotics and automation '85, Lugano.

(PRE85b) S. Prešern: Računalniški vid - inteligentni senzorski sistem za merjenje dolžine, ISEMEC 85, 1985.

(STO80) H. Stone: Introduction to computer architecture, SRA computer science series, 1980.

(WIL86) A.C. Wilson: Array processors - the best way to process images, Digital Design, januar 1986.

(ŽEL86) A.P. Železnikar: A Method for parallel process distribution and selection in a processor array, MIPRO'86, Opatija 1986.

Igor Ozimek, Viktor Avbelj
Institut Jožef Stefan, Ljubljana

UDK: 681.3.06

Članek govori o video terminalu PMT-100. Poudarek je na nekaterih bistvenih izvirnih elementih organizacije logike in programske opreme. Nekaj besed je posvečenih tudi opisu razvojnega dela in njegovi organizaciji ter učinkovitosti, kar v razmerah nizke produktivnosti v našem okolju zasluži posebno pozornost.

The article is about a VT100 compatible video terminal PMT-100. Some interesting hardware and software solutions are described, which enables the complete logic circuitry and power supply of the terminal to be housed in the keyboard housing.

UVOD

PMT-100 je terminal, ki je funkcionalno enakovreden terminalu VT100 (DIGITAL). Ob obstoječih tovrstnih terminalih na našem tržišču (PAKA 1000, PAKA 3000) je seveda opravičljivo začudenje, čemu razvijati še enega. Razlog je preprost: Obstoječi terminali so sorazmerno dragi. Visoka cena terminala postane problem pri trženju cenenih mikroračunalnikov. Lahko se zgodi da je terminal celo dražji od računalnika. (V našem primeru je šlo za mikroračunalnik PMP-11, razvit na IJS. To je računalnik, programsko skladen s sistemi PDP-11 in LSI-11 oziroma ustreznimi sistemi DELTA ali ENERGOINVEST). Cilj razvoja je bil torej terminal, ki bi imel funkcionalnost terminala VT100, vendar bi bil bistveno cenejši od obstoječih tovrstnih terminalov na našem tržišču.

Ideja, ki je bistveno poenostavila (torej: pocenila) PMT-100, je za terminale razreda VT100 neobičajna, ob misli na obstoječe hišne računalnike pa niti ne preveč originalna: vsa logika terminala skupaj s pripadajočim napajalnikom, transformatorjem in tipkami je narejena na eni sami tiskanini, ki je vgrajena (seveda) v ohišje tipkovnice. Uporabljene so domače tipke proizvodnje IEVT. Napajalnik je preklopnega tipa in priključna moč je manjša od 8 (osem) W. Celotna zadeva je enako velika kot so običajne tipkovnice - uporabljen je isti tip ohišja, v kakršnega je vgrajena tipkovnica terminala PAKA 1000 ali PAKA 3000. Na to "tipkovnico" (to je pravzaprav terminal) priključimo običajen (cenen) računalniški monitor.

MATERIJALNA OPREMA (HARDWARE)

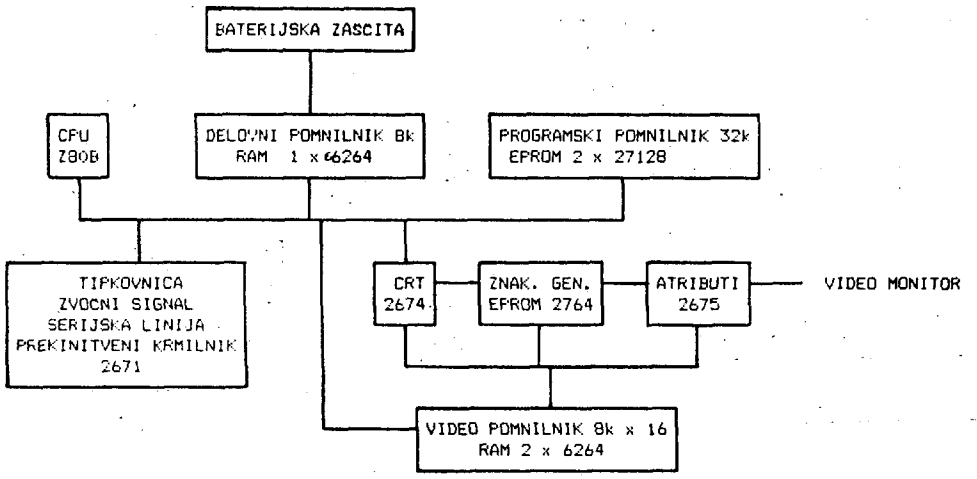
Uporabljena so tri standardna integrirana vezja iz družine 267x (Signetics):

- 2674 - CRT krmilnik (z video RAM pomnilnikom je vezan v načinu "Independent Mode"
- 2675 - krmilnik atributov (podčrtovanje, mežikanje, ...)
- 2671 - krmilnik tipkovnice, ki je hkrati tudi serijski asinhroni komunikacijski vmesnik s programskim generatorjem takta, generator zvočnih signalov in prekinitveni krmilnik v načinu za intel8080.

Kot znakovni generator je uporabljeno EPROM vezje 2764, ki omogoča definicijo 512 različnih znakov. Izbrano število točk prikazovanih znakovnih celic je 10 x 12. Znakovne celice so definirane z matriko dimenzije 8 x 12. Prirojena lastnost CRT krmilnika omogoča, da se vsebina 8. kolone v opisu celice uporabi tudi za 9. in 10. kolono (npr.: velika črka A ima v tem opisu dimenzijo 7 x 7). Sorazmerno visoka matrika znakov (12 vrstic v primerjavi z 10 vrsticami pri VT100) omogoča lep zapis jugoslovanskih znakov, onemogoča pa delovanje terminala po ameriškem video standardu (60 Hz), čemur smo se že vnaprej brez škode odrekli.

Na plošči je še video RAM pomnilnik zmogljivosti 8k znakov in atributov (dve integrirani vezji 6264), potrebna "lepilna" logika v tehniki 74LS in mikroračunalnik v naslednji konfiguraciji:

- procesor Z80B. B izvedba procesorja je potrebna zaradi potrebe po izredno hitri obdelavi prekinitvenih zahtev. Glavni vzrok za to je sorazmerno skromna (cena!) materijalna oprema terminala - za mehak pomik (soft scroll) je edina materijalna podpora tista, ki je že vgrajena v CRT krmilniku, ki pa je (kot bomo še omenili), zelo nepopolna. Procesor deluje v prekinitvenem načinu mode 1 (vstopna točka obdelave vseh prekinitev je skupna), kar se je v tem primeru pokazalo kot najboljše.
- EPROM pomnilnik zmogljivosti 32k zlogov (dve vezji 27128).
- delovni RAM pomnilnik zmogljivosti 8k (6264) zlogov v CMOS tehniki in zaščiten pred izpadom napajanja (alkalne, litijeve ali NiCd baterije na tiskanini). Zaščita omogoča uporabo tega pomnilnika za "SETUP" nastavitve, s čimer se izognemo zapletom pri uporabi EEPROM (electrically ereasable PROM) vezij, malce pa si tudi olajšamo pisanje programske opreme v visokem programskem jeziku.



Bločna shema PMT-100

Logični načrt terminala ni priložen, ker (razen pravkar opisanih stvari) ne prinaša ničesar novega.

NEKAJ PRIPOMB O CRT VEZJU 2674

Ob prvem pogledu v priročnik daje to vezje zelo dober vtis, saj podpira mehak pomik in znake dvojne višine/širine, kar je potrebno za emulacijo VT100 in česar CRT krmilniki običajno nimajo vgrajenega. (V povezavi s krmilnikom atributov podpira tudi vse VT100 znakovne atribute - podčrtovanje, mežikanje, intenzivnost, ozadje). Kmalu nato pa sledi kruta stvarnost: uporaba pomika (scroll) in znakov dvojne višine/širine zahteva nenavadno močno dinamično programsko podporo na osnovi prekinitvenih zahtev. V skrajnem primeru imamo pri uporabi mehkega pomika (soft scroll) samo 64 mikrosekund časa za obdelavo prekinitvene zahteve, kar predstavlja za procesor Z80 (in njegovega programerja) zelo težko nalogo.

(Vprašanje časa je, kdaj se bo na tržišču pojavil kak "inteligentnejši" CRT krmilnik, ki bo samostojno reševal opisane probleme pa morda še problem mapiranja vrstic zaslona, ki je opisan spodaj.)

PROGRAMSKA OPREMA

Programska oprema je pretežno pisana v jeziku C, kar se je pokazalo kot zelo dobra odločitev. Visok programski jezik omogoča mnogo večjo učinkovitost pri programiranju (manj napak), lažje vzdrževanje (popravljanje, dopolnjevanje), razpoložljivost in enostavnost uporabe aritmetičnih operacij pa tudi večjo možnost "off-line" razvoja in testiranja delov programa na poljubnem (velikem) računalniku z ustreznim prevajalnikom.

Uporaba visokega programskega jezika pomeni dve potencijalni nevarnosti:

1. povečanje obsega prevedenega programa. V našem primeru zaseda program 24k, kar je kar 3-krat več, kot v nekaterih starejših enakovrednih terminalih. To zelo veliko razliko lahko pripišemo predvsem dvema vzrokoma:
   - strojna koda procesorja Z80 je precej neprimerna za prevode iz višjih programskih jezikov; dovolj je omeniti nezmožnost naslavljanja sklada relativno glede na kazalec sklada. Zaradi tega so prevodi velikokrat precej "okoli ovinka".
   - obstoj "lepih" podatkovnih struktur (record v PASCALU, struct v C) kaj hitro "zapelje" programerja k njihovi uporabi, pri čemer hitro pozabi, kako naporna postane potem takšna stvar za procesor, kot je Z80 (zopet zaradi neobstoja primernih strojnih instrukcij in naslovnih načinov).

   Vendar so danes EPROM vezja dovolj poceni, da opisana težava pravzaprav ni težava, vse dokler nam ne zmanjka spominskega prostora.
2. Druga nevarnost je podaljšanje časa izvajanja (ki je seveda povezano z večjo dolžino prevedenega programa). Ta težava je kritična samo na določenih posebej obremenjenih delih programa, ki se jih da običajno napovedati (v primeru PMT-100 je to pot od sprejema izpisljivega znaka po serijski liniji do njegovega izpisa na zaslon). Tej težavi se da izogniti z več ukrepi, ki imajo skupno ime "grdo programiranje". V kritičnem delu se je potrebno izogibati zapletenim podatkovnim strukturam (bolje več enostavnih spremenljivk kot niz), preveč modularnemu programiranju (bolje ena dolga procedura kot več kratkih), pametno izbrati deklaracijo spremenljivk (v jeziku C npr.: register bolje kot auto, static bolje kot auto), zadnja rešitev pa je programiranje kritičnih delov v zbirnem jeziku. Nekateri visoki jeziki omogočajo vstavljanje zbirnih ali strojnih instrukcij neposredno v višjem jeziku, malce nerodnejša varijanta pa je pisanje ločenih zbirnih procedur, ki jih lahko potem kličemo iz visokega jezika.

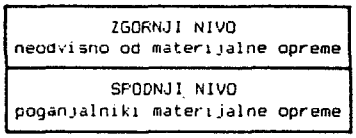
ORGANIZACIJA PROGRAMSKE OPREME - NIVOJI

Programsko opremo PMT-100 lahko razdelimo na dva nivoja. Zgornji nivo (to je nivo, kjer se med drugim dogaja razpoznavanje krmilnih zaporedij), je pisan tako, da je neodvisen od materijalne opreme. Vse osnovne operacije materijalne opreme (recimo: postavitev kazalca (cursor), vpis znaka na zaslon,

pomik vrstic (scroll), zvočni signal, prižiganje lučk) zahteva s klicem ustreznih procedur. Seveda je treba priznati, da način uporabe teh procedur do neke mere vendarle odseva strukturo materialne opreme in organizacije spodnjega nivoja programske opreme, kar je nujen (minimalen) kompromis za doseganje čimvečje hitrosti obdelave.

Spodnji nivo programske opreme sestavljajo procedure, ki neposredno krmilijo periferijo.

Ta delitev programske opreme na nivoje je hkrati naravna delitev dela pri programiranju.



ORGANIZACIJA PROGRAMSKE OPREME - PROCESI

V videoterminalu teče hrati več opravil. Medtem ko uporabnik pritisne tipko (terminal jo ne sme spregledati), prileti po serijski liniji znak (terminal ga ne sme spregledati), medtem je potrebna programska podpora osveževanja zaslona, itd. V našem primeru imamo procese (opravila), ki so našteti spodaj. Vsi razen glavnega procesa imajo svoj izvor v prekinitvenih zahtevah. Glavni proces je proces najnižje prioritete, ki časovno ni kritičen. Nad njim so nanizani prekinitveni procesi, ki imajo različne prioritete glede na nujnost akcije, ki jo opravljajo. Prioriteta posameznega procesa je fiksna. Proces z višjo prioriteto lahko (mora) prekine proces z nižjo prioriteto.

Seznam procesov (po rastoči prioriteti).

Glavni proces - nivo 0
Tu je vključena procedura za nastavitev terminala ("SETUP"), procedure za razpoznavanje VT52 in VT100 krmilnih zaporedij, ter procedure za komunikacijo med prekinitvenimi procesi. Ta komunikacija poteka preko podatkovnih vrst, ki so prirejene večini prekinitvenih procesov (razen CRT procesu). Primer:
- tipkovni proces sprejme tipko in jo postavi v svojo oddajno vrsto
- glavni proces to ugotovi in pod določenimi pogoji (recimo: če terminal ni v lokalnem načinu) vzame znak iz oddajne vrste, ga postavi v sprejemno vrsto oddajnega procesa in oživi oddajni proces (dovoli prekinitve oddajnika)
- oddajni proces odda znak po serijski liniji. Če ni več znaka, se izključi.

(Manipulacija podatkovnih vrst poteka s pomočjo ustreznih procedur spodnjega nivoja programske opreme. Te procedure morajo biti v kritičnem delu zaščitene pred prekinitvijo. Prekinitev jemanja iz vrste in hkratno postavljanje v isto vrsto bi povzročilo zmedo.)

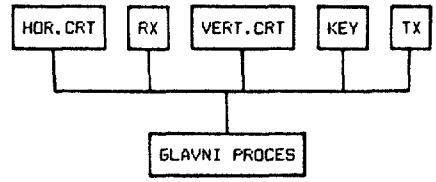
TX (oddajni) proces - nivo 1
Ta proces skrbi za oddajanje znakov iz sprejemne vrste TX procesa po liniji RS232. Poleg tega skrbi za oddajo XON/XOFF, če ju najde v posebenem visokoprioritetnem vmesniku.

KEY (tipkovni) proces - nivo 2
Ta proces sprejema tipke s tipkovnice in jih postavlja v oddajno vrsto tipkovnega procesa. Posebej procesira nekatere tipke (npr. SETUP, SCR) in postavlja ustrezne zastavice za glavni proces.

Proces CRT vertikalnih prekinitev - nivo 3
Precej obsežno procesiranje med vračanjem žarka v zgornjo levo točko. Tu se dogaja urejanje tabel kazalcev vrstic na zaslonu v zvezi s pomikanjem vrstic na zaslonu - glej spodaj.

RX (sprejemni) proces - nivo 4
Proces sprejema znake z linije RS232 in jih postavlja v oddajno vrsto sprejemnega procesa. Nekatere znake posebej prepoznava in obdeluje (NULL, DEL, (NO)SCR). Proces je kritičen in ne sme zamuditi znaka tudi pri hitrosti komunikacije 19200 (cca 0.5 msec na znak). Zato ga CRT vertikalni proces ne sme (in tudi ne more) prekiniti.

Proces CRT horizontalnih prekinitev - nivo 5
To je najvišji nivo. Ta proces skrbi za dinamično postavljanje registrov CRT krmilnika. Dovoljen čas za obdelavo je izredno kratek, največ 120 mikrosek., zato ima ta proces najvišjo prioriteto in ga ni moč prekiniti.



PROBLEM UREDITVE SLIKE V VIDEO POMNILNIKU

Poseben problem predstavlja ureditev slike v video pomnilniku v zvezi z delnim pomikanjem (scroll) zaslona. Delno pomikanje namreč podre enostavno razmerje med spominom in zaslonom, kar vidimo na spodnjem primeru pomikanja vrstic 1 do 3 navzdol (vrstica 3 zgine z zaslona in jo moramo pobrisati, nakar jo lahko uporabimo kot 1. vrstico nove slike na zaslonu).

RAM	zaslon pred skrolanjem	/ po skrolanju
1	1	3 (pobrisana)
2	2	1
3	3	2
4	4	4
5	5	5

Obstajata dve rešitvi tega problema:

1. Ohranja se stalno razmerje med video pomnilnikom in zaslonom. Vsebina video pomnilnika neposredno ustreza prikazu na zaslonu. To pomeni, da se pomikanje izvaja v video pomnilniku, kar zahteva intenzivno prepisovanje in porabi veliko časa ter bistveno upočasni delovanje.

2. Vsebina video pomnilnika se ne preureja, pač pa se preureja posebna tabela kazalcev, ki kažejo na mesta v video pomnilniku, kjer so zapisane posamezne vrstice zaslona. V zvezi s tem je potrebno tudi dodatno procesiranje pri osveževanju zaslona, vendar se izognemo zamudnemu prepisovanju video pomnilnika. Seveda mora uporabljeni CRT krmilnik omogočati tak način dela. Ta način je uporabljen tudi pri PMT-100.

Procesiranje pomika (scroll) opravlja proces CRT vertikalnih prekinitev. Glavni proces zahteva to s pomočjo procedure, ki postavi ustrezno zastavo (DOL / - / GOR) kot zahtevo zgoraj omenjenemu procesu. Območje pomika postavlja glavni proces s pomočjo druge procedure.

PREKINITVENA STRUKTURA

Zaradi neobstoja prioretizirane materijalne prekini-

tvene opreme (s stališča materijalne opreme (hardware) imajo vse prekinitve isti nivo) je stvar rešena na nivoju programske opreme, kar seveda zahteva nekaj več obdelave. Zadeva je rešena na ta način, da vsak od procesov nastavlja prekinitveno masko tako, da onemogoči tiste prekinitve, ki imajo po definiciji nižjo prioriteto od njega, po zaključku pa restavrira staro prekinitveno masko. Zadeva se seveda še malo zaplete, ker sta v našem primeru dve prekinitveni maski, od katerih je ena naslovljiva samo po posameznih bitih, poleg tega pa je treba voditi še evidenco o življenju oddajnega procesa (ki seveda ni vedno živ).

ZAKLJUCEK

Razvojno delo na PMT-100 je dalo videoterminal, ki je glede na zasnovo, majhno število integriranih vezij in uporabljene domače tipke cenejši od običajnih VT100-skladnih terminalov. Pri tem ima standardno vgrajene spsosobnosti kot VT100 z AVO opcijo (132 x 80 znakov, vsi znakovni atributi), jugoslovanski nabor znakov, mehak pomik (soft scroll) in celostranski pregleden "setup". Vgrajeno ima možnost izpisovanja krmilnih znakov in zaporedij brez njihove interpretacije. Vgrajena je semigrafika 160 x 96 (264 x 96).

Kvaliteta emulacije (VT52 in VT100) je bila preskušena na vrsti igric (ki so najboljši test za take stvari). Pri tem je bila pokazana popolna skladnost z VT100, kar nikakor ne bi mogli trditi za nekatere druge tovrstne terminale na našem tržišču.

Če odštejemo nekaj začetnih tavanj in standardne usluge (izdelava tiskanine), sta na projektu terminala delala dva človeka.

Efektiven čas na projektu:
- razvoj materijalne opreme in spodnjega nivoja programske opreme: 1 človek, 4 meseci
- razvoj zgornjega nivoja programske opreme (emulacija, setup, ...): 1 človek, 2 meseca

Tako kratek čas je bil mogoč iz naslednjih razlogov:
- enostavna realizacija materijalne opreme, praktično ni mehanskih montažnih problemov.
- uporaba visokega programskega jezika (C) pri programiranju z izjemo časovno kritičnih delov.

Pri tem velja omeniti, da ni bilo na voljo skoraj nikakršne razvojne opreme (nobenega in-circuit emulatorja). Uporabljeni C prevajalnik je izdelek firme Whitesmith in teče na računalniku LSI-11 pod RT-11 ali SHARE.

DODATEK: ZAKAJ JEZIK C

Jezik C se vse bolj uveljavlja, vendar še ni povsem udomačen in nekaj besed o njem ne bo škodovalo. Nepoučenemu pusti pogled na program v C vtis zmede, ki je kaj malo podobna recimo Pascalu. Vendar je ta razlika samo navidezna in izvira iz neobičajnih označb za operatorje in nekatere simbole. Vlogo pascalskih begin/end besed imajo tukaj recimo zaviti oklepaji. Ko odmislimo te razlike, sodi C v isto skupino kot PASCAL. V čem je torej razlika in zakaj je C primeren za pisanje sistemske programske opreme (recimo operacijskega sistema UNIX).

Predvsem: C nima nikakršnih konstruktov za programiranje paralelnih procesov (recimo obdelava prekinitvnih zahtev) in je v tem enakovreden PASCALU. Pač pa ima C vgrajen nabor operatorjev za boolove operacije in pomikanje (shift), kar je nujno potrebno za sistemsko programiranje. V PASCALU je to izvedljivo s klicanjem ustreznih procedur, kar pa je nujno neučinkovito. Marsikdaj je koristno tudi dejstvo, da obvlada celoštevilčno računanje na 32 bitov. Kontrole nad tipi spremenljivk, številom in tipom argumentov v procedurah, velikostjo nizov ipd. sploh ni. Zaradi te kontrole v PASCALU ni moč definirati ekvivalenta PASCAL-skega stavka WRITE. V C-ju je to možno, zato tak stavek v standardni jezik C sploh ni vgrajen. Ta odsotnost kontrole omogoča storiti marsikaj neprecenljivo koristnega, neizkušenemu programerju pa še več usodno in skrivnostno napačnega.

Lahko bi nadaljevali, vendar strnimo na kratko. Jezik C omogoča vrsto stvari (pravzaprav malenkosti), ki so koristne ali nujne za sistemsko programiranje in ki jih PASCAL in podobni jeziki običajno ne omogočajo, vsaj ne na učinkovit ali standarden način. Vendar pa je s svojo pomankljivo notranjo kontrolo lahko tudi nevarna past za neizkušene programerje. Za programiranje jedra operacijskega sistema ali vhodno izhodnih procedur pa se je v splošnem še vedno treba v večji ali manjši meri zateči k zbirniku.

TRINIVOJSKA ARHITEKTURA INFORMACIJSKIH SISTEMOV PO MODELU ISO TC97/DP 9007

B. Jerman-Blažič, I. Fabič
Institut Jožef Stefan, Ljubljana

UDK: 681.3(497.1)

Abstract

Three level architecture of an information system according to the model of ISO/TC97/DP 9007

The paper deals with the three level architecture of an information system designed according to the model defined in ISO/TC97/DP 9007. Previously this model was known under the name ANSI/X3/SPARC DBMS Framework.The model was widely accepted in the fields of data base design and systems analysis. The model is set up on three level architecture, i.e three schema framework. The conceptual schema comprises a unique central description of the various information contents that may be in a data base of an information system. This includes the description of what actions, such as changes and retrievals are permissible on the information content. The data base itself may be implemented in any one of a number of possible ways. Users and application programs may be view the data in a variety of ways, each described by an external schema. Each external schema is therefore derived from the common conceptual schema. The physical storage structure that may be in use at any given time is described by an internal schema that is also derived from the conceptual schema.

Povzetek

V prispevku je podana trinivojska arhitektura informacijskega sistema, zgrajena po modelu ISO/TC97/DP 9007. Model je nastal v okviru ANSIja in zato je znan tudi pod imenom ANSI/X3/SPARC koncept podatkovne baze. Model se je uveljavil zlasti na področju načrtovanja podatkovnih baz in sistemske analize. Model je zasnovan na konceptu treh nivojev ali treh shem. Konceptualna shema je enotni, centralizirani zapis vseh informacij, ki nastopajo v podatkovni bazi sistema. Poleg zapisa informacij v konceptualni shemi so zapisane vse dovoljene in nedovoljene akcije nad podatki. Implementacijo podatkovne baze lahko izvedemo na več načinov. Kako uporabniki vidijo podatkovno bazo od zunaj, zapišemo v zunanji shemi. Fizičen zapis podatkov na hranilnih medijih je opisan v notranji shemi. Skupen pregled nad vsebino zunanje in notranje sheme se hrani v konceptualni shemi.

1.Uvod

Trendi v razvoju informacijske tehnologije in sistemov za obdelavo podatkov kažejo na uvajanje različnih pripomočkov, ki imajo za namen poenostaviti delo uporabnikov, oziroma oblikovati delo z računalnikom na najbolj human način. Za te pripomočke je značilno, da so načrtovani z najvišjo stopnjo abstrakcije, ki omogoča posplošitev procesa izdelave programske opreme. Visoka stopnja abstrakcije omogoča z druge strani tudi standardizacijo programske opreme, kar samo po sebi prinaša ekonomičnost v procesu proizvodnje, poenotenost izdelkov, njihovo lažjo distribucijo ter lažje vzdrževanje. Standardizacija izdelave programske opreme dovoljuje tudi enostavno prilagajanje izdelkov glede na potrebe različnih naročnikov. Med temi prizadevanji za standardizacijo izdelave programske opreme zavzema poleg že znanih in uveljavljenih metod pomembno mesto metoda za izdelavo standardnih podatkovnih modelov in podatkovnih slovarjev v informacijskih sistemih, znana pod imenom KONCEPTUALNA SHEMA in MODEL TRINIVOJSKE ARHITEKTURE (Conceptual schema and three level architecture ali skrajšano CS and TLA). Metodo so razvili najprej v okviru ANSIa, zatem je bila sprejeta s strani ISOa TC97/SC21 1.1985. Naloga, ki je bila zastavljena v okviru odbora za plan ANSI, komite X3, je zajemala naslednje točke:

1.Definirati koncepte CSL (Conceptual Schema Language), kot orodje, ki avtomatično opiše miselni model in poenostavi testiranje pri razvoju in uporabi informacijskih sistemov;

2.Opredeliti definicije v konceptih za CSL;

3.Razviti metodologijo za oceno predlogov za jezik CS, ki naj omogoča formalen opis informacijskih zahtev uporabnikov;

4.Oceniti prispele predloge za jezik CS;

5.Definirati koncepte različnih stopenj abstrakcije CS;

6.Opredeliti možnosti uporabnikov pri CS;

7.Razviti koncepte in terminologijo za uporabo CS in njeno vključitev v referenčni model OSI (Open System Interconnection) v sodelovanju z ISO/TC97/SC21.

2.Osnovni principi CS in Informacijska baza

V tem poglavju bomo predstavili problematiko CS in TLA in podali osnovne informacije o uporabljeni terminologiji in konceptih informacijskega modela.

2.1 Informacijski sistemi

Osnovna zahteva vsake človeške družbe je informiranje ali posredovanje informacij. Ta temeljna zahteva vsake sodobne družbe poteka ob uporabi določenega znanja pod pogojem, da je to znanje razpoložljivo. Proces pridobivanja znanja ne poteka individualno, ne učimo se na poskusih in na napakah, ki jih naredimo, ampak z **informiranjem**. To pomeni, da se učimo le, če od nekod ali od nekoga dobimo informacijo. V tem kontekstu pomeni informacija "znanje" o nečem, o stvareh, dejstvih, konceptih, metodah ipd. To znanje lahko izmenjujemo. Proces izmenjave informacij v sodobni družbi poteka najpogosteje v tehničnem okolju.

Vsak proces izmenjave informacij je zasnovan na sledečih dejstvih:

-**dobavitelj** informacije (pošiljatelj) in **uporabnik** (prejemnik) nista nikoli na isti lokaciji, torej potrebuje vsaka komunikacija funkcijo za **premostitev razdalje** t.j. **komunikacijsko funkcijo**;

-**dobavitelj** informacije in **uporabnik** ne obravnavata nikoli iste informacije hkrati, torej potrebuje vsak komunikacijski proces funkcijo za **premostitev časa** t.j. **spominsko funkcijo.**

Dobavitelja in uporabnika lahko obravnavamo kot dve komponenti enega podjetja. Tretja komponenta tega podjetniškega sistema, ki nam pomaga urejati probleme pri izmenjavi informacij, je **informacijski sistem.** V tem kontekstu je informacijski sistem tak sistem, ki pomaga uporabnikom pomniti in pošiljati oz. sprejemati informacije.

Načrtovanje informacijskih sistemov je bilo do sedaj usmerjeno k uvajanju učinkovitih metod za obdelavo podatkov. Interpretacija podatkov je bila prepuščena uporabniku, da je to opravil tako, kot je lahko znal oz. zmogel. Osnovni namen izmenjave informacij je posredovanje znanja, to pomeni, da prejemnik sprejme in razume informacijo tako, kot jo razume pošiljatelj. Predstavitev *informacije* (oblika podatkov) je sekundarnega *značaja*. Podatke lahko hranimo, delamo in manipuliramo z njimi le, če obstaja jasna in natančna definicija oziroma sporazum, kaj ti podatki predstavljajo in katera semantična pravila zanje veljajo.

2.2 Okolje

Namen informacijskega sistema je, da omogoča izmenjavo informacij med različnimi komponentami enega aktivnega sistema. Komponente sistema, ki uporabljajo informacijski sistem, sestavljajo **okolje** informacijskega sistema. Povedano z drugimi besedami: vsi "uporabniki" informacijskega sistema oblikujejo "okolje" tega sistema. Uporabnik je vsak objekt, ki dela in komunicira z informacijskim sistemom, to je lahko človek, stroj ali drug informacijski sistem.

Uporabnik komunicira z informacijskim sistemom s pomočjo **sporočil**. Z gledišča informacijskega sistema je okolje IS (informacijskega sistema) vir in kraj, kamor sporočila odhajajo oziroma prihajajo. IS je popolnoma statičen in pasiven do prihoda sporočila, na katerega reagira. IS ne vzpodbuja nobene akcije sam po sebi. Dela le tako, kot to od njega zahtevajo uporabniki.

Uporabnike omejujejo pravila, ki določajo, kaj je dovoljeno zahtevati od informacijskega sistema oziroma kaj mu lahko ukažejo. Nekaterim uporabnikom je dovoljeno spreminjati informacije v informacijskem sistemu, nekaterim (ponavadi je to število majhno) pa je dovoljeno spreminjati tudi obliko IS ali posamezne dele IS.

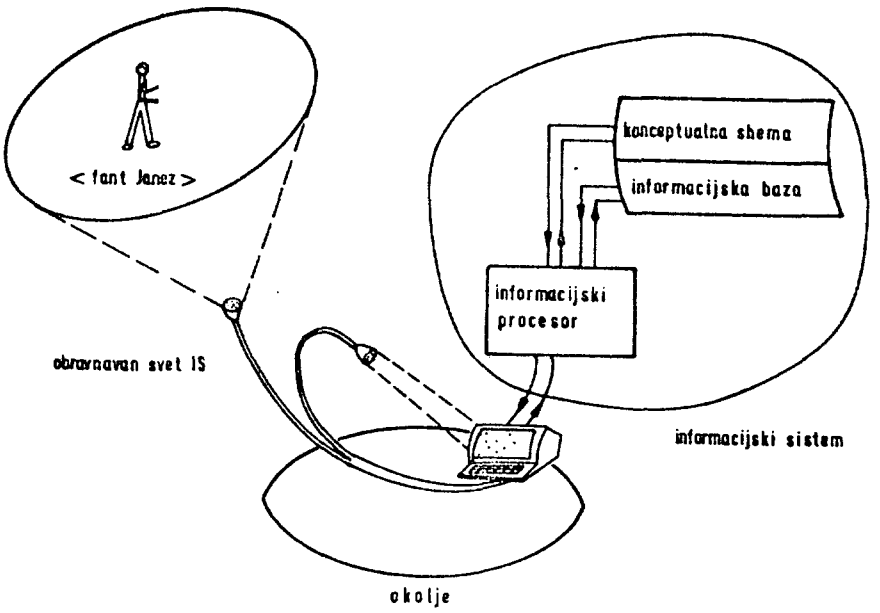
2.3 Obravnavani svet

IS so sistemi, ki omogočajo izmenjavo informacij (tukaj nam informacija pomeni dejstvo oz. resnico o nekaterih rečeh - predmetih, poslih ali osebah). Zbirka predmetov, oseb, poslov ali z eno besedo zbirka, za katero potrebujemo informacije, imenujemo "obravnavan svet" (Universe of Discourse) informacijskega sistema ali na kratko **"svet IS"**. Svet IS je del realnega sveta in je lahko v nekaterih primerih le eno skladišče ali celotna organizacija enega podjetja.

Tipični svet IS razumemo kot zbirko, sestavljeno iz realnih in abstraktnih predmetov, imenovanih **nosilci informacij** (entities) ali na kratko nosilci. Bolj natančno: svet IS je sestavljen iz razredov nosilcev kot so na primer: osebje, oddelki, datumi ipd. Izbor značilnosti za razvrstitev nosilcev v razrede je lahko poljuben, najboljša rešitev je ponavadi pragmatična in je pogojena z namenom uporabe posamezne zbirke. Nekatere splošne značilnosti pa le vplivajo na samo razvrstitev, tako na primer osebje ne more biti oddelek, ena oseba ne more pripadati več kot enemu oddelku ipd. Neformalno te **relacije** med elementi sveta IS opisujemo kot "razvrstitve", "pravila", "določila" ali "omejitve" glede vedenja ali dela z nosilci v svetu IS. Na splošno je to, kar pripada svetu IS, časovno spremenljivo ali povedano z drugimi besedami: izbrani predmeti, osebe in dejanja v svetu IS se spreminjajo s časom. Enako velja za razvrstitve, pravila, določila ipd. Vsekakor je hitrost sprememb teh zadnjih nekajkrat manjša kot hitrost, s katero se **spreminjajo nosilci** v svetu IS.

2.4 Poslovni sistem

Poslovni sistem sestavljajo: svet IS, okolje in informacijski sistem tako, kot nam kaže slika 1.



Slika 1. Okolje in IS

Svetu IS pripadajo vsi predmeti, osebe in posli, **o katerih** zahtevamo informacije, v okolju so vsi uporabniki, **za katere** te informacije zahtevamo; informacijski sistem vsebuje mehanizme, **s katerimi** te informacije hranimo in obravnavamo.

V literaturi je pojem "informacijski sistem" uporabljen kot sinonim za računalniško zasnovan poslovni sistem določenega podjetja. To pomeni, da so v sistemu združeni uporabniki informacij skupaj z aktivnostmi in postopki, ki spremljajo sleherno **uporabo** informacije. V kontekstu dokumenta DP9007 je pojem "informacijski sistem" omejen na komponento poslovnega sistema, ki se uporablja le kot orodje za izmenjavo informacij (sprejetih, shranjenih in obdelanih). V zvezi s tem je informacijski sistem ločen od okolja na sledeči način:

a)informacijski sistem je formalni sistem, okolje pa ni, če ga opazujemo kot celoto;

b)vedenje informacijskega sistema je popolnoma določeno s pravili in z omejitvami, postavljenimi direktno ali indirektno iz samega okolja;

c)informacijski sistem je popolnoma predvidljiv in zaradi tega ne odstopa od zastavljenih pravil ali omejitev. Okolje pa lahko odstopa.

2.4.Konceptualna shema in informacijska baza

Izmenjava informacij je mogoča le, če obstaja obojestransko razumevanje in natančen dogovor o tem, iz česa je sestavljen "svet IS". Nasprotno temu je neuspešna komunikacija skoraj vedno rezultat nesporazuma o nosilcih v svetu IS. Komunikacija mora biti natančna in nedvoumna posebno v primeru, ko uporabniki komunicirajo preko računalnika. Računalniki razumejo le modele (opise) realnega sveta in ne sam realni svet; zato da bi razumeli "svet IS", potrebujemo model "sveta IS".

Model "sveta IS" imenujemo **Konceptualna Shema - Conceptual Schema**. CS opisuje nosilce v "svetu IS", in sicer nosilce informacij, ki so obstajali, ki sedaj obstajajo in ki bodo obstajali. Istočasno opisuje CS vsa možna dejstva in dogodke, ki se nanašajo na nosilce. Pri opisu sveta IS definira sistemski analitik najprej dejstva, kot so razvrstitve in pravila. Pri identifikaciji dejstev se pogovarja in posvetuje z uporabniki. Ko dejstva identificira in jih zapiše, zgradi "okostje opisa" sveta IS ali osnutek CS.

Najbolj pomembna sistema glede na model CS in TLA sta svet IS in računalniški sistem, ki vsebuje lingvistično predstavitev sveta IS. Informacije o svetu IS "opisujejo" ali "modelirajo" svet IS.

Proces opisovanja sveta IS je lahko zelo zahtevna naloga, ki zahteva kreativno analizo in večkratno popravljanje in izboljševanje opisa. Konkretna fizična predstavitev informacije, ki opisuje svet IS, imenujemo **podatkovna baza**. Pod pojmom **sistem za delo s podatkovno bazo** ali na kratko DBS (data base system) razumemo sistem za obdelavo podatkov, ki obdeluje to podatkovno bazo.

Splošno gledano pristop CS dovoljuje, da je tudi DBS del sveta IS, oziroma eden od elementov, ki jih opisujemo. Zaradi enostavnosti je DBS najpogosteje obravnavan kot del, ki ne pripada svetu IS. Koncept TLA predpostavlja, da je formalen opis CS vsebovan v DBS Vse dodatne informacije, ki opisujejo "nosilce" iz "sveta IS"in njihova stanja v danem trenutku, predstavljajo **"informacijsko bazo"** informacijskega sistema.

2.5. Princip treh nivojev

Opis sveta IS podajamo v obliki, ki omogoča komunikacijo, to pomeni, da nas najbolj zanima interpretacija predstavitve. Sama oblika predstavitve v tem kontekstu je sekundarnega pomena. Pojem **informacija** pomeni, da se nanaša na interpretacijo opisa sveta IS. Pojem **podatek** pa se bo nanašal le na obliko informacije. Za vsak IS so najbolj pomembni trije vidiki obravnave:

a)**za kaj gre**;

b)**kakšen** je od zunaj;

c)**kako** je narejen;

Prvi vidik se nanaša na to, **kakšne** informacije hranimo in obdelujemo s sistemom in **kaj** je svet IS. Ta vidik imenujemo **konceptualni vidik** informacijskega sistema. Drugi vidik se nanaša na to, **kako** podatki predstavljajo informacije, oziroma **kako** so ti podatki predstavljeni v informacijskemu sistemu in **kako** te podatke iščemo v informacijskem sistemu. To je **zunanji vidik** informacijskega sistema. V žargonu AOPa to pomeni "definicija podatkovnega modela" in "manipulacija s podatki". Ta dva vidika sta najbolj pomembna za uporabnika. Za razvijalca sistema je pomemben predvsem tretji vidik. To je: **kako** so podatki, ki predstavljajo določene informacije, shranjeni in obdelani v računalniškem sistemu in kako je varovana njihova integriteta oziroma kakšna je njihova varnost. To je tki. **notranji vidik** informacijskega sistema. Ta vidik je za končnega uporabnika neviden (transparenten).

Gledano zgodovinsko, so informacijske sisteme načrtovali predvsem s stališča zunanjega in notranjega vidika IS. Implementacija informacijskega sistema je potekala tako, da so najprej opravili analizo podatkov in njihove uporabe in zatem izdelali podatkovni model, ki je opisoval relacije med podatkovnimi elementi in manipulacijo z njimi. Ta metoda je učinkovita le v nekaterih primerih. Sama metoda temelji na dveh predpostavkah:

1.vedenje sveta IS lahko določamo na podlagi vedenja (uporabe) podatkov;

2.informacijske potrebe sedanjih in bodočih uporabnikov lahko določimo na podlagi tekočih in željenih oziroma zahtevanih podatkov.

Obe predpostavki sta sami po sebi zelo vprašljivi, kljub dejstvu, da so te trditve predstavljale temelj vseh metod načrtovanja informacijskih sistemov v sedemdesetih letih. Ne bi bilo prav, če bi zatrjevali, da je posnetek stanja na podlagi tekočih podatkov in manipulacij nekoristen. Vsekakor pa ta postopek ne vodi v razvoj sistemov z dolgo življenjsko dobo. Tekoči podatki odražajo tekoče organizacijske sheme, za te pa je značilno, da so časovno spremenljive. Tudi če se zgodi, da posnetek stanja dobro opredeli nosilce v svetu IS, ostanejo informacijska pravila najpogosteje neodkrita zaradi tega, ker jih podatki ne vsebujejo in ne odražajo. Najbolj pogosto uporabljene tehnike za izdelavo podatkovnih modelov (ki vključujejo tudi različne normalizirane oblike v relacijskih podatkovnih modelih) nimajo možnosti, da izrazijo **znotraj samega modela** vse vrste pravil, na katere naletimo, ko analiziramo realni svet, a ne pretok podatkov.

Danes kaže, da je prišlo do spoznanja, da je **konceptualni vidik** najbolj pomemben. Dokaz tega je sprejetje dokumentov DP 9007 s strani ISOa (International Standard Organization) To pomeni, da moramo najprej ugotoviti, katere informacije bomo obravnavali z informacijskim sistemom. Šele zatem bomo določali, kako bo ta informacija predstavljena kot podatek in kako jo bomo obdelovali, kakšna bo notranjost informacijskega sistema, ki bi ustrezno pokrila potrebe uporabnikov po mehanizmih za hranjenje in manipuliranje s podatki. Pristop k načrtovanju IS, ki upošteva predvsem pomen informacije, je danes znan kot **Trinivojski pristop** načrtovanja. Ta pristop ima sledeče tri komponente:

konceptualni nivo

zunanji nivo

notranji nivo

Vse, kar se nanaša na pomen in interpretacijo informacije, je del **konceptualnega nivoja**. To vključuje: specifikacije, manipulacije in krmiljenje z informacijami v smislu njihovega pomena. Formalni jeziki za opis konceptualne sheme so orodje, s katerim opisujemo konceptualni nivo IS.

Zunanji nivo se nanaša na vse, kar ima opraviti s končnim uporabnikom, oziroma z vsem, kar končni uporabnik vidi. To se nanaša predvsem na predstavitev informacij v zunanjih vmesnikih, t.j. vmesnikih, ki se nahajajo med informacijskim sistemom in okoljem. To vključuje predstavitev informacije, t.j. podatka na način, ki najbolj ustreza končnemu uporabniku in seveda vse računalniške komponente, udeležene pri manipulaciji uporabniško-orientiranih podatkov.

notranji nivo zajema vse vidike računalniške implementacije. Najpogosteje je ta komponenta IS nevidna za končnega uporabnika. Notranji nivo sestavljajo:

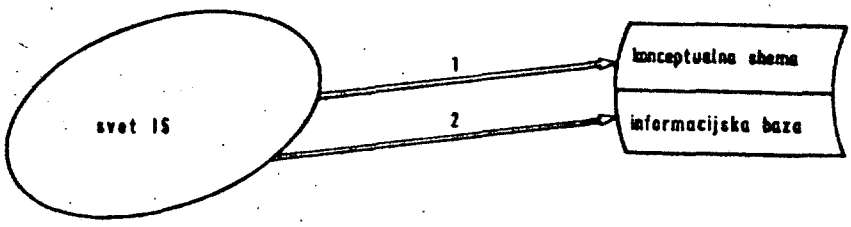
1)interna (fizična) predstavitev informacij v računalniku in v pomnilniških medijih;

2)učinkovitost računalniških procesov in učinkovitost mehanizmov dostopa do shranjenih podatkov;

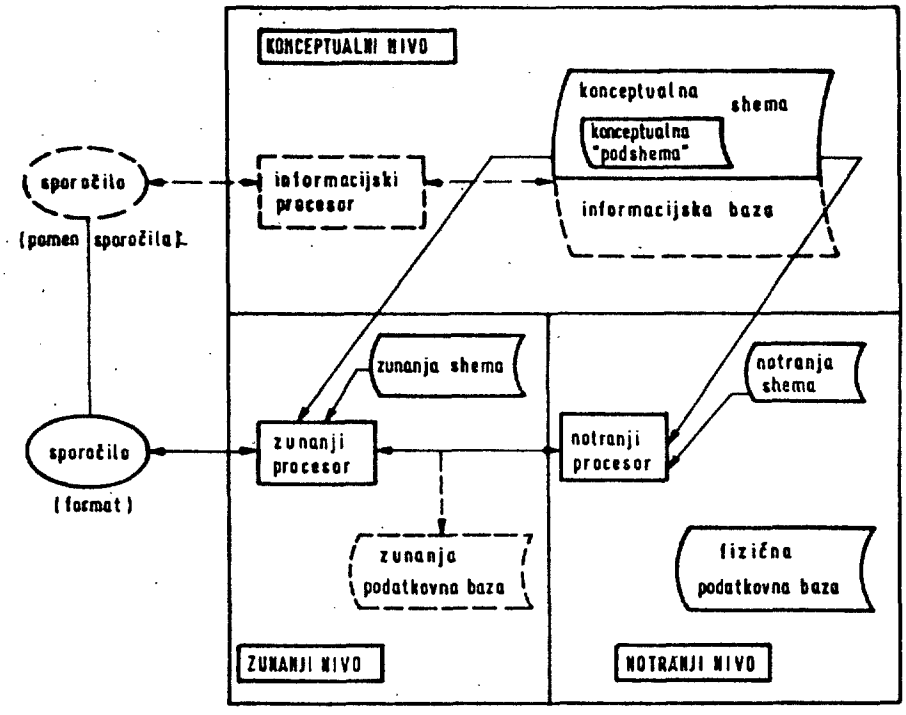
3)nadzor nad vzporedno uporabo istih podatkov, mehanizmi za obnavljanje (recovery) ipd.

Predlagatelji modela CS in TLA poudarjajo, da beseda **nivo** ne pomeni, da so vse tri komponente v kakšni hierahični odvisnosti. Razširjena uporaba imena tega modela ne dovoljuje dodatnega spreminjanja terminologije. Vsi dosedanji sodelavci pri razvoju modela so si bili enotni glede tega, da beseda nivo veliko bolje odraža koncept modela kot pa beseda "sfera" ali "domena" (realm v angleščini), ki naj bi nadomestila termin nivo.

Če strnemo vse, kar je bilo povedanega, lahko zapišemo, da smo identificirali sledeče pojme:



Slika 2. Proces opisovanja IS



Slika 3. Trinivojska arhitektura

svet IS
Zbirka objektov (nosilcev), ki so, ali ki so kadarkoli bili, oziroma bodo sestavni del izbranega dela realnega sveta, ki bo obravnavan v IS;

konceptualna shema
Opis možnih stanj sveta IS, ki vključuje tudi razvrstitve, pravila, omejitve ipd. v svetu IS;

informacijska baza
Opis nosilcev, ki v danem trenutku obstajajo v svetu IS in njihova dejanska stanja.

Proces, s katerim opišemo svet IS, je dvokomponenten in je ponazorjen na sliki 2.

1) Opis in zapis vseh razvrstitev, abstrakcij, posplošitev in vzpostavitev pravil o svetu IS. To je mentalni proces, ki poteka v glavah razvijalcev;

2) Zapis dejstev in dogodkov iz sveta IS in potrebnih nosilcev na formalen način.

CS vsebuje splošna pravila, ki opredeljujejo obnašanje sveta IS. Ta pravila določajo, kaj se lahko in kaj se ne sme zgoditi v informacijski bazi. **Informacijski procesor** vsebuje mehanizme za manipulacijo z vsebino informacijske baze. Spremembe, ki jih opravlja nad informacijsko bazo in v CS se opravijo na podlagi sprejetih sporočil. Vsebina sporočil so ukazi ali določene informacije in te prihajajo iz okolja. Informacijski procesor je brez lastne iniciative in se vede le tako kot določajo pravila. To zlasti poudarjamo, ker imajo po principih ISO TC 97/DD9007 svoj delež v informacijskem procesorju tudi imajo manualni posegi.

2.6 Trinivojska arhitektura

Na podlagi opisanega trinivojskega pristopa načrtovanja informacijskih sistemov je zgrajena **Trinivojska arhitektura**, ki je prikazana na sliki 3.

Manipulacijo z vsebino informacij opravljamo na konceptualnem nivoju z informacijskim procesorjem. Informacijski procesor je sestavljen iz materialne in programske opreme ter ljudi. Informacijski procesor izpelje spremembe ali iskanje informacij za potrebe uporabnikov iz okolja IS. V računalniško zasnovanem informacijskem sistemu je informacijski procesor dejansko računalnik skupaj s sistemsko programsko opremo, programsko opremo za delo s podatkovno bazo in aplikacijskimi programi. Informacijska baza je implementirana kot vsebina podatkov v podatkovni bazi (PB) skupaj s programi za generiranje sekundarnih podatkov iz vsebine PB. Pravila za delo informacijskega procesorja so delno implementirana v podatkovnih strukturah PB, delno v rutinah sistema za DBS in delno v aplikacijskih programih. Pojem informacijskega procesorja je v Trinivojski arhitekturi podan precej splošno. Edini pogoj, ki je postavljen pred informacijskim procesorjem je, da deluje aktivno pri manipulaciji z informacijami v skladu z določenimi pravili. To velja za ljudi in za tehnične sisteme. Pod manipulacijo je mišljeno sledeče:

Informacijskemu sistemu je poslano sporočilo, ki vsebuje nove informacije za vnos v informacijsko bazo. Informacijski procesor bo na podlagi pravil, določenih v CS in mogoče v drugih stavkih, zakodiranih v informacijski bazi, informacije sprejel ali jih bo zavrnil in istočasno poslal sporočilo o rezultatih te akcije. Vse ostale akcije informacijskega procesorja delujejo na podoben način.

V informacijskemu procesorju lahko delujejo računalniški sistemi tudi kot uporabniki. Tak primer je mreža računalniških sistemov, ki komunicirajo med seboj. V primerih, ko vsak sistem deluje na podlagi množice neodvisnih pravil, je vsak sistem uporabnik drugega. Iz tega sledi, da je opredelitev, ali je kakšen sistem uporabnik ali informacijski procesor, odvisna od vloge, ki jo ima sistem in ki jo določajo zastavljena pravila obnašanja.

V praksi je uporabnik v glavnem zainteresiran le za svoje potrebe, za svoj svet IS. To pomeni, da je zainteresiran za dele CS in IB, za katere je že prej določil, da mu ustrezajo, oziroma da jih potrebuje. Ponavadi imamo več uporabnikov takšnih svetov in jih zaradi tega združujemo v CS s širšo zasnovo. Tako na primer potrebe večjega števila oddelkov združimo v skupino s podobnimi potrebami oziroma "podsveti". Na podlagi tega opišemo CS zatem kot "unijo" skupin, oziroma kot unijo različnih "podsvetov", ki imajo ekvivalent vsak v svoji "podshemi".

Na zunanjem nivoju je za razliko od konceptualnega zelo pomembna prav oblika za predstavitev informacij, zato so formati podatkov v notranji shemi natančno opredeljeni. Formate podatkov določamo glede na potrebe naročnikov oziroma uporabnikov. V kontekstu CS in IB to pomeni, da je predstavitev podatkov prilagojena različnim uporabniškim procesom. Ta posebni, zunanji videz informacij in njihov pomen je opisan v **zunanji shemi**. Sami formati pa so podani v **zunanji podatkovni bazi**. Z ozirom na to, da imamo več "podsvetov", ima vsaka uporabniško prirejena skupina informacij, ki pripada določenem uporabniškem procesu, preslikavo v eni ali več zunanjih shem. V vsaki od teh shem je definirana oblika predstavitve in opis zunanje podatkovne baze, za katero predpostavljamo, da odraža vsebino in obseg tega uporabniškega procesa. Vsaka od teh zunanjih podatkovnih baz je **navidezna** in je dejansko preslikana v odgovarjajoči del informacijske baze celotnega sistema. Takšna zgradba pa zahteva sledeče:

Na zunanjem nivoju je za razliko od konceptualnega zelo pomembna prav oblika za predstavitev informacij, zato so formati podatkov v notranji shemi natančno opredeljeni. Formate podatkov določamo glede na potrebe naročnikov oziroma uporabnikov. V kontekstu CS in IB to pomeni, da je predstavitev podatkov prilagojena različnim uporabniškim procesom. Ta posebni, zunanji videz informacij in njihov pomen je opisan v **zunanji shemi**. Sami formati pa so podani v **zunanji podatkovni bazi**. Z ozirom na to, da imamo več "podsvetov", ima vsaka uporabniško prirejena skupina informacij, ki pripada določenem uporabniškem procesu, preslikavo v eni ali več zunanjih shem. V vsaki od teh shem je definirana oblika predstavitve in opis zunanje podatkovne baze, za katero predpostavljamo, da odraža vsebino in obseg tega uporabniškega procesa. Vsaka od teh zunanjih podatkovnih baz je **navidezna** in je dejansko preslikana v odgovarjajoči del informacijske baze celotnega sistema. Takšna zgradba pa zahteva sledeče:

integracijo akcij različnih uporabnikov;

transformacijo zunanjega videza v skupni **konceptualni videz**, znan celotnemu informacijskemu sistemu.

V primerih, ko je zunanji videz unija več videzov (na primer, če so zunanje podatkovne baze več individualnih oddelkov grupirane v skupno zunanjo podatkovno bazo na skupnem nivoju), bo zunanja shema zajela vse posamezne individualne zunanje sheme. Tako združena zunanja shema je dejansko le opis skupne podatkovne baze, ki ima **poenoteno zunanjo obliko**. Vse funkcije zunanje sheme vzdržuje in nadzira zunanji informacijski procesor.

Informacije so predstavljene v računalniku kot fizični podatkovni elementi, kot so zapisi, segmenti, polja in pd. in so del fizične podatkovne baze. Ti formati so zapisani v notranji shemi.

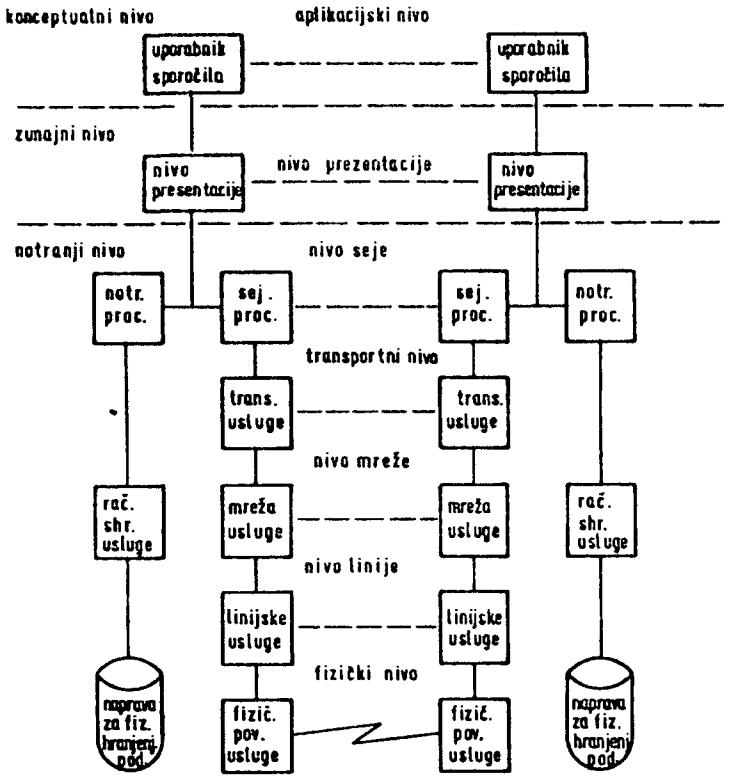
Zunanja podatkovna baza je preslikana v **fizično podatkovno bazo**. Najpogosteje se ta preslikava opravi na eni PB. Po principih ISO DP9007 to ni omejitev, preslikava več zunanjih baz je možna na več podatkovnih bazah ali obratno, eno zunanjo podatkovno bazo lahko preslikamo na več podatkovnih baz. Dovoljeni so tudi porazdeljeni sistemi. Notranji procesor opravi transformacijo iz zunanje v notranjo obliko. V primeru distribuiranih podatkovnih baz opišemo povezave med notranjo in zunanjo podatkovno bazo s porazdeljeno shemo, ki je lahko del poenotene zunanje sheme ali pa je podana kot vmesnik med zunanjo in notranjo shemo. Naloge informacijskega procesorja implementiramo kot množico "postopkov" ali procedur. Ni rečeno, da opravlja vse te naloge le en procesor, tako so v primeru distribuirane obdelave lahko procedure porazdeljene med odgovarjajočimi zunanjimi in notranjimi procesorji. Pri vsem tem ostane najbolj pomembno, da konceptualna shema nadzira, **kaj je** opisano v informacijski bazi in ne, **kako** je opisano. Konceptualna shema nadzira semantični pomen vseh predstavitev, to pomeni, da definira vse dovoljene akcije nad vsebino podatkov (set of checking, generating and deducing procedures). Vmesna stanja (intermediate states) v procesu transformacije med zunanjo in notranjo obliko niso opisana. Zunanja shema opisuje, kakšna je oblika informacij, ki ustreza uporabnikom. Zunanji procesor direktno komunicira z uporabniki in koordinira izmenjavo informacij. Notranja shema opisuje interno fizično predstavitev informacije. Transformacijo med zunanjo obliko in notranjo obliko opravi notranji procesor. To pomeni, da zunanji procesor

komunicira z notranjim procesorjem. Preslikava med zunanjo in notranjo shemo ohranja pomen informacij, tako kot je to določeno v CS.

Glede na principe Trinivojske arhtekture dovoljujeta zunanja in notranja shema večnivojsko strukturo v notranjem in zunanjem procesorju. Notranja podatkovna baza je lahko implementirana kot družina internih podatkovnih baz, v katerih je shranjen del informacijske baze. V nekaterih primerih se lahko te baze prekrivajo (porazdeljeni sistemi).

in sinhronizira njihov dialog oziroma njihovo izmenjavo (zunanjo) podatkov. Za izvajanje teh funkcij vzpostavlja nivo seje povezave med nivoje predstavitve dveh enot in podpira potek izmenjave podatkov. Transportni nivo ter ostali trije nižji nivoji skrbijo za tehnične pogoje povezovanja računalnika, mreže, komunikacij, ki jih potrebuje nivo predstavitve.

Korelacija med trinivojsko arhitekturo in referenčnim modelom OSI je prikazana na sliki 4.



Slika 4. Nivojska struktura referenčnega modela v povezavi s trinivojsko arhitekturo

2.7. Model odprtih sistemov povezovanja in trinivojska arhitektura

Referenčni model ISOa opredeljuje koncepte komuniciranja informacijskih sistemov (aplikacijski nivo) ob uporabi mehanizmov za povezovanje odprtih sistemov. Model je razdeljen v sedem funkcionalnih delov: uporabniški nivo, nivo prezentacije, nivo seje, nivo prenosa, nivo linije in fizični nivo. Uporabniški nivo ima vlogo okna oziroma okvirja, skozi katerega poteka izmenjava informacij z identificirano vsebino v času trajanja komunikacije med dvema uporabnikoma. Nivo predstavitve skrbi za predstavitev informacij (v zunanji obliki predstavitve) in za ohranitev vsebine informacij pri sintaksni analizi podatkov v času komunikacije med dvema uporabnikoma. Funkcija nivoja seje je, da preskrbi potrebne pogoje za nemoteno sodelovanje dveh enot na ravni predstavitve ter da organizira

Konceptualni in zunanji nivo trinivojske arhitekture sta povezana s funkcijami nivoja aplikacije in nivoja predstavitve. Za pričakovati je, da bodo koncepti, razviti v okviru konceptualne sheme, implementirani v semantiko informacijskega prometa v okviru teh dveh najvišjih nivojev. Točne korelacije in povezave med konceptualnim in zunanjim nivojem CS in IB z ene strani, ter funkcij aplikacijskega nivoja in nivoja predstavitve v modelu OSI, so še naprej predmet raziskovanj in natančnih specifikacij v okviru delovnih skupin ISOa.

Notranji nivo ima opravka z notranjo predstavitvijo podatkov, z obdelavo podatkov ter z dejanskim shranjevanjem podatkov na fizičnih medijih. Značaj teh operacij lahko primerjamo z značajem, ki ga imajo nivoji 1 do 5 v modelu ISO. Pri tem moramo seveda opozoriti, da so njihove funkcije (shranjevanje in komu-

nikacije) popolnoma različne. Skupne raziskave (CS in IB ter ISO) so trenutno usmerjene k:

* študiju stopnje korelacij, koincidence in uporabnosti konceptov ISO na informacijskih sistemih, tako kot je to prikazano na sliki 4;

* študiju uporabnosti (izbrane segmente) konceptov OSI pri razvoju **notranjih** komunikacij v informacijskih sistemih (porazdeljeni informacijski sistemi).

je zadoščeno z definicijo "macrojev". Izbor teh macrojev je odvisen od vrste aplikacij. V opisu sveta IS se sklicujemo na nosilce informacij z imeni. To zahteva previdnost glede uporabe sinonimov in uporabe hononimov. CS in IB vsebujeta opise stanj v svetu IS in hkrati njihovo evolucijo, zato morajo uporabljeni konstrukti omogočati hkrati opis nosilcev informacij in manipulacij z njimi. Meje in vsebino CS določajo razvijalci informacijskega sistema, pomoč in vodilo pri tem predstavljajo principi, obdelani in predstavljeni v ISO DP 9007.

2.8. Zaključek

Najbolj pomemben prispevek zastavljenih konceptov v ISO DP 9007 je harmonizacija komuniciranja med ljudmi. V zvezi s tem je treba pričakovati, da bodo zastavljeni principi vplivali predvsem na metode, s katerimi analiziramo poslovne sisteme in njihove potrebe. Naloga CS je, da poda splošni dogovor o tem, kako opisati "obravnavani svet" v poslovnem sistemu, ki pa ima to lastnost, da ne omejuje evolucije in razvoj aplikacij tekom življenskega cikla IS in sprememb v svetu IS.

CS in IB opisujeta le konceptualni (logični) vidik informacijskega sistema, kar pomeni, da sta CS in IB definirani z elementi in konstrukti, ki se nanašajo na objekte iz sveta IS in ki odražajo stanje teh objektov. Teoretske osnove za zapis teh stanj najdemo v formalni logiki. Uporabljeni elementi formalne logike pa morajo biti enostavni zaradi zahtev po enostavni uporabi modela. Potrebam po bolj kompleksnih konstruktih, ki omogočajo ugodnejši opis različnih dogodkov v svetu IS,

3. Reference

Delovni dokumenti:

ISO TC/97/SC5/WG5 on DBMS Coordination

ISO TC97/SC21/WG1 on OSI Reference Model

ISO TC97/SC21/WG5 on OSI Application and Presentation Layers

ANSI/X3/SPARC X3H4, IRDS Technical Committee

Zahvala: To delo je bilo delno podprto s strani Iskre Delte in za to se avtorji zahvaljujejo DO Iskra Delta.

B. Završnik
Iskra Elektrooptika, Ljubljana

UDK: 681.519.7

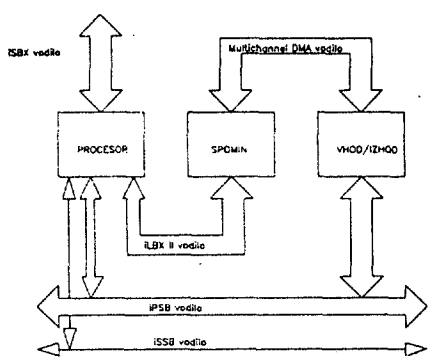
Multibus II je napredno sistemsko vodilo odprtega tipa,namenjeno sistemom s porazdeljeno arhitekturo ter multiprocesiranju. Sistem Multibus II je najbolj uporaben za industrijske aplikacije, kot so, kontrola procesov, robotika, biomedicinska oprema ter zbiranje podatkov. Prednosti so izrazite pri računalniško podprtem načrtovanju, grafiki, razvojnih sistemih in simulaciji s pomočjo računalnikov.

Multibus II is an advanced open-systems bus designed to meet the needs of systems that are moving toward distributed architecture and multiprocessing. Multibus II systems are most appropriate for real-time industrial applications such as process control and robotics,as well as for biomedical equipment and data acquisition. Other applications include computer-aided-design and engineering work stations, graphics equipment, development systems, and computer simulation.

Uvod

Ob prodoru zmogljivih 32 bitnih mikroračunalnikov v multiprocesorske sisteme je nastala potreba po vodilu, ki bi nitro in učinkovito povezoval ta sistem. Novo vodilo Multibus II ima predhodnika v Multibus-u I, od katerega je privzel dve sestavni vodili.MULTIBUS II naj ne bi zamenjal dosedanjega vodila, temveč ga dopolnjeval, saj je na trgu še zelo malo 32 bitnih aplikacij.
Slika prikazuje arhitekturo MULTIBUS II vodila:

ARHITEKTURA MULTIBUS II



Vodilo je odprtega tipa, kar omogoča velikoserijsko proizvodnjo komponent in kar je najvažnejše,prenosljivost programske opreme,preko velikega števila raznovrstnih sistemov.Vodilo ima popolno 32 bitno strukturo, ki omogoča preko multipleksiranja,prenosne hitrosti preko 40 megabaytov na sekundo. Velika prednost intlovega vodila je v tem,da ima močno podporo v MULTIBUS I vodilu,za katerega je dosedaj 200 izdelovalcev proizvajalo preko 1300 izdelkov.

Pričetki razvoja MULTIBUS II vodila segajo tri leta nazaj,ko je intel spoznal,da starejše vodilo ne bo moglo služiti za 32 bitne aplikacije. Pri načrtovanju novega vodila so bile upoštevane želje mnogih proizvajalcev opreme in med njimi je bilo izbrano 17 podjetij, ki so kasneje tudi določile karakteristike MULTIBUS II vodila.

Vodilo iPSB

Paralelno sistemsko vodilo iPSB, predstavlja glavno vodilo za MULTIBUS II. To vlogo mu dajejo naslednje značilnosti : univerzalnost vodila, velika prepustnost, zanesljivost delovanja in izvajanje sistemskih funkcij.
Glavne značilnosti vodila iPSB so :

- na vodilo lahko priključimo do 20 udeležencev, ki so krmiljeni s 8,16 ali 32 bitnimi procesorji in imajo možnost prenašanja 8,16,24 ali 32 bitne podatke.

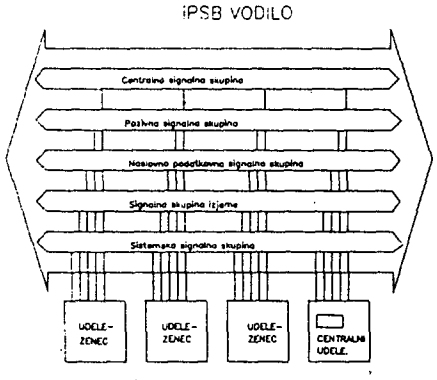
- udeleženci se lahko povežejo med seboj preko štirih naslovnih področij : 32 bitno spominsko, 16 bitno vhodno/izhodno, 8 bitno področje sporočil in 16 bitno povezovalno področje.
- velika prepustnost vodila je omogočena z hitrostjo delovanja 40 mega bajtov na sekundo in posebnimi protokoli delovanja : blokovni prenos in prenos sporočil.
- v vodilo je vgrajen tudi mehanizem,ki omogoča priključevanje po vnaprej določeni prioriteti.
- zanesljivost delovanja je izboljšana s sinhronim delovanjem vodila, parni zaščiti prenosa in DIN konektorji.
- glavne sistemske funkcije opravlja centralni udeleženec, ki izvaja zagon sistema, daje sistemske ure in kontrolira delovanje vodila.
- na vodilu se vse operacije izvajajo v določenih ciklih : pozivni, prenosni in izjemni.

Opis električnih signalov na iPSB vodilu

Glede na funkcije, ki jih opravljajo jih delimo na naslednje skupine :

- centralna skupina
- pozivna skupina
- naslovno/podatkovna skupina
- skupina izjeme
- sistemska skupina

Slika predstavlja priključitev udeležencev na iPSB vodilo.



Pozivna skupina signalov omogoča udeležencem iPSB vodila, preko pozivnega protokola,prevzeti vodstvo nad vodilom. To je potreben pogoj za začetek prenosa po vodilu. Sestavljajo jo naslednji signali : BREQ in ARB5 do AR-B0. S pomočjo prvega izrazi udeleženec željo po dostopu na vodilo, preko ostalih signalov se ob zagonu sistema določi geografski in pozivni naslov vsakega udeleženca na vodilu.
Iste linije določajo pri pozivnem ciklu prioriteto udeleženca.

Naslovno/podatkovno skupino lahko uporabljata samo lastnik vodila in pozvani.Skupino sestavljajo naslovno/pdatkovne linije AD31 do AD0 in štirje signali parnosti PAR3 do PAR0, od katerih vsak skrbi za pravilnost osmih AD linij. Ob pozivu je na AD linijah naslov in ob odgovoru podatek, ki je lahko 8,16,24 ali 32 bitni.

Sistemska skupina je sestavljena iz deset kontrolnih signalov SC9 do SC0, ki kontrolirajo prenosni cikel na iPSB vodilu. Signali so časovno multipleksirani in imajo različen pomen odvisno od faze prenosa. Ob pozivni fazi določajo ukaze in ob fazi odgovora stanje odgovarjajočega udeleženca. Z istimi linijami se izvede tudi izmenjalni protokol med dvema udeležencema na vodilu.

Pomen signalov med pozivno fazo :
SC0 začetek pozivne faze
SC1 zaklenjeno vodilo
SC2 širina podatka
SC3 širina podatka
SC4 naslovno področje
SC5 naslovno področje
SC6 čitanje/zapisovanje
SC7 rezerviran
SC8 parnost za SC7 - SC4
SC9 parnost za SC3 - SC0

Pomen signalov med fazo odgovora :
SC0 začetek pozivne faze
SC1 zaklenjeno vodilo
SC2 konec prenosnega cikla
SC3 pozivni udeleženec pripravljen
SC4 pozvani udeleženec pripravljen
SC5 napaka udeleženca
SC6 napaka udeleženca
SC7 napaka udeleženca
SC8 parnost za SC7 - SC4
SC9 parnost za SC3 - SC0

Signalna skupina izjeme opravlja funkcijo javljanja vsem udeležencem na vodilu, da je pri prenosu podatkov prišlo do napake. Dve signalni liniji javljata napake : TIMOUT javi napako pri izmenjalnem protokolu in BUSERR, ki javlja napake na naslovno/podatkovnih linijah.

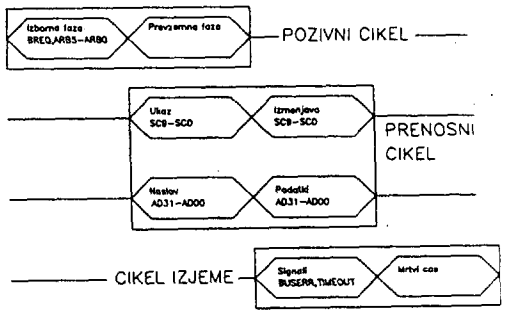
Centralna kontrolna skupina služi za izvajanje sistemskih funkcij celega sistema. Incijalizacija se izvrši s signaloma RST in RSTNC.Za javljanje napak na napajalnih linijah služita signala DCLOW in PROT. Sistemska ura za celotno vodilo je generirana po linijah BCLK in CCLK. S pomočjo signalnih linij LACHn določimo trenutek, ko so na vodilu naslov udeleženca in njegova prioriteta. Vse te signale generira centralni udeleženec.

Napajanje udeležencev je izvedeno preko iPSB vodila, +5V, +12V, -12V, GND in možnim rezervnim napajanjem.

Protokol na iPSB vodilu

Tri je cikli na iPSB vodilu določajo protokol: pozivni cikel, prenosni cikel in cikel izjeme. Izvajajo ga udeleženci na vodilu s krmiljenjem kontrolnih signalov.

Slika prikazuje cikle na iPSB vodilu.



CIKLI NA iPSB VODILU

Pozivni cikel

Udeleženec,ki hoče prenesti informacijo po vodilu mora izvajati pozivni cikel. Med pozivnim ciklom lahko eden ali več udeležencev izrazi željo, da postane lastnik vodila.Rezultat tega

cikla je, da samo eden udeleženec dobi lastništvo nad vodilom in lahko nato izvaja prenosni cikel. V primeru, ko več udeležencev istočasno želi postati lastnik vodila, dobi lastništvo tisti, ki preko ARB5 do ARB0 signalov izkaže največjo prioriteto. Obstajata dve vrsti prioritete, normalna in velika prioriteta. Vsi ostali,med pozivnim ciklom ne izbrani,udeleženci imajo privilegiran položaj in po končanem prenosnem ciklu izbranega udeleženca,dobijo glede na svojo prioriteto lastništvo nad vodilom.Tako je možen vsem udeležencem hiter dostop na vodilo.Pozivni cikel sestavljajo izborna faza, ki izbere lastnika vodila in prevzemna faza,v kateri izbrani udeleženec prevzame kontrolo nad vodilom.

Prenosni cikel

Udeleženec izvede prenosni cikel s pomočjo naslovno/podatkovnih linij AD31 do AD00 ter sistemskimi kontrolnimi signali SC9 do SC0. Prenosni cikel je razdeljen v fazo poziva in fazo odgovora.
Obstajajo tri vrste prenosnega cikla: enojni prenos, blokovni prenos in prenos sporočil. Enojni prenos potrebuje samo en prenosni cikel za njegovo izvršitev. Podoben mu je tudi blokovni prenos le da je faza odgovora podalšana. Pri tem prenosu si dva udeleženca na vodilu izmenjata vlak podatkov. Prenos sporočil uporablja udeleženec, ki hoče istočasno prenesti sporočilo večjemu številu udeležencev.Med prenosnim ciklom mora pozivni udeleženec kontrolirati in detektirati eno od pet možnih napak, ki se lahko pojavijo na vodilu: trajna napaka, napaka v širini podatka, napaka podatka,napaka sporočila, večkratna napaka. Kot posledica detektiranja napake sledi prekinitev prenosnega cikla s strani pozivnega udeleženca. Pri prenosnem ciklu je možno izbirati med tremi naslovnimi področji s katerimi lahko izmenjamo podatke ali sporočila:

- spominsko področje ima 32 bitni naslov in 8, 16,24,32 podatke,blokovni prenos z inkrementom ter možnostjo izmenjave z enim udeležencem.
- vhodno/izhodno področje s 16 bitnim naslovom in 8,16,24,32 bitnim podatki, blokovnim prenosom z enim udeležencem.
- področje sporočil z 8 bitnim naslovom,z možnostjo samo zapisovanja 16,32 bitnih podatkov,blokovnim prenosom z enim ali več udeleženci.
- povezovalno področje je 5 bitno z 8 bitnimi podatki in možnostjo izmenjave podatkov z enim udeležencem.

Vsako od teh področij ima svoje značilnosti in uporabnost v MULTIBUS II arhitekturi.

Cikel izjeme

Cikel izjeme izvede potrebne postopke za odpravo nepravilnosti na iPSB vodilu.Prekine vse aktivnosti na vodilu in omogoči mrtvi čas, ki je potreben,da se obnovi normalno stanje vodila.Cikel izjeme je sestavljen iz faze detektiranja napake ter faze obnovitve. Napako dekodira centralni udeleženec in jo preko signalov izjeme BUSERR in TIMEOUT javi ostalim udeležencem na vodilu.Udeleženci,ki dekodirajo prisotnost signalov napake, prekinejo vse tekoče aktivnosti in ostanejo neaktivni do trenutka, ko preteče mrtvi čas.

Centralne funkcije MULTIBUS II sistema

Izvaja jih centralni udeleženec preko kontrolne skupine signalov. Preko njih se izvede hladni zagon, vroči zagon ali incijalizacija sistema ter regeneracija sistema ob izpadu napetosti.

Električne značilnosti iPSB vodila

Tri vrste eleektričnih vezij so uporabljene za prenos informacij preko iPSB vodila:trinivojska vezja, vezja z odprtim kolektorjem in diferencijalna vezja. Vsa gonilna vezja morajo generirati tokove do 60 mA in izpolnjevati časovne zahteve, ki jih določa največja hitrost prenosa informacij.Na vezni plošči morajo biti vsi signali ustrezno omsko zaključeni.Priključni 96 polni konektorji morajo izpolnjevati zahteve po maksimalnem toku potrebnim za delovanje iPSB vodila.

Mehanske značilnosti iPSB vodila

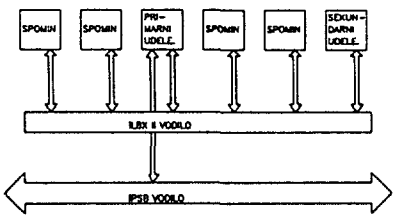
MULTIBUS II določa tudi mehanske dimenzije povezovalnih in osnovnih plošč ter DIN 96 polne priključitvene konektorje. Povezovalna plošča za iPSB vodilo ima 20 konektorjev in vse potrebne omske zaključitve.
Poznamo dve vrste osnovnih plošč:enojna (220mm x 100mm) in dvojna (220mm x 233.4mm). Pri dvojni osnovni plošči je zgornji konektor izbran za iPSB vodilo, spodnji pa za iLBX II vodilo. Pri enojni osnovni plošči imamo samo en konektor, ki je povezan na iPSB vodilo.
Osnovne plošče so standardizirane po IEC normativih.

Vodilo iLBX II

To vodilo omogoča lokalnemu procesorju razširitev spomina.Zaradi velike hitrosti prenosa (48 mega baytov) se dodani spomin vede kot lokalni spomin vgrajen na sami mikroprocesorski plošči. Vodilo lahko povezuje do 6 udeležencev,primarni in sekundarni udeleženec ter štirje udeleženci, ki prvim dvema razširjajo spomin. Upravljanje vodila,si med seboj menjata primarni in sekundarni udeleženec po izmenjalnem protokolu.Vodilo iLBX II je ločeno od iPSB vodila.Možna povezava med obema vodiloma je preko primarnega udeleženca.

Slika prikazuje maksimalno konfiguracijo iLBX II vodila :

MAKSIMALNA KONFIGURACIJA iLBX VODILA



Glavne značilnosti iLBX II vodila so :

- možno je priključiti največ šest udeležencev na vodilo od teh sta dva vodja vodila
- možna je blokada vseh aktivnosti na drugih vodilih, dokler poteka nujna izmenjava podatkov med udeleženci na iLBX II vodilu
- zaradi ločenih linij za naslavljanje,sistemske ukaze in podatke je omogočeno prekrivanje aktivnosti na vodilu
- vse operacije na vodilu se izvajajo sinhrono po taktu, ki ga daje XBCLK ura (maksimalno 12 MHz)
- vodilo uporablja zaradi večje hitrosti delovanja enosmeren izmenjalni protokol
- omogočen je dostop do dveh naslovnih podro-

cij, spominsko in povezovalno.
- delovanje vodila je določeno z naslednjimi cikli : pozivni cikel,prenosni cikel in cikel izjeme.
- velikost podatkovnega vodila je 32 bitov
- velikost naslovnega vodila je 26 bitov

Opis signalnih linij na iLBX II vodilu

Za izvajanje operacij na iLBX II vodilu so potrebne signalne linije,ki so glede na funkcije, ki jih opravljajo, razdeljene na štiri grupe :
- pozivna signalna grupa
- prenosna signalna grupa
- signalna grupa izjeme
- sistemsko kontrolna signalna grupa

Pozivna signalna grupa

Sestavljata jo dva signala XBUSACK in XBUSREQ, katera omogočata izmenjavo lastnistva nad vodilom med primarnim in sekundarnim udeležencem. Trenutno stanje teh dveh linij določa tudi lastnika iLBX II vodila.

Prenosna signalna grupa

Sestavljajo jo 64 signalnih linij, ki omogočajo dve fazi prenosa po iLBX II vodilu:pozivno fazo in fazo odgovora.
Pri pozivni fazi prenosa sodelujejo naslednje signalne linije :
- naslovne linije XA25 do XA00
- komandne linije XC3 do XC0
- signal parnosti XAPAR

Pri fazi odgovora sodelujejo naslednje signalne linije :
- podatkovne linije XD31 do XD00
- signal parnosti XDPAR

Naslovne linije krmili med pozivno fazo prenosa primarni ali sekundarni udeleženec. Vsi udeleženci,ki odgovarjajo na poziv,sprejemajo te linije. Naslovnim in komandnim linijam je dodan zaradi zanesljivosti prenosa signal parnosti. Komandne linije krmili pozivni udeleženec ter preko signalnih linij xC1 in XC0 določa obliko podatkov pri prenosu (1 do 4 byte).Linija XC2 pove ali bo prenos zapis ali čitanje. Linija XC3 določa področje naslavljanja, spomin ali povezovalni spomin.
Podatkovne linije so dvosmerne in prenašajo zapisane in čitane podatke po vodilu iLBX II.Tudi tem linijam je zaradi zanesljivosti prenosa dodan signal parnosti.

Signalna grupa izjeme

Ta signalna grupa omogoča javljanje napak,ki se pojavljajo pri prenosu. Temu namenu služita signala XDERR in XAERR, ki označujeta napako na naslovnem oziroma podatkovnem vodilu.

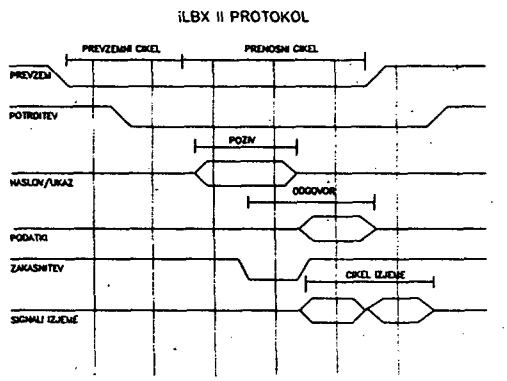
Sistemsko kontrolna signalna grupa

Omogoča kontrolo sistemskih funkcij celega iLBX II vodila, reset, incijalizacijo, takt ter kontrolo sprejema. Te funkcije omogočajo naslednji signali :
XWAIT signal podaljšuje operacije na iLBX II vodilu in tako omogoča priključitev počasnih udeležencev.
XACCREQ signal označuje začetek in konec pozivne faze prenosnega cikla ter veljavost naslovnih in komandnih signalov.
XLOCK signal izključi vse aktivnosti udeležencev iLBX II vodila na ostalih MULTIBUS II vodilih.
XBTCTL označuje blokovno izmenjavo informacij med udeleženci.
XINT signal omogoča udeležencem, da javijo primarnemu udeležencu prekinitev izmenjave informacij na iLBX II vodilu.
XBCLK signal je sinhrona ura za celo iLBX II vodilo in ga generira primarni udeleženec.
RESET signal omogoča incijalizacijo vodila.
XID2 do X1D0 signali priredijo enolične naslove vsem udeležencem na vodilu.
Po iLBX vodilu so speljane tudi napajalne linije +5V in GND.

Protokol na iLBX II VODILU

Protokol na iLBX II vodilu sestavljajo trije cikli :
- prevzemni cikel
- prenosni cikel
- cikel izjeme

Slika prikazuje iLBX II protokol



Prevzemni cikel

Označuje aktivnosti na iLBX II vodilu,pri katerih primarni in sekundarni udeleženec zamenjata vodstvo nad vodilom. Te aktivnosti se izvajajo samo, če sta na vodilu priključena oba udeleženca. V primeru,ko sekundarni udeleženec ne potrebuje vodila z njim upravlja primarni udeleženec. S pomočjo izmenjalnih signalov XBUSREQ (generira ga primarni udelezenec) in XBUSACQ (generira ga sekundarni udelezenec) se izvrši izmenjava vodstva nad vodilom. Po končanem delu mora sekundarni udeleženec vodilo zapustiti.

Prenosni cikel

Omogoča primarnemu ali sekundarnemu udeležencu čitanje ali zapis iz ene od spominskih področij. Cikel sestavljata dve operaciji, poziv in odgovor. Poziv določa vrsto izmenjave informacij, njen naslov, tip spomina in obliko prenosa. Spomin na iLBX II vodilu je razdeljen v dve področii, razširitveni in povezovalni spomin. Organiziran je v obliki (4 x N) bytov. Povezovalni spomin omogoča incijalizacijo vodila ob resetu. Prenosni cikel omogoča tudi zelo hitre paketne prenose informacij. Enemu pozivu lahko sledi več odgovorov. Pozivu pri prenosnem ciklu sledi odgovor, ki je lahko s pomočjo signala XWAIT poljubno zakasnjen. Udeleženci na iLBX II vodilu morajo imeti 32 bitno podatkovno priključitev in s pomočjo prenosnega protokola lahko izločijo 8, 16 ali 32 bitne podatke.

Cikel izjeme

Namenjen je javljanju napak, nastalih pri izmenjavi informacij na iLBX II vodilu. Javljene so lahko tri vrste napak : napaka poziva, napaka odgovora in stalna napaka. Protokol te napake javlja,ne določa pa načina za njihovo odpravljanje.

Električne značilnosti iLBX II vodila

Signali na vodilu so razdeljeni v tri grupe:tri nivojski signali, signali z odprtim kolektorjem in diferencialni signali. Umske zaključitve signalov so izvedene na štiriplastni povezpvalni plošči, ki nosi šest 9o kontaktnih konektorjev (DIN 41612). Glede na potrebno hitrost iLBX II vodila je izbrana tehnologija za krmiljenje signalov FAST in ADVANCED SCHOTTKY.

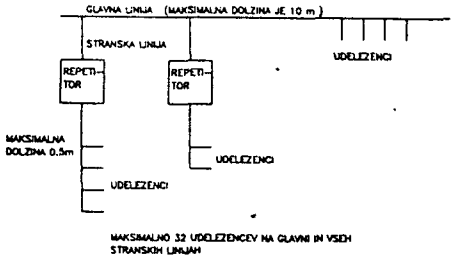
Vodilo iSSB

Vodilo iSSB omogoča serijsko povezavo enot na sistemu MULTIBUS II. Predstavlja ceneno alternativo paralelnemu sistemskemu vodilu iPSB, katerega nadomešča ali dopolnjuje pri kontroli ter diagnostiki.

Glavne značilnosti vodila

Vodilo lahko serijsko poveže do 32 enot,priključenih direktno,ali več,združenih preko repetitorjev. Repetitorji zmanšujejo in izolorajo kapacijativno obremenitev.
Maksimalna dolžina iSSB vodila je lahko 1o metrov.Repetitorji lahko povezejo v celoto največ 20 enot. Slika prikazuje možnosti za priključevanje enot na iSSB vodilo :

MOZNA KONFIGURACIJA iSSB VODILA



Serijsko vodilo iSSB fizično sestavljajo dve liniji z odprtim kolektorjem (SDA, SDB).Te linije vsaka enota lahko cita ali krmili. Dekodirana stanja na dveh linijah prikazuje tabela.

SDA linija	SDB linija	pomen
0	0	zmesnjava
0	1	logična 0
1	0	logična 1
1	1	neaktivno

Sperejemniki lahko sprejet signal vzorčijo in tako izločijo motnje iz sporočila, kar omogoča funkcijonalnost vodila v teških pogojih delovanja (velike motnje iz okolice in ostalih vodil).

Protokol na iSSB vodilu

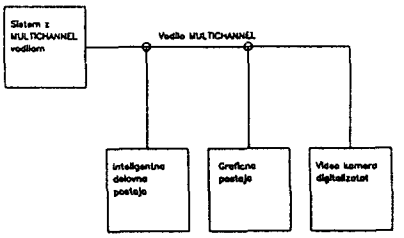
Vsaki enoti na vodilu je omogočen prenos kadarkoli je pripravljena. Pred prenosom se mora prepričati, da je vodilo prosto in počakati na okno v katerem lahko poslje sporocilo.Ker je možno,da več enot istočasno prične s prenosom sporočila je dodan posebni algoritem, ki preprečuje zmesnjavo na vodilu. Ob detekciji takega stanja se časovno porazdeli vsaki enoti okno v katerem lahko prične s prenosom. Iako je čas, potreben za razrešitev zmešnjave na vodilu, zelo kratek,kar omogoča zelo hitre prenose.

Napake na vodilu detektiramo s pomočjo dodatne 16 bitne kode, ki se dodaja prenosu.Umogočen je tudi ponoven prenos sporočila, če le ta,ni veljaven.

Vodilo MULTICHANNAL

Tudi to vodilo je bilo privzeto od MULTIBUS I in omogoča priključevanje periternih enot, ki uporabljajo zelo hitre blokovne prenose podatkov. Priključitev takih enot direktno na iPSB vodilo, bi povzročilo močno zmanšanje njegove prepustnosti.Dolžina vodila je lahko do 15 metrov in omogoča priključevanje do 16 naprav, ki so prilagojene za delovanje na tem vodilu.Vsaki od teh naprav je dodeljen naslov,razen kontrolni napravi, ki vodi celotno vodilo.
Podatki, katere prenašamo po vodilu so lahko 8 ali 16 bitni. Področje nasalavljanja obsega 16 mega-bytov. Blokovni prenosi uporabljajo asinhroni protokol in prenašajo zaporedne lokacije spomina tako, da pošiljatelj, kakor tudi sprejemnik sama skrbita za pravilno naslavljanje. Prenos se prične z vodilnim naslovom in vsak udeleženec v prenosu sam povečuje naslov odvzemanja ali shranjevanja podatka.
Zanesljivost prenosa preverjamo s pomočjo parnosti podatkov.
Na sliki je prikazan sistem, ki ga povezujejo v celoto MULTICHANNAL vodila. Video kamera je izvor velikega števila podatkov, ki jih posredovalna naprava lahko preusmeri direktno na grafični terminal ali poslje podatke na drug MULTIBUS sistem, kjer se dodatno obdelajo in nato izpisejo na grafični terminal.

PRIMER UPORABE MULTICHANNEL VODILA



Udeleženci na MULTICHANNEL vodilu.

Glede na funkcije, ki jih opravljajo na vodilu jih delimo na :
-vodja vodila
-kontrolor vodila
-osnovni udelezeneč

Vodja vodila opravlja funkcijo organizatorja delovanja celega vodila. On določa kakšno delo bodo opravljali kontrolorji vodila in lahko v vsakem trenutku prekine aktivnosti na vodilu ter prevzame kontrolo. To je tudi edini udeleženec vodila, ki nima prirejen naslov. Na enem MULTICHANNEL vodilu je lahko samo eden vodja vodila.

Kontrolor vodila lahko samostojno usmerja podatke po vodilu in obenem opravlja iste funkcije kakor osnovni udeleženec. Ne more pa prevzeti kontrole nad vodilom in odgovoriti na interapt. Na vodilo je lahko priključeno do 15 kontrolorjev katerim so prirejeni naslovi od 0H do 0EH.

Osnovni udeleženec lahko na zahtevo vodje vodila ali kontrolorja vodila posilja ali sprejema podatke. Lahko opravlja tudi obe operaciji istočasno. Na vodilu je lahko istočasno priključenih 15 osnovnih udeležencev, katerim so prirejeni naslovi.
Vse enote priključene na vodilu so lahko v dveh

stanjih, aktivnem in neaktivnem. Vodja vodila lahko enote izloči iz vodila tako, da jih preklopi v neaktivno stanje. Aktivna enota je lahko v enem od naštetih stanj :

- pasivno stanje
- izbrano stanje
- stanje poslušanja
- stanje pošiljanja
- stanje popolne odvisnosti
- stanje popolnega nadzora

V pasivnem stanju enote ves čas sledijo dogajanju na vodilu. Izbrano stanje označuje enoto, ki je spoznala svoj naslov in ostane v tem stanju dokler ne dobi ukaza za izklopitev ali generalni reset. Le dve enoti sta lahko istočasno v izbranem stanju.
Stanje poslušanja označuje enoto, ki sprejema podatke preko naslovno/podatkovneih linij. Vedno je na vodilu vsaj ena enota v tem stanju.
V stanju pošiljanja so enote takrat kadar pošiljajo podatke na naslovno/podatkovne linije.
Enota je v stanju popolne odvisnosti, kadar je vodena od druge enote. Usnovni udeleženec in kontrolor vodila sta lahko v tem stanju.
V stanju popolnega nadzora sta lahko vodja vodila ali kontrolor vodila, kadar vodita prenos podatkov. V tem stanju je istočasno lahko le ena enota.

Opis signalnih linij.

MULTICHANNEL vodilo predstavljajo naslednje linije :

- naslovno/podatkovne linije
- kontrolne linije
- linije za dogovarjanje
- interaptne linije
- komandne linije vodje vodila

Naslovno/podatkovne linije (AD15 - AD0) služijo naslavljanju in prenosu podatkov. Enota, ki pošilja krmili te linije, medtem ko ostali udeleženci poslušajo. Tem linijam je zaradi kontrole dodana tudi linija parnosti prenosa (PB), ki je diferencijalna.

Izbran vodja vodila ali kontrolor vodila krmili kontrolni diferencialni liniji (ADDRESS-NOT-DATA ,READ-NOT-WRITE). Preko njih označuje, kdaj je na vodilu naslov ali podatek ter pove kdaj se izvaja operacija čitanja ali zapisovanja.

Linije za dogovarjanje služijo pošiljatelju in poslušalcu za dogovor o izmenjavi podatkov med seboj. To operacijo opravlja diferencijalna linija (DATA READY) in dve liniji (ADDRESS MODE ACCEPT,DATA MODE ACCEPT). Linija , ki označuje sprejem naslova je logično povezana v AND funkcijo, kar omogoča, da vsi poslušalci na vodilu sprejmejo naslov. Dve prekinitvene linije (SUPERVISOR TAKE OVER, SERVICE REQUEST) sta povezani z logično OR funkcijo, da lahko istočasno več enot generira prekinitev. Edini posredovalec pri prekinitvi je lahko samo vodja vodila. Prekinitev generirajo enote, ki zaznajo parno napako na podatkovnih linijah ali napako same enote.
Komandni liniji vodje vodila, služita za incijalizacijo vodila in za prevzem kontrole nad vodilom. Označeni sta s (SUPERVISOR ACTIVE,RESET). Ob resetu preidejo vse enote na vodilu razen vodje vodila v stanje poslušanja.

Protokol MULTICHANNEL vodila

Protokol je sestavljen iz ciklov vodila, ki se združujejo v sporočila. Minimalno sporočilo sestavljajo :

- dva uvodna cikla, ki izbirata poslušalca
- en cikel za prenos podatka (8,16 bitnega)
- dva zaključna cikla za izklop poslušalca

Med uvodnima in zaključnima cikloma je lahko poljubno dolg vlak podatkov. Prenos poteka asinhrono v treh oblikah :

- naslovna oblika
- podatkovna oblika
- kontrolna oblika

Naslovna oblika prenosa omogoča izbor poslušalca in njegovo izklopitev. Sestavljena je iz dveh ciklov, ki preneseta sporočilo združeno v dveh besedah (8 ali 16 bitnih). Besedi določata naslov spomina ali registrov, naslov enote s katero želi dogovarjanje in smer prenosa.

Podatkovna oblika služi za prenos podatkov med dvema enotama na vodilu in je lahko le sestavni del celega sporočila. Stevilo prenesenih podatkov med pozivom in izklopom pozvanega ni omejeno. Smer pošiljanja podatkov je določena v uvodnem naslovnem sporočilu, poslanem pozvanemu.

Kontrolna oblika služi izmenjavi vodstva nad vodilom med vodjo vodila in kontrolorjem vodila. Kontrolna oblika je potrebna tudi, ko vodja vodila prevzame kontrolo in izklopi vse udeležence.

Polek spominskega naslavljanja, ki obsega 16 mega bajtov, imamo tudi registrsko naslavljanje z obsegom 32 mega bajtov. Registri so lahko 16 ali 8 bitni. Prvih 16 registrov ima poseben namen v MULTICHANNEL vodilu in so namenjeni za sistemsko uporabo. Preko njih lahko vodja vodila ugotavlja vzrok napake v priključenih enotah in programira kontrolorje vodila, da izvedejo samostojne naloge.

Električne lastnosti MULTICHANNEL vodila

Samo vodilo ni namenjeno tudi napajanju posameznih enot, temveč mora vsaka od njih imeti svoje lastno napajanje.
Signalne linije so izvedene s pomočjo prepletenega kabla z maksimalno dolžino 15 metrov. Trije tipi ojačevalnikov signala so uporabljeni za prenos signalov : tri nivojski TTL, TTL z odprtim kolektorjem in diferencijalni.

Mehanske značilnosti MULTICHANNEL vodila

Priključitev MULTIBUS II kompatibilne plošče na MULTICHANNEL vodilo je izvedeno s 60 pinskim konektorjem, priključenem na 60 linijski ploščat prepleten kabel. Moški del konektorja je pritrjen poljubno na vrhu plošče, ki je prilagojena za MULTICHANNEL enoto. Kabel mora imeti na začetku in koncu ustrezno prilagoditev.

Vodilo iSBX

To vodilo je bilo privzeto od MULTIBUS I in omogoča enostavne in cenene razširitve vhodno/izhodnih funkcij osnovnih MULTIBUS II plošč. Osnovni plošči dodamo preko iSBX vodila manso ploščico, ki ima lahko dve normirani velikosti. Tako osnovni plošči, ki je lahko poljubno zasnovana, dodamo paralelni vhod/izhod, serijsk vhod/izhod, grafiko ali izpopolnjene matematične funkcije. Vsak uporabnik lahko tako doda specifične značilnosti univerzalni plošči in v svoje izdelke vnese nove dosežke LSI tehnologije. Ker dodajamo vedno le tisto, kar potrebujemo, se zmanšajo stroški za razširitev sistema.
Vodilo iSBX sestavljajo naslednje linije :

Naslovne linije in linije za izbor vezja
Podatkovne linije
Kontrolne linije
Prekinitvene linije
Dodatne linije
Napajalne linije
Rezervne linije

Naslovne linije in linije za izbor vezja sestavljajo tri naslovne linije (MA0,MA1,MA2) in dve

liniji za izbor vezij (MCS0,MCS1).Te linije omogočajo osnovni plošči, ki je lahko 16 ali 8 bitna, da naslavlja naslove na 16 ali 8 bitni razširitveni plošči.
Podatkovne linije so namenjene obojestranskemu prenosu podatkov med razširitveno ploščo in osnovno ploščo. Odvisno od vrste plošč, ki jih povezuje,je uporabljeno 8 ali 16 linij. Osem bitnemu prenosu služijo linije (MD0-MD7) in šestnajst bitnemu, linije (MD0-MD15).
Kontrolne linije kontrolirajo smer pretoka informacij z ukazoma za čitanje in zapisovanje (IORD,IOWR). Incijalizacijo dodane plošče izvedemo preko linije (RESET). Takt za razširitev je voden preko linije (MCLK) in sistemske linije za pavzo in prisotnost razširitve, preko (MWAIT,MPST).Kontrolne linije omogočajo tudi hitre prenose podatkov s pomočjo linij za direktni spominski dostop (MDRQT,MDACK,TDMA).
Prekinitveni liniji (MINTR0,MINTR1) lahko razširitvena plošča uporabi za zahtevo po prenosu podatkov od osnovne plošče.
Dodatne linije z imenom (OPT0,OPT1) so namenjene dodatnim razširitvam, dodatnim prekinitvenim linijam ali za drugi direktni kanal.
Napajalne linije služijo za napajanje razširitvene plošče z +5 [V] in +12 [V].
Rezervirane linije so namenjene za kasnejše dograjevanje iSBX vodila.

Delovanje iSBX vodila je omejeno na naslednje operacije :

- vhodno/izhodno čitanje
- vhodno/izhodno zapisovanje
- direktni spominski prenos podatkov
- prekinitvene operacije

Vhodno izhodno čitanje in zapisovanje lahko poteka s polno hitrostjo ali s pavzami, ki jih generira razsiritvena plošča ali osnovna plošča.
Direktni spominski prenos, prenasa bloke podatkov brez stalnega posredovanja procesorja.Krmiljenje tega prenosa se vedno prične na osnovni plošči,ki določi smer prenosa podatkov ter začetek in konec.Tudi ta prenos lahko poteka s polno hitrostjo ali z vstavljenimi prekinitvami.
Prekinitvene operacije zahteva razsiritvena plošča, ki aktivira linijo (MINTR0) ali (MINTR1). Prekinitev zaključi osnovna plošča,ko izvrši zahtevano nalogo.

Električne in mehanske značilnosti iSBX vodila.

Vse vhodno/izhodne linije imajo TTL oddajnike in sprejemnike, z izjemo (MD0-MD15),ki imajo tri nivojska stanja. Minimalni tok oddajnikov je 1.6 do 2.0 mA.Maksimalni tok za aktiviranje sprejemnikov je od 0,5 do 2.4 mA. Razširitvena plošča ima lahko dve standardizirani dimenziji.Mehansko sta plošči povezani z enim oziroma tremi plastičnimi vijaki in nosilci. Vodilo iSBX je povezano preko konektorja s 36 kontakti(8 bitna povezava) ali 44 kontakti (8 ali 16 bitna povezava).

Literatura

MULTIBUS Handbuk ,1983 INTEL Corporation

MULTIBUS II BUS Architecture Specification Handbook, 1983 INTEL Corporation

Revija ELECTRONICS / Marec 1984
Multiprocessing 32 bit buses are starting to blossom

ORGANIZACIJE KLJUČEVA U ORGANIZACIJAMA PODATAKA

Siniša J. Djordjević

UDK: 681.3.06

U radu se razmatraju ključevne organizacije podataka kao atribut organizacije podataka. Formalizuje se uticaj ključeva na redosled slogova čime se dalje odredjuje formalizam traženja i pretraživanja. Formalizovano je i sortiranje u funkciji traženja i pretraživanja. Slogovi datoteke posmatraju se i kao retko posednuta matrica.

KEY ORGANIZATIONS IN DATA ORGANIZATIONS: Data key organizations are in this paper considered as atribute of data organization. The influence of keys on order of types is formalized, which further determines search and retrieval formality.

1. PREDPOSTAVKE VIŠEKLJUČNE ORGANIZACIJE PODATAKA

SLOGOVNA ORGANIZACIJA PODATAKA PRIMENJUJE SE I IZVAN AUTOMATSKE OBRADE PODATAKA I PREDSTAVLJA OSNOVNI OBLIK ORGANIZOVANJA PODATAKA. NIJE MALI BROJ 'KNJIGA' KOJE SLUŽE ZA RAZLIČITE EVIDENCIJE A KOD KOJIH JE ORGANIZACIONA JEDINICA PODATAKA JEDAN (ILI VIŠE) KOD KOJI JE U AUTOMATSKOJ OBRADI PODATAKA NAJJEDNOSTAVNIJE I ČESTO I NAJEFIKASNIJE ORGANIZOVATI KAO SLOG. I POREO SAVREMENIJIH OBLIKA ORGANIZACIJE PODATAKA UVEK ĆE POSTOJATI USLOVI POD KOJIMA JE OPTIMALNA ORGANIZACIJA PODATAKA UPRAVO ORGANIZACIJA PODATAKA PREKO SLOGOVA ODNOSNO ORANIZACIJA PODATAKA PREKO DATOTEKA.

U SLOGOVNIM ORGANIZACIJAMA PODATAKA, U DATOTEKAMA, MOŽE POSTOJATI, U OPŠTIM USLOVIMA, SAMO JEDAN FORMALNI FIZIČKI REDOSLED SLOGOVA. FORMALNI REDOSLED JE REDOSLED SLOGOVA KOJI PREDSTAVLJA OBLIK ORGANIZACIJE PODATAKA NA OSNOVU KOJEG SE PROJEKTUJE KORIŠĆENJE BAZE PODATAKA. POD FORMALNIM FIZIČKIM REDOSLEDOM PODRAZUMEVA SE REDOSLED SLOGOVA U ODNOSU NA ADRESE SLOGOVA U MEMORIJI. PODRAZUMEVA SE DA JE JEDAN SKUP SLOGOVA, JEDNA DATOTEKA, U MEMORIJI SMEŠTEN SAMO JEDNOM. DRUGIM REČIMA, AKO SE NA OSNOVU JEDNOG POLJA SLOGOVA FORMIRA FORMALNI FIZIČKI REDOSLED ONDA, U OPŠTIM USLOVIMA, NE POSTOJI FORMALNI FIZIČKI REDOSLED U ODNOSU NA BILO KOJE DRUGO POLJE SLOGOVA. REDOSLED SADRŽAJA DRUGIH POLJA U ODNOSU NA FIZIČKI REDOSLED SLOGOVA JE SLUČAJAN. NA PRIMER, U OPŠTIM USLOVIMA SLOGOVI MOGU BITI SORTIRANI SAMO PREMA SADRŽAJU JEDNOG POLJA.

JEDAN FORMALNI FIZIČKI REDOSLED SLOGOVA JE ATRIBUT VIŠEKLJUČNE ORGANIZACIJE PODATAKA.

FORMIRANJE FORMALNIH REDOSLEDA SLOGOVA JE POSLEDNJA FAZA U PROJEKTOVANJU ORGANIZACIJE PODATAKA. POREO FORMALNOG FIZIČKOG REDOSLEDA POSTOJE FORMALNI REDOSLEDI KOJI SE ZADAJU SPECOM ILI TABELAMA. U VIŠEKLJUČNOJ ORGANIZACIJI PODATAKA PRISTUPA SE FORMIRANJU FORMALNIH REDOSLEDA A NA OSNOVU SADRŽAJA OSTALIH POLJA U SLOGU JER SE SAMO JEDNO POLJE SLOGA MOŽE UPOTREBITI ZA FORMIRANJE FORMALNOG FIZIČKOG REDOSLEDA SLOGOVA. VIŠEKLJUČNA ORGANIZACIJA PODATAKA PREDSTAVLJA FORMIRANJE VIŠE FORMALNIH REDOSLEDA A NA OSNOVU VIŠE RAZLIČITIH POLJA U SLOGOVIMA. KLJUČEVI SU SADRŽAJI POLJA U SLOGOVIMA KOJA SU ISKORIŠĆENA ZA FORMIRANJE FORMALNIH REDOSLEDA.

U VIŠEKLJUČNOJ ORGANIZACIJI PODATAKA KLJUČEVI SE DELE NA JEDAN PRIMARNI I VIŠE SEKUNDARNIH KLJUČEVA A PREMA BROJU FORMIRANIH FORMALNIH REDOSLEDA. POD PRIMARNIM POLJEM NEKOG SLOGA PODRAZUMEVAĆE SE POLJE ZA KOJE VAŽI DA U DATOM SKUPU SLOGOVA NE POSTOJE DVA SLOGA SA ISTIM SADRŽAJEM TOG POLJA. POD SEKUNDARNIM POLJEM PODRAZUMEVAĆE SE POLJE KOJE NIJE PRIMARNO. KLJUČ JE SADRŽAJ POLJA KOJE JE UPOTREBLJENO ZA DEFINISANJE FORMALNOG REDOSLEDA. BEZ ZNAČAJA JE DA LI SE POLJE ČIJI SADRŽAJ PREDSTAVLJA KLJUČ NALAZI ILI NE NALAZI U SAMOM SLOGU. DEFINISANJE PRIMARNOSTI KLJUČA NA OSNOVU PRIMARNOSTI POLJA ČIJI SADRŽAJ JE ISKORIŠĆEN ZA KLJUČ, NIJE EFIKASNO. MOGU POSTOJATI I TAKVE ORGANIZACIJE PODATAKA, PREKO SKUPA SLOGOVA, TAKO DA NI JEDAN KLJUČ NIJE PRIMARNI, DA SE NI JEDAN KLJUČ NE DEFINIŠE NA OSNOVU POLJA KOJE JE PRIMARNO.

PRIMARNI KLJUČ SE DEFINIŠE TAKO DA SE NA OSNOVU PRIMARNOG KLJUČA FORMIRA FORMALNI REDOSLED SLOGOVA. SEKUNDARNI KLJUČ JE KLJUČ KOJI NIJE PRIMARNI ŠTO ZNAČI DA SE NA OSNOVU SEKUNDARNIH KLJUČEVA FORMIRANJU FORMALNI REDOSLEDI SLOGOVA ALI OVI FORMALNI REDOSLEDI NE ZAVISE OD POLOŽAJA SLOGOVA U MEMORIJI. OVAKVI FORMALNI REDOSLEDI MOGU SE FORMIRATI BEZ OBZIRA NA REDOSLED SLOGOVA U MEMORIJI.

VIŠEKLJUČNE ORGANIZACIJE PODATAKA MOGU BITI I BEZ PRIMARNOG KLJUČA ŠTO ZAVISI OD USLOVA PRIMENE.

2. TRAŽENJE I PRETRAŽIVANJE U VIŠEKLJUČNIM ORGANIZACIJAMA PODATAKA

SA STANOVIŠTA ORGANIZACIJE PODATAKA, PREMA[2] MEMORIJA SE MOŽE SHVATITI KAO SKUP ZNAKOVA, KAO SKUP POLJA ILI KAO SKUP SLOGOVA. IZMEĐU ZNAKA POLJA I SLOGA POSTOJE SAMO ORGANIZACIJSKE RAZLIKE JER SVE TRI ORGANIZACIONE JEDINICE PREDSTA-

VLJAJU SADRŽAJ SKUPOVA, SKUP MOŽE IMATI I SAMO JEDAN ELEMENAT MEMORIJSKIH ĆELIJA. SVE MEMORIJSKE ĆELIJE SU ITERATIVNO SUSEDNE. SVAKVE MEMORIJSKE ĆELIJE OBRAZUJU SKUPOVE MEMORIJSKIH ĆELIJA KOJI SE FORMIRAJU PREKO LEVO SUSEDNIH (ILI DESNO SUSEDNIH) MEMORIJSKIH ĆELIJA POČEV OD PROIZVOLJNE MEMORIJSKE ĆELIJE. DUŽINA ITERACIJE JE PROIZVOLJNA. AKO JE ORGANIZACIONA JEDINICA FORMIRANA NA SAMO JEDNOJ MEMORIJSKOJ ĆELIJI ONDA JE TA ĆELIJA SAMA SEBI ITERATIVNO SUSEDNA. BIT KAO ORGANIZACIONA JEDINICA PODATAKA NEMA ZNAČAJA ZA ORGANIZACIJE PODATAKA KOJE SU NAJČEŠĆE U PRAKTIČNOJ PRIMENI.

ZA DEFINISANJE TRAŽENJA I PRETRAŽIVANJA MOGU SE ZANEMARITI RAZLIKE IZMEDJU ORGANIZACIONIH JEDINICA KAKO BI SE DEFINISALI OPŠTI ASPEKTI. HARDVERSKO POLJE SE PODRAZUMEVA KAO SKUP ITERATIVNO SUSEDNIH MEMORIJSKIH ĆELIJA ODNOSNO SKUP ITERATIVNO SUSEDNIH BAJTOVA. OVAJ SKUP MOŽE IMATI I SAMO JEDAN ELEMENAT. SVE ORGANIZACIONE JEDINICE PREDSTAVLJAJU SADRŽAJE HARDVERSKIH POLJA. PREMA DUŽINI HARDVERSKIH POLJA ILI PREMA ODNOSU SADRŽAJA HARDVERSKIH POLJA DEFINIŠU SE ORGANIZACIONE JEDINICE PODATAKA.

SA STANOVIŠTA ORGANIZACIJE PODATAKA MEMORIJA JE SKUP SADRŽAJA HARDVERSKIH POLJA.

DEFINICIJA: PRETRAŽIVANJE JE BINARNA RELACIJA U SKUPU SADRŽAJA HARDVERSKIH POLJA.

DEFINICIJA: TRAŽENJE JE PRESLIKAVANJE U SKUPU SADRŽAJA HARDVERSKIH POLJA.

U OVOM TEKSTU, PRVI I DRUGI ELEMENT UREDJENOG PARA BINARNE RELACIJE SMATRAĆE SE ARGUMENTOM I SLIKOM, RESPEKTIVNO. TIME SE POJEDNOSTAVLJUJE INTERPRETACIJA.

PREMA DEFINICIJAMA, PRETRAŽIVANJE JE POSTUPAK KOJIM SE OD SADRŽAJA JEDNOG HARDVERSKOG POLJA DOLAZI DO SADRŽAJA DRUGOG HARDVERSKOG POLJA. RAZLIKUJEMO SADRŽAJ POLJA OZNAČENO SA X KOJI PREDSTAVLJA ARGUMENT PRETRAŽIVANJA I SADRŽAJE POLJA OZNAČENIH SA Y ,KOJI PREDSTAVLJAJU SLIKE PRETRAŽIVANJA.

BINARNE RELACIJE ZA PRETRAŽIVANJE SU, PREMA POSTUPKU PO KOJEM SE DEFINIŠU, SLOŽENE U ČESTIM ORGANIZACIJAMA PODATAKA ŠTO PODRAZUMEVA PRIMENU I ODREDJENIH SUKCESIVNIH PRESLIKAVANJA U DOBIJANJU KONAČNIH SLIKA Y .

U ORGANIZACIJI PODATAKA BINARNA RELACIJA U U KOJOJ JE ARGUMENT JEDNAK SVIM SLIKAMA NEMA SMISLA JER NEMA SMISLA PRETRAŽIVATI DA BI SE DOBIO PODATAK KOJI SE POSEDUJE I PRE PRETRAŽIVANJA.

TRAŽENJE JE SPECIJALAN SLUČAJ PRETRAŽIVANJA.

UVODJENJEM SADRŽAJA HARDVERSKIH POLJA ZA OSNOVNE ORGANIZACIONE JEDINICE IZBEGNUTO JE VEZIVANJE ZA OSTALE ORGANIZACIONE JEDINICE U DEFINISANJU PRETRAŽIVANJA. OVO JE OPRAVDANO I ZATO ŠTO I ARGUMENT I SLIKE PRETRAŽIVANJA MOGU DA BUDU SVE ORGANIZACIONE JEDINICE.

U DATOTEKAMA POSTOJE ČESTI ZAHTEVI ZA PRETRAŽIVANJEM PO SADRŽAJU POLJA KOJE NIJE UPOTREBLJENO ZA DEFINISANJE FORMALNOG FIZIČKOG REDOSLEDA SLOGOVA. OVAKVO PRETRAŽIVANJE PODRAZUMEVA, AKO NEMA ODGOVARAJUĆE ORGANIZACIJE, DA SE ISPITA SVAKI SLOG I IZDVOJE NADJENI SLOGOVI NA OSNOVU ARGUMENATA PRETRAŽIVANJA. KAKO SE SLOGOVI SMEŠTAJU NA SPORIJIM SPOLJNIM MEMORIJAMA TO OVAKVO POTRAŽIVANJE ZAHTEVA VREME KOJE JE ČESTO SA STANOVIŠTA ELEKTRONSKE OBRADE PODATAKA NEPRIHVATLJIVO. U TOM SMISLU FORMIRAJU SE NOVI FORMALNI REDOSLEDI, PARALELNO SA POSTOJEĆIM FORMALNIM FIZIČKIM REDOSLEDOM, ČIME SE OBEZBEDJUJE PRETRAŽIVANJE I ŠTO JE VAŽNIJE, U ZAVISNOSTI OD ORGANIZACIJE, MOGUĆE JE PRETRAŽIVANJE VRŠITI U CENTRALNOJ JEDINICI. FORMALNI REDOSLEDI FORMIRAJU SE PREMA SADRŽAJIMA POLJA KOJIMA U DATOJ ORGANIZACIJI PODATAKA POSTOJE ZAHTEVI ZA PRETRAŽIVANJEM OVIM POLJA PO ČIJIM SADRŽAJIMA SE VRŠI PRETRAŽIVANJE POSTAJU SEKUNDARNI KLJUČEVI.

ORGANIZACIJE PODATAKA PREKO SEKUNDARNIH KLJUČEVA POSTAJU SLOŽENIJE, ZAUZIMAJU VEĆI MEMORIJSKI PROSTOR ALI SU PRETRAŽIVANJA ZNATNO BRŽA.

3. SPREGNUTI SEKUNDARNI KLJUČEVI

OVAJ DEO PREDSTAVLJA, SA MANJIM IZMENAMA, INTERPRETACIJU VIŠEKLJUČNE ORGANIZACIJE PODATAKA OPISANE U (1). VIŠEKLJUČNA ORGANIZACIJA PODATAKA POSLEDICA JE ISKAZNOG PRETRAŽIVANJA U SLOGOVNIM ORGANIZACIJAMA PODATAKA. NA OSNOVU TAKVOG PRETRAŽIVANJA DOBIJAJU SE SLOGOVI, PODACI, U KOJIMA SU SADRŽAJI ODREDJENIH POLJA U ODNOSIMA DEFINISANIM ISKAZIMA. TO SU NAJČEŠĆE I I ILI ILI SLOŽENI I ILI ISKAZI MADA JE PRETRAŽIVANJEM MOGUĆE DOBITI I SLOGOVE U KOJIMA SU SADRŽAJI ODREDJENIH POLJA U ODNOSIMA I DRUGIH PREDSTAVA (NI,NILI,ISKLJUČIVO ILI) I SLOŽENIH ISKAZA. DA BI SE OVO PRETRAŽIVANJE POJEDNOSTAVILO, DA SE NE BI PRISTUPALO SVAKOM SLOGU, FORMIRAJU SE FORMALNI REDOSLEDI A PREMA SADRŽAJIMA POLJA PO KOJIMA ĆE SE VRŠITI PRETRAŽIVANJE. SVAKO OVAKVO POLJE POSTAJE SEKUNDARNI KLJUČ. FORMIRANJEM FORMALNIH REDOSLEDA SMANJUJE SE NEPRIHVATLJIVO VREME PRETRAŽIVANJA SVAKOG SLOGA KADA SE RADI O DATOTEKAMA SA VELIKIM BROJEM SLOGOVA.

NAJJEDNOSTAVNIJI OBLIK FORMIRANJA FORMALNOG REDOSLEDA MEDJU SLOGOVIMA JE FORMIRANJE SPREGE KOJOM SE SLOGOVI POVEZUJU. ZANEMARUJE SE VRSTA UPOTREBLJENE SPREGE. SPREGA SE FORMIRA PO SVAKOM SEKUNDARNOM KLJUČU ODNOSNO PO SADRŽAJU SVAKOG POLJA NA OSNOVU KOJEG ĆE SE VRŠITI PRETRAŽIVANJE. SPREGOM SE DOBIJA NOVI FORMALNI REDOSLED SLOGOVA JER SVAKI SLOG SADRŽI, PO SVAKOM SEKUNDARNOM KLJUČU, JOŠ JEDNO POLJE NA OSNOVU ČIJEG SADRŽAJA SE DOBIJA ADRESA NEKOG SLOGA U MEMORIJI. RAZLIČITI SEKUNDARNI KLJUČEVI UPUĆUJU NA RAZLIČITE SLOGOVE U MEMORIJI. SADA JE NA OSNOVU SPREGE MOGUĆE FORMIRATI FORMALNE REDOSLEDE PO SVAKOM SEKUNDARNOM KLJUČU ZA KOJI POSTOJI POLJE SPREGE. MOGUĆE JE, NA PRIMER, SORTIRATI SLOGOVE PO SVAKOM SEKUNDARNOM KLJUČU ČIME SE PRETRAŽIVANJE ZNATNO POJEDNOSTAVLJUJE.

JEDNA OD ČESTIH OSOBINA SEKUNDARNIH KLJUČEVA JE DA POLJE ZA SEKUNDARNI KLJUČ SADRŽI MANJI BROJ RAZLIČITIH SADRŽAJA, DA SU ISTI SADRŽAJI MNOGIH SLOGOVA PO TOM POLJU. SLOGOVI SA ISTIM SADRŽAJEM POLJA ZA ODREDJENI SEKUNDARNI KLJUČ MOGU SE PROIZVOLJNO POVEZIVATI ALI SE PREMA HARDVERSKIM KARAKTERISTIKAMA OVO POVEZIVANJE VRŠI OD MANJIH KA VEĆIM ADRESAMA. OVO ZATO DA BI SE JEDNIM PROLAZOM MOGLI UTVRDITI SVI SLOGOVI KOJI SADRŽE DATI SADRŽAJ TOG POLJA, KOJI IMAJU ODREDJENI SEKUNDARNI KLJUČ.

PRI PRETRAŽIVANJU JAVLJA SE PROBLEM KOJI SE MOŽE OZNAČITI KAO "TRKA SPREGE". DA BI SE OSTVARIO ŠTO MANJI BROJ PROLAZA UVEK JE POTREBNO SLEDITI SPREGU KA MANJOJ ADRESI ŠTO ZNAČI DA STALNO TREBA UPOREDJIVATI POKAZIVAČE. POSTOJI GUBITAK U VREMENU KOJI JE NEZNATAN I DALEKO MANJI OD GUBITKA KOJI BI SE OSTVARIO VRAĆANJEM MEHANIZMA ZA ČITANJE JER SU PODACI SMEŠTENI NA SPOLJNIM MEMORIJAMA.

KOD FORMIRANIH FORMALNIH REDOSLEDA MOGUĆE JE POSTAVLJATI INDEKSE KOJIMA SE POSTUPAK PRETRAŽIVANJA JOŠ VIŠE SKRAĆUJE. TABELOM INDEKSA IZ PRETRAŽIVANJA SE ELIMINIŠU PODSKUPOVI SLOGOVA KOJE NE TREBA PRETRAŽIVATI TE SE PODSKUP SLOGOVA KOJI MOŽE BITI REZULTAT PRETRAŽIVANJA SMANJUJE. PRINCIP JE ISTI KAO KOD INDEKS SEKVENCIJALNE DATOTEKE.

ZA SEKUNDARNI KLJUČ SA MANJIM BROJEM VREDNOSTI POBOLJŠANJE SE POSTIŽE ORGANIZOVANJEM SPREGE PO SVAKOJ VREDNOSTI KLJUČA POSEBNO. FORMIRA SE TABELA VREDNOSTI SEKUNDARNOG KLJUČA SA ANKERIMA KOJIMA SE UKAZUJE NA PRVI SLOG (SLOG SE NAJMANJOM ADRESOM) SA DATOM VREDNOŠĆU SEKUNDARNOG KLJUČA. POSTIŽE SE JEDNOSTAVNIJE PRETRAŽIVANJE PO JEDNOJ VREDNOSTI SEKUNDARNOG KLJUČA. I OVO JE INDEKSNA ORGANIZACIJA ALI SE PODSKUP SLOGOVA KOJI SU OBUHVAĆENI JEDNIM INDEKSOM ODREDJUJE NA DRUGI NAČIN, PRMA SLOGOVIMA KOJI IMAJU ISTI SADRŽAJ. IZBOR INDEKSNE ORGANIZACIJE, OD NAVEDENE OVE ORGANIZACIJE, ZAVISI OD BROJA VREDNOSTI DATOG SEKUNDARNOG KLJUČA.

AKO JE BROJ VREDNOSTI SEKUNDARNOG KLJUČA MALI ONDA SE U OKVIRU JEDNE VREDNOSTI SEKUNDARNOG

KLJUČA PRAVI VIŠE TABELA ODNOSNO, ORGANIZUJE SE VIŠE ANKERA KOJI SE PRILAGOĐAVAJU ODREĐENIM CELINAMA SKUPA SLOGOVA. OVE CELINE MOGU BITI LOGIČKOG ILI HARDVERSKOG KARAKTERA. TABELE MOGU DA SADRŽE I INFORMACIJE O BROJU UPISA PO CELINAMA. OVAKVOM ORGANIZACIJOM IMA SE BOLJI UVID O RASPODELI VREDNOSTI (SADRŽAJA) SEKUNDARNIH KLJUČEVA PO CELINAMA.

S OBZIROM NA PARALELNE OPERACIJE, PARALELNI PRISTUP DISK JEDINICAMA, VIŠEKLJUČNA ORGANIZACIJA PODATAKA MOŽE SE DALJE UNAPREDITI AKO SE HARDVERSKE CELINE SMEŠTAJU NA RAZLIČITE DISK JEDINICE. INDEKSIRANJE SE VRŠI NA NIVOU HARDVERSKIH CELINA PO DISK JEDINICAMA A NAJEFIKASNIJE JE DA TO BUDU CILINDRI. NA TAJ NAČIN, PRI DOBROJ ORGANIZACIJI, PRETRAŽIVANJE SE MOŽE VRŠITI BEZ POMERANJA MEHANIZMA ZA ČITANJE I PISANJE. OVAKVA, PARALELNA ĆELIJSKA ORGANIZACIJA, UNAPREĐUJE PRETRAŽIVANJE NA OSNOVU HARDVERSKIH KARAKTERISTIKA SISTEMA.

AKO LISTA INDEKSA SADRŽI UPIS PO SVAKOM SLOGU DATOTEKE ONDA JE TO INVERTOVANA LISTA. U INVERTOVANOJ LISTI SE NALAZE, U SVAKOM SLOGU INVERTOVANE LISTE, SVI SEKUNDARNI KLJUČEVI PO KOJIMA SE VRŠI PRETRAŽIVANJE KAO I ADRESE ILI PRIMARNI KLJUČ SLOGA IZ KOJEG SU PREUZETI TI SEKUNDARNI KLJUČEVI. INVERTOVANIM LISTAMA SMANJUJE SE MEMORIJSKI PROSTOR KOJI SE PRETRAŽUJE A KAO REZULTAT PRETRAŽIVANJA DOBIJA SE SPISAK ADRESA ILI PRIMARNIH KLJUČEVA SLOGOVA IZ KOJIH ĆE SE PREUZETI PODACI. SA SMANJENJEM MEMORIJSKIM PROSTOROM SMANJUJE SE KRETANJE MEHANIZMA ZA ČITANJE I PISANJE ČIME SE UBRZAVA PRETRAŽIVANJE. DRUGA MOGUĆNOST JE, KADA INVERTOVANE LISTE NISU VELIKE, DA SE INVERTOVANE LISTE U CELINI, ILI U DELOVIMA, PRETRAŽUJU U CENTRALNOJ JEDINICI ČIME SE ZNATNO UBRZAVA PRETRAŽIVANJE.

I INVERTOVANE LISTE MOGU IMATI SAMO JEDAN FORMALNI FIZIČKI REDOSLED. FORMIRANJEM FORMALNIH REDOSLEDA MOŽE SE, NORMALNO, POVEĆATI EFIKASNOST PRETRAŽIVANJA INVERTOVANE LISTE. SADA SE INVERTOVENE LISTE MOGU SHVATITI KAO VIŠEKLJUČNA ORGANIZACIJA PODATAKA TE SE MOŽE PRIMENITI PRETHODNO RečENO O FORMIRANJU FORMALNIH REDOSLEDA. MOGU SE IZDVOJITI SEKUNDARNI KLJUČEVI KOJI IMAJU ZNAČAJNIJE MESTO U PRETRAŽIVANJU I SAMO NA OSNOVU NJIH FORMIRATI FORMALNI REDOSLED INVERTOVANE LISTE. POSTUPAK SE MOŽE NASTAVLJATI U ZAVISNOSTI OD BROJA SEKUNDARNIH KLJUČEVA.

POVEĆANJEM BRZINE PRETRAŽIVANJA ORGANIZACIJA PODATAKA POSTAJE SLOŽENIJA I ZAUZEĆE MEMORIJSKOG PROSTORA POSTAJE VEĆE. REŠENJE OVOG PROBLEMA JE U OPTIMIZACIJI A PREMA SOPSTVENIM KRITERIJUMIMA.

4. SORTIRANJE KAO METODA PRETRAŽIVANJA

USVOJIĆE SE UOBIČAJENE PREDPOSTAVKE SORTIRANJA. SLOGOVI SE SORTIRAJU PREMA SADRŽAJIMA POLJA JEDNAKIH DUŽINA. MEĐU ZNACIMA POSTOJI REDOSLED ŠTO ZNAČI DA SE SORTIRAJU ALFANUMERIČKA POLJA. LEVO ZNAKOVNO MESTO JE UVEK VEĆE TEŽINE. TEŽINA SU DEFINISANE TAKO DA SE UPOREĐIVANJE POLJA VRŠI NA SLEDEĆI NAČIN: TRAŽI SE, S LEVA U DESNO, PRVO ZNAKOVNO MESTO NA KOJEM SE RAZLIKUJU POLJA KOJA SE UPOREĐUJU. ODNOS IZMEĐU POLJA JEDNAK JE ODNOSU ZNAKOVA NA TOM ZNAKOVNOM MESTU.

POD PRIDRUŽIVANJEM POLJA B POLJU A, U OZNACI A||B, PODRAZUMEVAĆE SE POLJE KOJE PREDSTAVLJA POLJE A PRODUŽENO S DESNE STRANE POLJEM B. PRIDRUŽIVANJE POLJA NE PREDPOSTAVLJA I FIZIČKI SMEŠTAJ POLJA B PORED POLJA A.

NEKA JE DATO POLJE A KOJE PREDSTAVLJA N POLJA PRIDRUŽENIH POLJU A(0).A(1) JE OZNAKA POLJA U POLJU A PRI ČEMU JE I=0,1,2,...,N. PRIDRUŽIVANJE SE ZADAJE ITERACIJOM:

$$A'(0)=A(0),\ A'=A'(I-1)||A(I)\ ,\quad A'(I-1)=A$$

GDE SU A'(I) I A POMOĆNA POLJA.

NEKA JE DAT SKUP SLOGOVA TAKVIH DA SVAKI SLOG SADRŽI POLJE A. SORTIRANJEM DATOG SKUPA SLOGOVA PO SADRŽAJU POLJA A DOBIJA SE FORMALNI FIZIČKI REDOSLED (SORTIRANI SLOGOVI) U ODNOSU NA SADRŽAJ POLJA A(0). PRIDRUŽIVANJE POLJA NE PODRAZUMEVA I ODGOVARAJUĆI FIZIČKI SMEŠTAJ POLJA JER JE SORTIRANJE MOGUĆE BEZ OBZIRA NA POLOŽAJ POLJA U SLOGU. IZMEĐU SADRŽAJA POLJA A(I) I POLJA A(I+1) POSTOJI ODNOS GRUPA - PODGRUPA U POGLEDU FIZIČKOG REDOSLEDA SLOGOVA JER ĆE SE U OKVIRU JEDNE VREDNOSTI POLJA A(I) NAĆI JEDAN PORED DRUGOG SVI SLOGOVI KOJI IMAJU ISTU VREDNOST POLJA A(I) I SVE OSTALE VREDNOSTI, POREĐENE, POLJA A(I+1). OVO JE POSLEDICA PREDPOSTAVKI SORTIRANJA ODNOSNO NAČIN NA KOJI SU DEFINISANE TEŽINE SLOGOVA.

NEKA JE DATO POLJE J<N, A_J POLJE JE MEĐUPROIZVOD ITERACIJE (I).

STAV: SVI SLOGOVI SA ISTIM SADRŽAJEM POLJA A_J NALAZE SE FIZIČKI SMEŠTENI JEDAN POREĐ DRUGOG U SKUPU SLOGOVA SORTIRANOM PREMA SADRŽAJU POLJA A_J. STAV JE DIREKTNA POSLEDICA PREDPOSTAVKI SORTIRANJA.

NA OSNOVU PRETHODNOG STAVA, PRETRAŽIVANJA SKUPA SLOGOVA PO POLJU A_J, J=0,1,2,...,N, JE JEDNOSTAVNO JER POSTOJI FORMALNI FIZIČKI REDOSLED U ODNOSU NA SVAKO POLJE A_J.

INVERTOVANA LISTA PREDSTAVLJA SKUP SLOGOVA U KOJEM BI SE SEKUNDARNI KLJUČEVI MOGLI SMATRATI POLJIMA A(I). SORTIRANJEM INVERTOVANE LISTE PREMA SADRŽAJU POLJA A OMOGUĆUJE SE PRETRAŽIVANJE PO SADRŽAJIMA POLJA A_J NA OSNOVU FORMALNOG FIZIČKOG REDOSLEDA SLOGOVA U ODNOSU NA SADRŽAJE POLJA A_J (J OZNAKA ZA INDEKS UMESTO UPISIVANJA INDEKSA U SLEDEĆEM REDU).

A_J SADA PREDSTAVLJA ODREĐENI SEKUNDARNI KLJUČ ILI ODREĐENU KOMBINACIJU SEKUNDARNIH KLJUČEVA PO KOJIMA SE PRETRAŽUJE. JASNO JE DA JE MOGUĆE PROIZVOLJNO BIRATI REDOSLED SEKUNDARNIH KLJUČEVA U POLJU A. OVAJ REDOSLED IMA SMISLA AKO JE DEFINISAN ZAHTEVOM ZA PRETRAŽIVANJE.

POSTUPAK POČINJE ODREĐIVANJEM REDOSLEDA SEKUNDARNIH KLJUČEVA. REDOSLED SEKUNDARNIH KLJUČEVA TREBA DA OMOGUĆI DA SE ZAHTEVI PRETRAŽIVANJA MOGU OSTVARITI NA OSNOVU POLJA A_J.POLJE A SAČINJAVAJU SAMO ONI SEKUNDARNI KLJUČEVI PO KOJIMA SE VRŠI PRETRAŽIVANJE. INVERTOVANA LISTA SE DALJE SORTIRA NA OSNOVU SADRŽAJA POLJA A.

RAZMATRAĆE SE POSTUPAK ODREĐIVANJA REDOSLEDA SEKUNDARNIH KLJUČEVA U POLJU A NA OSNOVU SLOŽENIH I-ILI ISKAZA PRETRAŽIVANJA. NA ISTI NAČIN MOGU SE RAZVITI I METODE ODREĐIVANJA REDOSLEDA SEKUNDARNIH KLJUČEVA I ZA DRUGE PROSTE I SLOŽENE ISKAZE PRETRAŽIVANJA. NEKA JE $V_{(SK)}^{K\ J}$ K-TA VREDNOST I - TOG SEKUNDARNOG KLJUČA. PREKIDAČKA PROMENLJIVA $V_{(SK)}^{K\ I}$ I TOG SLOGA IMA VREDNOST 1 AKO U SLOGU I-TI SEKUNDARNI KLJUČ IMA K-TU VREDNOST. U SKLADU SA PREKIDAČKOM ALGEBROM DEFINIŠE SE I FUNKCIJA ISKAZA ZA SVAKI SLOG I PREDSTAVLJA NEKU KOMBINACIJU I /ILI ILI FUNKCIJA. IZRAZI OBLIKA: $V_{(SK)}^{I1\ I}$ 'ILI' $V_{(SK)}^{I2\ K}$ OZNAČAVAJU, NPR. , ODREĐIVANJE SVIH SLOGOVA KOJI IMAJU I1 VREDNOST I-TOG SEKUNDARNOG KLJUČA ILI I2-TU VREDNOST K TOG SEKUNDARNOG KLJUČA. U SKLADU SA PREKIDAČKOM ALGEBROM FUNKCIJA ISKAZA:

$$F = V_{(SK)}^{I1\ I}\ \text{'ILI'}\ V_{(SK)}^{I2\ K}$$

IMA VREDNOST 1 AKO BAR JEDAN OD PROMENLJIVIH IMA VREDNOST 1.

AKO FUNKCIJA F IMA VREDNOST 1 ZA ODREĐENI SLOG ONDA JE TAJ SLOG REZULTAT PRETRAŽIVANJA.

KOMPLEMENT PREKIDAČKE PROMENLJIVE V-K SU SVE OSTALE VREDNOSTI I-TOG SEKUNDARNOG KLJUČA ŠTO ZNAČI DA JE:

$$\bar{V}(SK_K)=V(SK_I)\ 'ILI'\ V(SK_1)\ 'ILI'\ \ldots$$

$$'ILI'\ V(SK_2)\ 'ILI'\ V(SK_{K+1})\ 'ILI'\ \ldots$$

$$'ILI'\ V(SK_M)\qquad \ldots\ (2)$$

M JE BROJ VREDNOSTI I-TOG SEKUNDARNOG KLJUČA.

KAKO JE I,ILI I NE POTPUNI SKUP PREKIDAČKIH FUNKCIJA TO SE SVI OSTALI ISKAZI MOGU REALIZOVATI PREKO I,ILI I NE SKUPA ISKAZA.

SVAKO PRETRAŽIVANJE JE ISKAZNO PA JE PREKIDAČKA INTERPRETACIJA NAPOČITO EFIKASNA KADA SE PRETRAŽIVANJE VRŠI PREGLEDAVANJEM SVAKOG SLOGA. DAJE SE TABELA VREDNOSTI SEKUNDARNIH KLJUČEVA PO KOJIMA SE PRETRAŽUJE I ODREDJUJE SE VREDNOST ISKAZNE FUNKCIJE ZA SVAKI SLOG. AKO JE VREDNOST ISKAZNE PREKIDAČKE FUNKCIJE 1 ONDA JE SLOG REZULTAT PRETRAŽIVANJA.

PROBLEM ISKAZNOG PRETRAŽIVANJA PO VEĆEM BROJU SEKUNDARNIH KLJUČEVA U ORGANIZACIJAMA PODATAKA SE SPREGNUTIM SEKUNDARNIM KLJUČEVIMA JE SLOŽENIJI. REŠENJE OVOG PROBLEMA BI TREBALO DA BUDE, ZBOG OBIMNOSTI, PREDMET POSEBNE ANALIZE. U OVOM TEKSTU RAZMATRAĆE SE SAMO PRETRAŽIVANJE SORTIRANJEM PREMA SLOŽENIM I,ILI I NE ISKAZIMA.

SVAKA VREDNOST JEDNOG SEKUNDARNOG KLJUČA JE JEDNA PREKIDAČKA PROMENLJIVA. U OKVIRU ODREDJIVANJA REDOSLEDA SEKUNDARNIH KLJUČEVA PO KOJIMA ĆE SE VRŠITI SORTIRANJE RADI JEDNOSTAVNIJEG PRETRAŽIVANJA RAZLIKUJEMO DVA POSTUPKA. PRVI JE OSLOBADJANJE OD SVIH KOMPLEMENATA PREMA RELACIJI (2). DRUGI POSTUPAK JE IZRAŽAVANJE FUNKCIJE PREKO PROIZVODA PROMENLJIVIH, PREKO IMPLIKANATA FUNKCIJE. OVO PREDSTAVLJA OSLOBADJANE OD ZAGRADA NA OSNOVU DISTRIBUTIVNOSTI I OPERACIJE U ODNOSU NA ILI OPERACIJU. IMPLIKANTI NE SADRŽE KOMPLEMENTE PROMENLJIVIH JER SU DEFINISANI, PREMA RELACIJI (2), TAKO DA SE ZAMENJUJU ILI ZBIROM NEKOMPLEMENTIRANIH PROMENLJIVIH. AKO SE U JEDNOM PROIZVODU PROMENLJIVIH JAVI JEDAN ISTI SEKUNDARNI KLJUČ SA DVE ILI VIŠE VREDNOSTI ONDA SE TAJ PROIZVOD MOŽE ISKLJUČITI IZ SKUPA PROIZVODA PROMENLJIVIH JER NE POSTOJI SLOG KOJI MOŽE IMATI DVE VREDNOSTI ISTOG SEKUNDARNOG KLJUČA. NI PROSTI IMPLIKANTI FUNKIJE ISKAZA NE SADRŽE KOMPLEMENTE PROMENLJIVIH JER JE VREDNOST FUNKCIJE 0 UVEK KADA JE MAKAR JEDNA OD PROMENLJIVIH JEDNAKA NULI. IZ MINIMALNOG POKRIVANJA ODREDJUJE SE BROJ REDOSLEDA SEKUNDARNIH KLJUČEVA NA OSNOVU KOJIH ĆE SE IZVRŠITI SORTIRANJE.

MINIMIZIRANJE BROJA REDOSLEDA JE JOŠ JEDAN, VRLO SLOŽEN, PRUBLEM KOJI SE ZBOG TOGA NEĆE RAZMATRATI U OVOM TEKSTU.

ZA RANJE SLOŽENE ISKAZE PRETRAŽIVANJA JEDNOSTAVNO SE ODREDJUJE BROJ REDOSLEDA PO KOJIMA ĆE SE IZVRŠITI SORTIRANJE. U PRAKSI SU ČEŠĆI JEDNOSTAVNIJI ISKAZI PRETRAŽIVANJA.

U SKUPU PROSTIH IMPLIKANATA BIRA SE NAJVEĆI PODSKUP, OSNOVNO ODREDJIVANJE PODSKUPOVA, TAKAV DA SE U SVAKOM PROSTOM IMPLIKANTU NALAZI ISTI SEKUNDARNI KLJUČ BEZ OBZIRA NA VREDNOSTI. OVAJ SEKUNDARNI KLJUČ POSTAJE POLJE A(0). U PODSKUPU SE DALJE BIRA NOVI PODSKUP NIŽEG REDA SA ISTOM OSOBINOM ALI ZA NEKI DRUGI SEKUNDARNI KLJUČ KOJI POSTAJE POLJE A(1). POSTUPAK SE DALJE NASTAVLJA DOK SE NE UKLJUČE SVI SEKUNDARNI KLJUČEVI U POLJE A. AKO SE OVA ILI VIŠE KLJUČA NALAZE U ISTOM BROJU PROSTIH IMPLIKANATA ONDA SE BIRA KLJUČ KOJI SA NANJIM BROJEM VREDNOSTI UČESTVUJE U IMPLIKANTIMA. AKO JE I BROJ VREDNOSTI JEDNAK ONDA SE PROIZVOLJNO BIRA JEDAN KLJUČ. SORTIRANJEM PO SADRŽAJU POLJA A FIZIČKI REDOSLED SLOGOVA ZAVISI OD ZAHTEVA PRETRAŽIVANJA. POVOLJAN FIZIČKI REDOSLED POSLEDICA JE PRETPOSTAVKI SORTIRANJA. UOČAVANJE PODSKUPOVA U SKUPU MINIMALNIH IMPLIKANATA ZAVRŠAVA SE KADA SE SVI MINIMALNI IMPLIKANTI IZ MINIMALNOG POKRIVANJA UVRSTE U NEKI PODSKUP. JEDAN MINIMALNI IMPLIKANT SE UVRŠĆUJE U SAMO JEDAN PODSKUP JER NIJE POTREBNO DUPLITATI PRETRAŽIVANJE PO TOJ KOMBINACIJI SEKUNDARNIH KLJUČEV. BROJ OSNOVNIH ODREDJIVANJA PODSKUPOVA ODREDJUJE BROJ SORTIRANJA KOJE BI TREBALO IZVESTI. OPISANIM POSTUPKOM DOBIJA SE ZADOVOLJAVAJUĆI BROJ SORTIRANJA. ZA VRLO SLOŽENE ISKAZE BROJ REDOSLEDA SEKUNDARNIH KLJUČEVA PO KOJIMA TREBA SORTIRATI POSTAJE NEPRIHVATLJIV.

NA PRIMER, NEKA SU DATA DVA ISKAZA PRETRAŽIVANJA FUNKCIJAMA:

$$F_1=(V(SK_1)\ 'ILI'\ V(SK_2))\ 'I'\ (V(SK_1)\ 'ILI'\ V(SK_2))$$

$$F_2=(V(SK_1)\ 'ILI'\ V(SK_2))\ 'ILI'\ (V(SK_1)\ 'ILI'\ V(SK_2))$$

FUNKCIJOM F-1 ZADAJE SE ISKAZ: ODREDITI SVE SLOGOVE KOI IMAJU V 1 ILI V 2 KAO VREDNOSTI PRVOG SEKUNDARNOG KLJUČA I V 1 ILI V 2 KAO VREDNOSTI DRUGOG SEKUNDARNOG KLJUČA.

FUNKCIJA F 1 POSTAJE:

$$F_1=V(SK_1)\ 'I'\ V(SK_2)\ 'ILI'\ V(SK_1)\ 'I'\ V(SK_2)$$

$$'ILI'\ V(SK_2)\ 'I'\ V(SK_1)\ 'ILI'\ V(SK_2)\ 'I'\ V(SK_1)$$

KAKO SE U SVAKOM IMPLIKANTU FUNKCIJE F1 NALAZI SK-1 TO JE POTREBNO SLOGOVE SORTIRATI PREMA SADRŽAJU KLJUČA

POSTOJI I DRUGO REŠENJE, MOŽE SE SORTIRATI I PO OBRNUTOM REDOSLEDU KLJUČEVA. IZBOR REDOSLEDA KLJUČEVA PO KOJEM ĆE SE IZVRŠITI SORTIRANJE U SMISLU POVOLJNIJEG FIZIČKOG REDOSLEDA SLOGOVA ZAVISI I OD ODNOSA VREDNOSTI SEKUNDARNIH KLJUČEVA KAO I OD UKUPNOG BROJA VREDNOSTI SEKUNDARNIH KLJUČEVA.

FUNKCIJA F 2 SADRŽI 4 PROSTA IMPLIKANTA. PREMA OPISANOM POSTUPKU POTREBNO JE SORTIRATI I PO JEDNOM I PO DRUGOM SEKUNDARNOM KLJUČU. SVAKI KLJUČ POSEBNO PREDSTAVLJA REDOSLED, NA OSNOVU SKUPA PROSTIH IMPLIKANATA, PO KOJEM BI TREBALO SORTIRATI. REŠENJE JE UNIJA OVA SKUPA SLOGOVA KOJI SE DOBIJAJU PRETRAŽIVANJEM I PO JEDNOM I PO DRUGOM KLJUČU.

U PRAKSI SU VRLO ČESTI JEDNOSTAVNI I ILI ISKAZI TAKO DA SE PRETRAŽIVANJE MOŽE IZVESTI SORTIRANJEM PO JEDNOM ILI PO OVA REDOSLEDA SEKUNDARNIH KLJUČEVA. U TAKVIM USLOVIMA SORTIRANJA PRETRAŽIVANJE SORTIRANJEM DAJE DOBRE REZULTATE.

DRUGI PROBLEM U ODNOSU NA PRETRAŽIVANJE SORTIRANJEM JE VREME SORTIRANJA. JASNO JE DA AKO JE VREME SORTIRANJA INVERTOVANE LISTE I PRETRAŽIVANJE U SORTITANOJ INVERTOVANOJ LISTI DUŽE OD PRETRAŽIVANJA U NESORTIRANOJ INVERTOVANOJ LISTI ONDA PRETRAŽIVANJE SORTIRANJEM NEMA SMISLA.

DA BI SE IZBEGLA SORTIRANJA INVERTOVANE LISTE MOGU SE SORTIRANE INVERTOVANE LISTE, PREMA NAJČEŠĆIM ZAHTEVIMA PRETRAŽIVANJA, ČUVATI U MEMORIJI. AKO NIJE U PITANJU NEPRIHVATLJIV MEMORIJSKI PROSTOR ONDA SE OVAKVOM, KOMBINOVANOM METODOM PRETRAŽIVANJA, MOGU OSTVARITI ZNATNA POBOLJŠANJA U BRZINI PRETRAŽIVANJA.

SORTIRANE INVERTOVANE LISTE KOJE SE ČUVAJU MEMORISANE MOGU SE JEDNOSTAVNO INDEKSIRATI. INDEKSIRANJEM SORTITANIH INVERTOVANIH LISTI PRETRAŽIVANJE POSTAJE BRŽE. INDEKSIRANJE SE, TAKODJE MOŽE PRILAGODITI NAJČEŠĆIM ZAHTEVIMA PRETRAŽIVANJA. UMESTO INDEKSIRANJA GRUPA SE JEDNAKIM BROJEM SLOGOVA INDEKSIRANJE SE MOŽE VRŠITI PREMA VREDNOSTIMA ODREDJENOG REDOSLEDA SEKUNDARNIH KLJUČEVA.

5. SLOGOVI DATOTEKE KAO RETKO POSEDNUTA MATRICA

RETKO POSEDNUT MATRICE SMEŠTAJU SE TAKO ŠTO SE ELEMENT MATRICAE SA SVOJIM INDEKSIMA SMEŠTA U SLOG A ONDA SE SLOGOVI SMEŠTAJU JEDAN PORED DRUGOG. MEMORISANJE RETKO POSEDNUTIH MATRICA TAKVO DA SE POLOŽAJ ELEMENTA MATRICE IZRAČUNAVA NA OSNOVU INDEKSA MATRICE NAJČEŠĆE BI ZAHTEVALO VELIKI MEMORIJSKI PROSTOR KOJI JE NEPRIHVATLJIV SA STANOVIŠTA EFIKASNOSTI ORGANIZACIJE PODATAKA ZA OVAKVO MEMORISANJE KARAKTERISTIČNO JE DA SE USVOJI JEDAN REDOSLED INDEKSA KOJI SE MEMORIŠE U SVAKOM SLOGU. TIME SE OMOGUĆUJE, POSLE SORTIRANJA, JEDNOSTAVNIJE TRAŽENJE ELEMENATA MATRICE. POSLE SORTIRANJA MOGUĆE JE I INDEKSIRANJE SLOGOVA A NA OSNOVU INDEKSA MATRICE. SVAKI REDOSLED INDEKSA MATRICE, OVAJ REDOSLED JE KONSTANTAN ZA SVAKI SLOG, PREDSTAVLJA INTERPRETIRANJE MATRICE, PREMA [2], PREKO LINEARNIH LISTI(LISTE SA MALIM L). TO SU LINEARNE LISTE U ODNOSU NA INDEKS MATRICE KOJI SE NALAZI U POLJU KOJE SE KOD SORTIRANJA TRETIRA KAO POLJE NAJMANJE TEŽINE. OVE LINEARNE LISTE SE MEMORIŠU JEDNA PORED DRUGE ŠTO JE POSLEDICA SORTIRANJA.

SKUP ISTIH SLOGOVA, SVI SLOGOVI SADRŽE REDOSLED ISTIH POLJA, MOŽE SE SMATRATI MEMORISANOM RETKO POSEDNUTOM MATRICOM. SADRŽAJ JEDNOG POLJA POSTAJE ELEMENAT MATRICE DOK SU SADRŽAJI OSTALIH POLJA INDEKSI MATRICE. KAKO SU SADRŽAJI OSTALIH POLJA U OPŠTEM SLUČAJU ALFANUMERIČKI TO SE TABELAMA SADRŽAJI ZAMENE PRIRODNIM BROJEVIMA PA SE SLOGOVI MOGU SMATRATI RETKO POSEDNUTIM MATRICAMA. TIME SE ZA SLOGOVE DATOTEKE MOGU PRIMENJIVATI ORGANIZACIONA REŠENJA KOJA SE PRIMENJUJU KOD MATRICA.

INVERTOVANA LISTA PREDSTAVLJA MEMORISANU RETKO POSEDNUTU MATRICU, SA STANOVIŠTA ORGANIZACIJE ZA ELEMENT MATRICE NAJPOGODNIJE JE UZIMATI PRIMARNI KLJUČ DOK SU SEKUNDARNI KLJUČEVI, PREKO TABELA, INDEKSI ELEMENATA MATRICE. SORTIRANJEM INVERTOVANE LISTE PREMA NEKOM REDOSLEDU SEKUNDARNIH KLJUČEVA, PRIMARNI KLJUČ JE U POLJU NAJMANJE TEŽINE, INVERTOVANA LISTA SE PREDSTAVLJA FORMALNIM FIZIČKIM REDOSLEDOM LINEARNIH LISTI PREMA SEKUNDARNOM KLJUČU KOJI JE U POLJU NAJMANJE TEŽINE OD POLJA ZA SEKUNDARNE KLJUČEVE. SORTIRANJE PO DRUGOM REDOSLEDU SEKUNDARNIH KLJUČEVA PREDSTAVLJA FORMIRANJE NOVOG FORMALNOG REDOSLEDA LINEARNIH LISTI. DO SORTIRANJA KAO METODE PRETRAŽIVANJA MOŽE SE DOĆI I PREDSTAVLJANJEM INVERTOVANE LISTE PREKO FORMALNOG FIZIČKOG REDOSLEDA LINEARNIH LISTI.

6. DATOTEKE SE BRZIM PRETRAŽIVANJEM

DATOTEKA SA BRZIM PRETRAŽIVANJEM, PREMA [1], PREDSTAVLJAJU ORGANIZACIONA REŠENJA PRETRAŽIVANJA TAKVA DA JE OSNOVNI KRITERIJUM BRZINA ODGOVARA. U ORGANIZACIJI PODATAKA POVEĆANJEM INDEKSNOG PROSTORA SMANJUJE SE SREDNJI BROJ PRISTUPA PO SLOGU TE SE POVEĆAVA BRZINA PRISTUPA. ZA ZAHTEVE VELIKIH BRZINA INDEKSNO PODRUČJE POSTAJE VRLO VELIKO. KOD DATOTEKA SE BRZIM PRETRAŽIVANJEM NE POSTOJI PROBLEM VELIČINE INDEKSNOG PODRUČJA. BITNO JE DA PRETRAŽIVANJE BUDE ŠTO BRŽE, DA ODGOVOR BUDE U REALNOM VREMENU.

INDEKS NAGOMILAVANJA JE TAKVO UKAZIVANJE DA PO SVAKOJ VREDNOSTI SEKUNDARNOG KLJUČA POSTOJI SPISAK ADRESA (ILI PRIMARNIH KLJUČEVA) SVIH SLOGOVA KOJI SADRŽE TU ISTU VREDNOST SEKUNDARNOG KLJUČA. MOŽE POSTOJATI I RAZLIČITO INDEKSIRANJE U ODNOSU NA VREDNOSTI SEKUNDARNIH KLJUČEVA. PRETRAŽIVANJE U OVAKO ORGANIZOVANOJ DATOTECI JE JEDNOSTAVNO I VRLO BRZO A SVODI SA NA FORMIRANJE UNIJA ILI PRESEKA SKUPOVA PREMA INDEKSIMA NAGOMILAVANJA.

INDEKS NAGOMILAVANJA SE MOŽE MODIFIKOVATI U CILJU POBOLJŠANJA BRZINE PRETRAŽIVANJA. INVERTOVANA LISTA SORTIRANA PO NEKOM REDOSLEDU SEKUNDARNIH KLJUČEVA, PRIMERNI KLJUČ JE U POLJU NAJMANJE TEŽINE, PREDSTAVLJA INDEKS NAGOMILAVANJA U ODNOSU NA SEKUNDARNI KLJUČ KOJI SE NALAZI U POLJU NAJVEĆE TEŽINE. TO JE JEDNOSTAVAN POSTUPAK DOBIJANJA INDEKSA NAGOMILAVANJA. OVAKO PROŠIREN INDEKS NAGOMILAVANJA SADRŽI JOŠ I VREDNOSTI OSTALIH SEKUNDARNIH KLJUČEVA PO SVAKOM SLOGU. DA BI SE DOBIO POTPUNI SKUP INDEKSA NAGOMILAVANJA POTREBNO JE INVERTOVANU LISTU SORTIRATI VIŠE PUTA. BROJ SORTITANJA JEDNAK JE BROJU SEKUNDARNIH KLJUČEVA JER BI TREBALO DA SE SVAKI SEKUNDARNI KLJUČ NAĐE U POLJU NAJVEĆE TEŽINE.

REDOSLED SEKUNDARNIH KLJUČEVA PO POLJIMA ČIJA TEŽINA NIJE NI NAJVEĆA NI NAJMANJA MOŽE DA BUDE PROIZVOLJAN ALI MOŽE DA PREDSTAVLJA I ORGANIZACIJU PREMA NAJČEŠĆIM ZAHTEVIMA PRETRAŽIVANJA.

7. ZAKLJUČAK

TEKST PREDSTAVLJA INTERPRETACIJU VIŠEKLJUČNE ORGANIZACIJE PODATAKA SA STANOVIŠTA REDOSLEDA SLOGOVA KOJI JE UZET KAO OSNOVNI ELEMENT ORGANIZACIJE.

ZA INTERPRETIRANJE SORTIRANJA KAO METODE, SORTIRANJA KAO PREDPOSTAVKE PRETRAŽIVANJA, NIJE KORIŠĆENA NIKAKVA LITERATURA. BEZ LITERATURE JE POSMATRAN I PROBLEM FORMALIZOVANJA ISKAZNOG PRETRAŽIVANJA. U TOM SMISLU OTVORENI SU I PROBLEMI ČIJA SE REŠENJA MOGU SMATRATI IMPLIKACIJAMA OVOG TEKSTA.

OPISANI POSTUPAK GENERISANJA PREUZETOG INDEKSA NAGOMILAVANJA SA MOGUĆIM KORISNIM PROŠIRENJEM MOŽE SE ISKORISTITI U PROJEKTOVANJU DATOTEKA SA BRZIM PRETRAŽIVANJEM.

8. LITERATURA

1. J. MARTIN, COMPUTER DATA BASE ORGANIZATION , SECOND EDITION, PRENTICE-HALL, ENGLEWOOD CLIFFS, NEW JERSEY 07632.

2. H. WEDEKIND, ORGANIZACIJA PODATAKA, ZAK.

INFORMATICA 3/86

IMPLIKACIONI PROBLEM ZA VIŠEZNAČNE ZAVISNOSTI I MEHANIČKO DOKAZIVANJE TEOREMA II

Maleković Mirko
CVVTŠ „General arm. Ivan Gošnjak", Zagreb

UDK: 681.3.01:519

Specifikacija uvjeta koje mora zadovoljavati relaciona šema da bismo korektno modelirali neku pojavu predstavlja jedan od važnih problema u teoriji projektiranja relacionih baza podataka. Od posebnog interesa su uvjeti nazvani zavisnostima. Prilikom optimalne specifikacije dolazimo do implikacionog problema za dani skup zavisnosti. U ovom radu prezentiramo metod rješavanja implikacionog problema za višeznačne zavisnosti. Metod se bazira na reprezentiranju višeznačne zavisnosti pomoću Skolemove standardne forme i primjeni procedura dokazivanja koje su razvijene u teoriji mehaničkog dokazivanja teorema .

IMPLICATION PROBLEM FOR MULTIVALUED DEPENDENCIES AND MECHANICAL THEOREM PROVING:
The Specification of the constraints that the data must satisfy to model correctly the part of the world under consideration is one of the important issuses in the design theory of the relational database schemas. Of particular interest are the constraints called data dependencies. In this work we present a method for solving of implication problem for multivalued dependencies. The method is based on representation of multivalued dependencies by Skolem's standard forms and aplication of proof procedures which are developed in mechanical theorem proving.

1. Uvod

U radu [4] , prezentirali smo metod rješavanja implikacionog problema za funkcionalne zavisnosti. Sada proširujemo navedeni metod i na višeznačne zavisnosti. Kako su višeznačne zavisnosti suptilnije prirode nego funkcionalne zavisnosti, Skolemova standardna forma višeznačne zavisnosti je općenito složenija od iste forme funkcionalne zavisnosti, dokazi teorema, tj. rješenja implikacionih problema, su znatno kompleksniji u slučaju višeznačnih zavisnosti nego u slučaju samo funkcionalnih zavisnosti. Kompleksnost procedura dokazivanja za višeznačne zavisnosti očituje se ne samo u dužini dokaza već i u poteškoćama vezanim za manipuliranje skupovima atributa. Navedene poteškoće su ozbiljna prepreka u potpunoj automatizaciji rješenja implikacionog problema za višeznačne zavisnosti.
Za potpuno razumijevanje članka, pretpostavlja se poznavanje teorije projektiranja relacionih baza podataka na nivou [5] i rezolucijskih procedura dokazivanja na nivou [1] ili [3] .

2. Implikacioni problem za višeznačne zavisnosti

U ovoj sekciji uvodima pojam višeznačne zavisnosti i opisujemo implikacioni problem .

Definicija Neka je $U = \{ A_1, .., A_n \}$ konačan skup atributa, $R(A_1, .., A_n)$ relacione šema, nad U , $X, Y \subseteq U$. Izraz oblika $X \twoheadrightarrow Y$ zovemo višeznačna zavisnost.

U navedenom slučaju kažemo da X višeznačno određuje Y.

Definicija Neka je relacija r primjer relacione šeme $R(A_1,..,A_n)$. Kažemoda $X \twoheadrightarrow Y$ vrijedi (ili da je zadovoljeno) u r ako i samo

$$\forall t_1, t_2 \in r\ (t_1[X] = t_2[X] \Rightarrow \exists t_3 \in r(t_1[X] = t_3[X] \wedge t_2[X] = t_3[X] \wedge t_1[Y] = t_3[Y] \wedge t_2[\overline{XY}] = t_3[\overline{XY}]))$$

Napomena U skladu sa uobičajenom rotacijom u teoriji baza podataka, pišemo jednočlan skup $\{A\}$ kao A, komplement $X^c = U \setminus X$ kao $\overline{X}$, te uniju $X \cup Y$ skupova X i Y atributa kao XY .

Definicija Kažemo da je relacija r model od $X \twoheadrightarrow Y$ ako i samo ako je $X \twoheadrightarrow Y$ zadovoljeno u r.

Istaknimo da se relacija r konvencionalno predstavlja tabelom, redovi tabele su elementi (tiplovi) od r, a stupci su imenovani atributima iz relacione šeme $R(A_1, A_2, .., A_n)$.

Definicija Kažemo da $X \twoheadrightarrow Y$ vrijedi u relacionoj šemi $R(A_1, .., A_n)$ ako i samo ako je svaki primjer r relacione šeme $R(A_1, .., A_n)$ model od $X \twoheadrightarrow Y$.

Uvedimo skup $MVD(U) = \{ X \twoheadrightarrow Y \,/\, X, Y \subseteq U \}$

Definicija Neka je $V \subseteq MVD(U)$. Za relaciju r ćemo reći da je model od V ako i samo ako je model svakog člana iz V.

Sa M(V) ćemo označavati skup svih modela od V. Lako uočavamo da vrijedi $M(X \rightarrow Y) \subseteq M(X \twoheadrightarrow Y)$

Definicija (logičke implikacije) Neka je $V \subseteq MVD(U)$, $v \in MVD(U)$
Kažemo da V logički implicira v ako i samo ako svaki model od V jeste i model od v .

Da V logički implicira v označavat ćemo sa $V \models v$ ili $\frac{V}{v}$ Notaciju $\frac{V}{Y}$ upotrijebit ćemo u sekciji 4. zbog pogodnosti.
Za dane V i v implikacioni problem je da testiramo da li $V \models v$.
Kako zahtjev za optimalnom specifikacijom skupa zavisnosti vodi na implikacioni problem opisano je u radu [4] .

3. Reprezentacija višeznačne zavisnosti pomoću Skolemove standardne forme

U prošloj sekciji smo rekli da $X \twoheadrightarrow Y$ vrijedi u r ako i samo ako $\forall t_1, t_2 \in r \; (t_1[X] = t_2[X] \Longrightarrow$
$\Rightarrow \exists t_3 \in r \; (t_1[X] = t_3[X] \wedge t_2[X] = t_3[X] \wedge$
$\wedge t_1[Y] = t_3[Y] \wedge t_2[\overline{XY}] = t_3[\overline{XY}]))$.

Za Z U uvodimo predikat E_Z, gdje $E_Z(t_1, t_2)$ znači jednakpst tiplova t_1 i t_2 na skupu atributa Z tj. $t_1[Z] = t_2[Z]$.
Pomoću predikata E_X, E_Y, $E_{\overline{XY}}$, višeznačnu zavisnost $X \twoheadrightarrow Y$ prikazujemo u obliku

$$X \twoheadrightarrow Y : (1) \forall t_1 \forall t_2 [E_X(t_1, t_2) \Rightarrow \exists t_3 (E_X(t_1, t_3) \wedge$$
$$\wedge E_X(t_2, t_3) \wedge E_Y(t_1, t_3) \wedge E_{\overline{XY}}(t_1, t_3))] .$$

Prema tome formulu (1) interpretiramo na skupu tiplova (relaciji) r, tj. domena interpretacije D je skup tiplova r.

Pr.1. Neka je zadana relaciona šema R(A,B,C) koja je u danom trenutku vremena predstavljena relacijom r:

r	A	B	C
	a_1	b_1	c_1
	a_1	b_1	c_2

Lako provjerimo da je r model za $A \twoheadrightarrow B$.
Također, r je model i za $AB \twoheadrightarrow C$.
U slijedećoj sekciji ćemo pokazati da $X \rightarrow Y \models X \twoheadrightarrow Y$. Da ne vrijedi obrat pokazuje upravo zadana relacija r; u r vrijedi $B \twoheadrightarrow C$, ali ne vrijedi $B \rightarrow C$.
Skolemizacijom formule (1) nalazim Skolemovu standardnu formu od $X \twoheadrightarrow Y$.

$$(2)\ SSF(X \twoheadrightarrow Y) \begin{cases} \sim E_X(t_1, t_2) \vee E_X(t_1, f(t_1, t_2)) \\ \sim E_X(t_1, t_2) \vee E_X(t_2, f(t_1, t_2)) \\ \sim E_X(t_1, t_2) \vee E_Y(t_1, f(t_1, t_2)) \\ \sim E_X(t_1, t_2) \vee E_{XY}(t_2, f(t_1, t_2)) \end{cases}$$

Isti postupak primijenjen na $\sim(X \twoheadrightarrow Y)$ daje

$$(3)\ SSF(\sim(X \twoheadrightarrow Y)) \begin{cases} E_X(a,b) \\ \sim E_X(a,t_3) \vee \sim E_X(b,t_3) \vee \\ \vee \sim E_Y(a,t_3) \vee \sim E_{\overline{XY}}(b,t_3) \end{cases}$$

4. Primjeri rješavanja implikacionog problema pomoću rezolucijskih procedura dokazivanja

U ovoj sekciji ćemo na primjerima pokazati točnost nekih pravila formalnog sistema V_A za funkcionalne i višeznačne zavisnosti. Dokaz točnosti pravila vodi na implikacioni problem. Ocrtajmo ukratko opći postupak dokazivanja.

Neka je $C \subseteq FD(U) \cup MVD(U)$, $f \in FD(U) \cup VD(U)$ i neka treba riješiti implikacioni problem $\frac{C}{f}$

Negirajući f i transformirajući $C \wedge \sim f$ u Skolemovu standardnu formu, dobit ćemo skup rečenica S.

Sada na skup S primjenjujemo odgovarajuću proceduru dokazivanja sa ciljem da izvedemo kontradikciju koja će biti označena sa $\square$.

Ako uspijemo, dokazali smo da je S kontradiktoran skup rečenica, odnosno da je zaista $\frac{C}{f}$.

U dokazima koji slijede skup C će osim zavi-

snosti sadržavati i neke uobičajene relacije na skupovima atributa. Također, skup S ćemo trebati proširiti sa aksiomima koji karakteriziraju predikat E_Z. Proširenje skupa S ćemo označiti sa S'. Rješenje implikacionog problema svodi se onda na dokazivanje kontradiktornosti skupa S'.

Primjer 4.1. Dokazujemo $\dfrac{X \longrightarrow Y}{X \twoheadrightarrow Y}$

Transformiramo $X \longrightarrow Y \wedge \sim(X \twoheadrightarrow Y)$ u Skolemovu standardnu formu. Dobivamo skup S koji se sastoji od slijedećih rečenica:

$$X \longrightarrow Y \begin{cases} (1)\ \sim E_X(t_1,t_2) \vee E_Y(t_1,t_2) \\ \sim(X \twoheadrightarrow Y) \begin{cases} (2)\ E_X(a,b) \\ (3)\ \sim E_X(a,t_3) \vee \sim E_X(b,t_3) \vee \sim E_{\overline{XY}}(b,t_3) \end{cases} \end{cases} S$$

Proširimo skup S šemom aksioma

$$(4)\quad E_X(t,t)$$

Sada, dokazujemo kontradiktornost skupa $S' = S \cup \{(4)\}$. Stablo dokaza je dano na slici 1. .

Primjer 4.2. Dokazujemo $\dfrac{X \twoheadrightarrow Y}{X \twoheadrightarrow \overline{XY}}$ (pravilo komplementiranja)

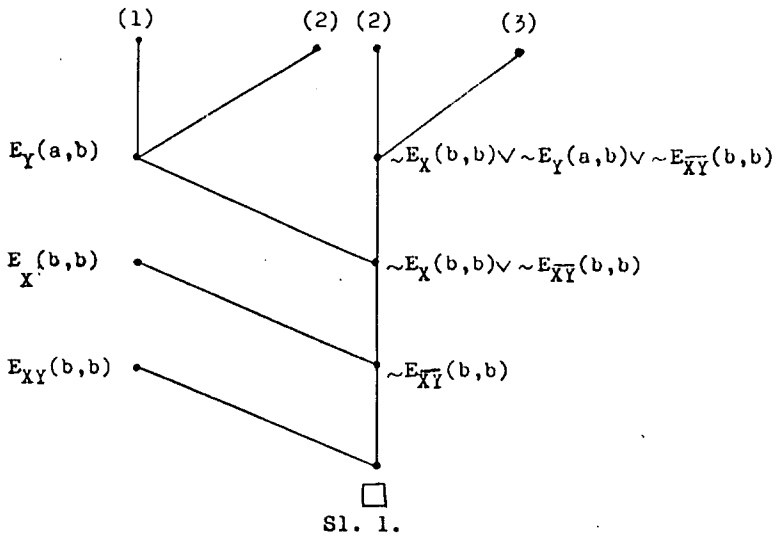
Transformacija $X \twoheadrightarrow Y \wedge \sim(X \twoheadrightarrow \overline{XY})$ daje skup S:

$$X \twoheadrightarrow Y \begin{cases} (1)\ \sim E_X(t_1,t_2) \vee E_X(t_1,f(t_1,t_2)) \\ (2)\ \sim E_X(t_1,t_2) \vee E_X(t_2,f(t_1,t_2)) \\ (3)\ \sim E_X(t_1,t_2) \vee E_Y(t_1,f(t_1,t_2)) \\ (4)\ \sim E_X(t_1,t_2) \vee E_{\overline{XY}}(t_2,f(t_1,t_2)) \\ \sim(X \twoheadrightarrow \overline{XY}) \begin{cases} (5)\ E_X(a,b) \\ (6)\ \sim E_X(a,t_3) \vee \sim E_X(b,t_3) \vee \sim E_{\overline{XY}}(a,t_3) \vee \\ \qquad \vee \sim E_{\overline{X\overline{XY}}}(b,t_3) \end{cases} \end{cases} S$$

Kako je $\overline{X\overline{XY}} = \overline{X} \cap Y$, rečenica (6) ima oblik :

$$(6')\ \sim E_X(a,t_3) \vee \sim E_X(b,t_3) \vee \sim E_{\overline{XY}}(a,t_3) \vee \sim E_{\overline{X} \cap Y}(b,t_3)\ .$$

Dalje, jer je $E_Y(t_1,f(t_1,t_2)) \equiv E_{\overline{X} \cap Y}(t_1,f(t_1,t_2)) \wedge E_Y(t_1,f(t_1,f(t_1,t_2)))$, rečenicu (3) transformiramo u ekvivalentnu



Sl. 1.

formu (3') $\sim E_X(t_1,t_2) \vee E_{\overline{X} \cap Y}(t_1,f(t_1,t_2))$.

(3'') $\sim E_X(t_1,t_2) \vee E_Y(t_1,f(t_1,t_2))$

Sada pokazujemo kontradiktornost skupa $S' = \{(1),(2),(3'),(3''),(4),(5),(6')\}$ koji je ekvivalentan sa skupom S . Stablo dokaza sa komentarom je dano na slici 2.

Primjer 4.3. dokazujemo da vrijedi

$$\frac{X \twoheadrightarrow Y,\ Z \subseteq Y,\ W \cap Y = \emptyset,\ W \longrightarrow Z}{X \longrightarrow Y}$$

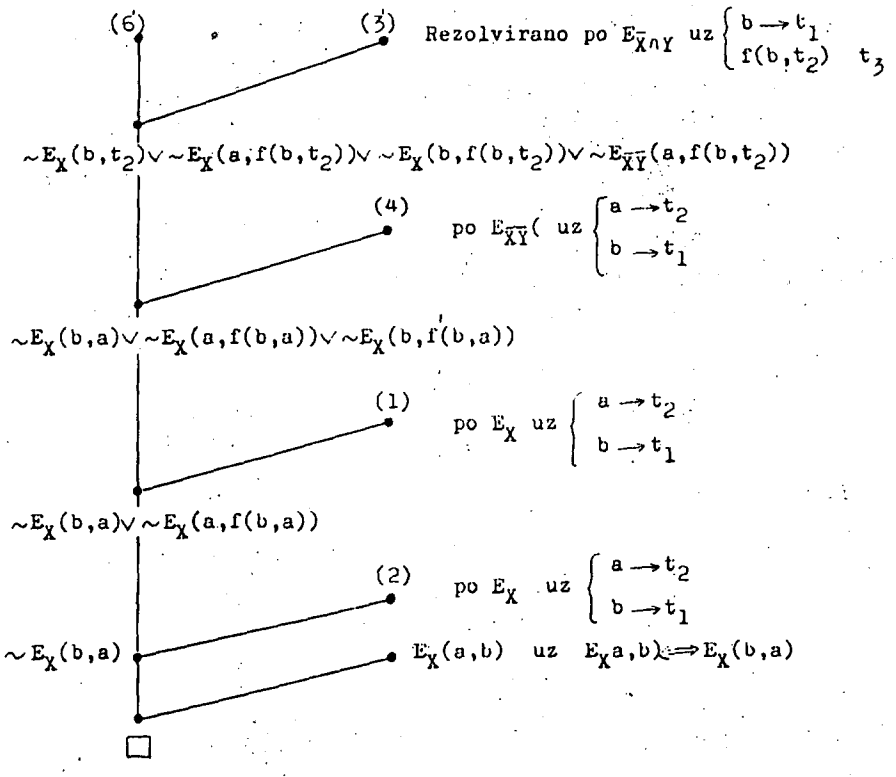
Kako je $Z \subseteq Y$ vrijedi $Y \longrightarrow Z$. Transformiramo $X \twoheadrightarrow Y \wedge W \longrightarrow Z \wedge Y \longrightarrow Z \wedge \sim(X \longrightarrow Z)$ u standardnu formu i dobivamo skup S:

$$X \twoheadrightarrow Y \begin{cases} (1)\ \sim E_X(t_1,t_2) \vee E_X(t_1,f(t_1,t_2)) \\ (2)\ \sim E_X(t_1,t_2) \vee E_X(t_2,f(t_1,t_2)) \\ (3)\ \sim E_X(t_1,t_2) \vee E_Y(t_1,f(t_1,t_2)) \\ (4)\ \sim E_X(t_1,t_2) \vee E_{\overline{XY}}(t_2,f(t_1,t_2)) \end{cases}$$

$$W \longrightarrow Z\ \{(5)\ \sim E_W(t_1,t_2) \vee E_Z(t_1,t_2)$$

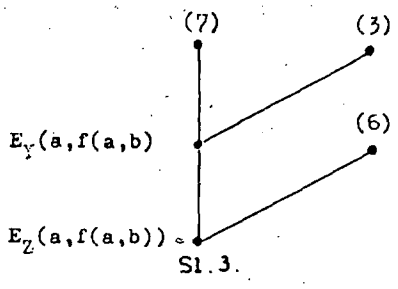
$$Y \longrightarrow Z\ \{(6)\ \sim E_Y(t_1,t_2) \vee E_Z(t_1,t_2)$$

$$\sim(X \longrightarrow Z) \begin{cases} (7)\ E_X(a,b) \\ (8)\ \sim E_Z(a,b) \end{cases} \Big\} S$$

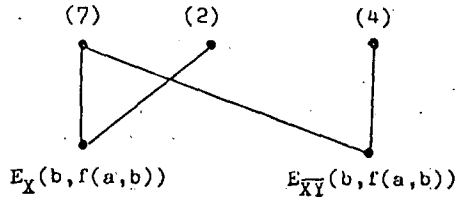


Sl. 2.

Slijedi stablo dokaza:

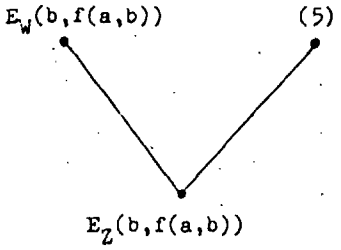


Sl.3.



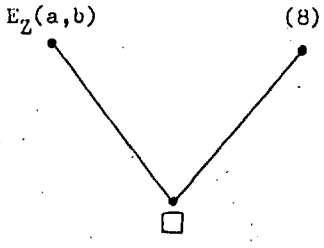
Sl.4.

Intervencija 1.
Sada ćemo iskoristiti pretpostavku $W \cap Y = \emptyset$.
Naime, $W \cap Y = \emptyset \Rightarrow W = (W \cap X) \cup \overline{XY}$. Zato,
$(E_X(b,f(a,b)) \wedge E_{\overline{XY}}(b,f(a,b))) \Longrightarrow E_W(b,f(a,b))$.



Sl.5.

Intervencija 2.
Iz $E_Z(b,f(a,b))$ i $E_Z(a,f(a,b))$ po tranzitivnosti dobijemo $E_Z(a,b)$.



Sl.6.

Prokomentirajmo navedeni dokaz.Stablo dokaza je dano na slikama 3., 4.,5 i 6.. Vidimo da je stablo dokaza prekinuto intervencijama 1. i 2. . I dok je intervencija 2. (tranzitivnost od E_Z) mogla biti aksiomatizirana i dodana skupu S , intervencija 1. je problematično mjesto sa stanovišta potpune automatizacije procedure dokazivanja. Da intervencije mogu biti još kompleksnije pokazuje slijedeći primjer.

Primjer 4.4. Dokazujemo $\dfrac{X \twoheadrightarrow Y,\ X \twoheadrightarrow Z}{X \twoheadrightarrow YZ}$

Iskazani teorem predstavlja pravilo unije. Skolemizacijom $X \twoheadrightarrow Y \wedge X \twoheadrightarrow Z \wedge \sim(X \twoheadrightarrow YZ)$, nalazimo skup S:

$X \twoheadrightarrow Y$
- (1) $\sim E_X(t_1,t_2) \vee E_X(t_1,f(t_1,t_2))$
- (2) $\sim E_X(t_1,t_2) \vee E_X(t_2,f(t_1,t_2))$
- (3) $\sim E_X(t_1,t_2) \vee E_Y(t_1,f(t_1,t_2))$
- (4) $\sim E_X(t_1,t_2) \vee E_{\overline{XY}}(t_2,f(t_1,t_2))$

$X \twoheadrightarrow Z$
- (5) $\sim E_X(t_1,t_2) \vee E_X(t_1,g(t_1,t_2))$
- (6) $\sim E_X(t_1,t_2) \vee E_X(t_2,g(t_1,t_2))$
- (7) $\sim E_X(t_1,t_2) \vee E_Z(t_1,g(t_1,t_2))$
- (8) $\sim E_X(t_1,t_2) \vee E_{\overline{XZ}}(t_2,g(t_1,t_2))$

$\sim(X \twoheadrightarrow YZ)$
- (9) $E_X(a,b)$
- (10) $\sim E_X(a,t_3) \vee \sim E_X(b,t_3) \vee \sim E_Y(a,t_3) \vee \vee \sim E_Z(a,t_3) \vee \sim E_{\overline{XYZ}}(b,t_3)$

} S

U rečenici (10) iskorištena je ekvivalencija $\sim E_{YZ}(a,t_3) \Longleftrightarrow \sim E_Y(a,t_3) \vee \sim E_Z(a,t_3)$, koja ustvari predstavlja razbijanje unije YZ na pojedinačne skupove Y,Z. Pravilo razbijanja treba respektirati prilikom formiranja skupa S.

Intervencija 1.

Želimo doći do tipla w za koji je

(11) $E_X(a,w) \wedge E_Y(a,w) \wedge E_Z(a,w) \wedge E_{\overline{XYZ}}(b,w)$,

što je u kontradikciji sa (10).

Početak stabla dokaza dan je na slici 7.

Intervencija 2.

Pokazujemo da je traženi tipl w=g(a,f(a,b)) tj. da vrijedi:

(a) $E_X(a,g(a,f(a,b)))$

(b) $E_X(b,g(a,f(a,b)))$

(c) $E_Y(a,g(a,f(a,b)))$

(d) $E_Z(a,g(a,f(a,b)))$

(e) $E_{\overline{XYZ}}(b,g(a,f(a,b)))$

(a) i (d) su rezolvente (12), odnosno (14). (b) slijedi iz (9) i (a) zbog simetričnosti i tranzitivnosti predikata E_X. Ostaje pokazati (c) i (e).

Dokazujemo (c): Trebamo dokazati $E_Y(a,g(a,f(a,b)))$. Vrijedi

$E_Y(a,f(a,b)) \wedge E_X(f(a,b),g(a,f(a,b))) \Longrightarrow$

$\Longrightarrow E_{X \cap Y}(a,g(a,f(a,b)))$.

Prema tome, dobili smo rečenicu

(16) $E_{X \cap Y}(a,g(a,f(a,b)))$.

Kažemo da je (16) dobiveno po tranzitivnosti za presjek.

Pokažimo još da je

(17) $E_{Y \setminus X}(a,g(a,f(a,b)))$, odnosno da je

$E_A(a,g(a,f(a,b))) \ \forall\, A \in Y \setminus X$.

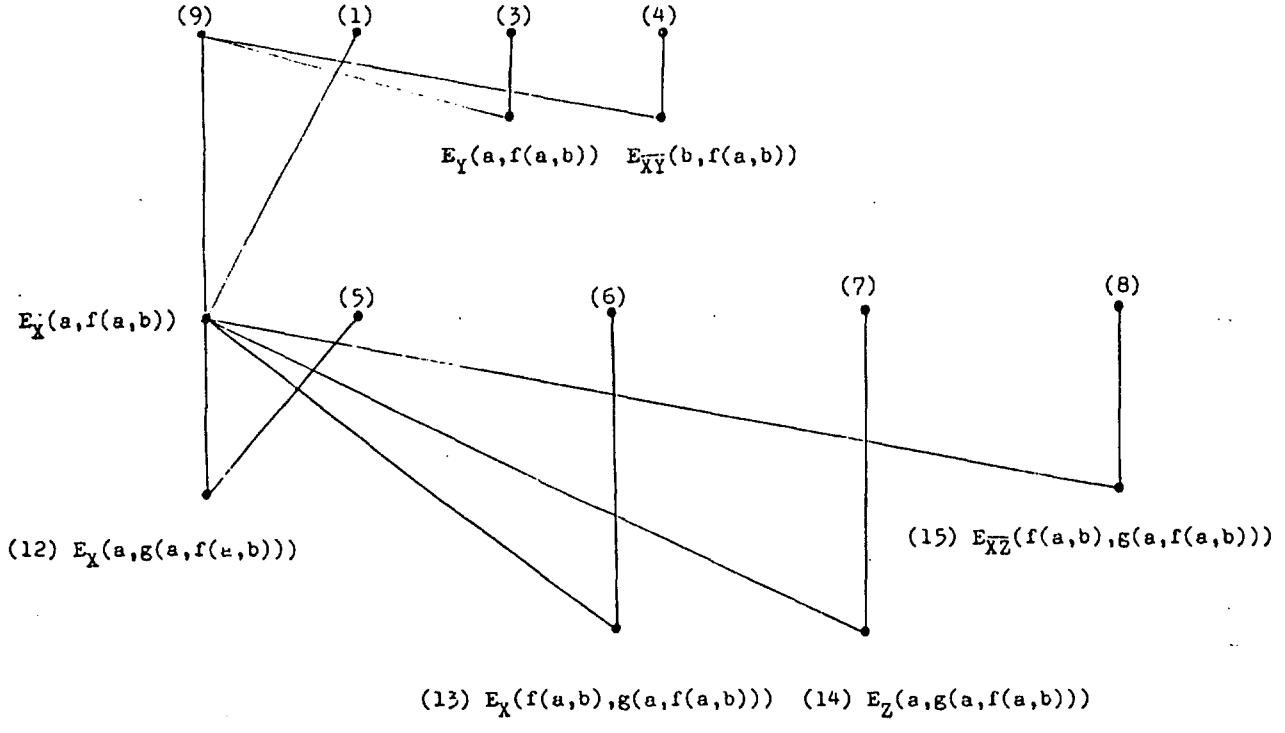
Dokaz je kao što slijedi:

$A \in Y \setminus X \Longrightarrow A \in Y \wedge A \notin X \wedge (A \in Z \vee A \notin Z) \Longrightarrow E_A(a,f(a,b)) \wedge$

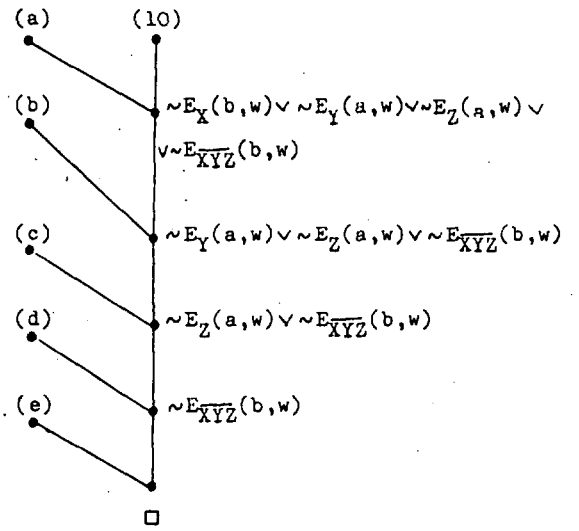
$\wedge (E_A(a,g(a,f(a,b))) \vee E_A(f(a,b),g(a,f(a,b)))) \Longrightarrow$

$\Longrightarrow E_A(a,g(a,f(a,b)))$.

Zbog proizvoljnosti atributa A, (17) vrijedi.



Konačno, iz (16) i (17) slijedi (c).
Dokazujemo (e): Trebamo dokazati $E_{\overline{XYZ}}(b,g(a,f(a,b)))$.
Po tranzitivnosti za presjek $\overline{XY} \cap \overline{XZ}$, nalazimo $(E_{\overline{XY}}(b,f(a,b)) \wedge E_{\overline{XZ}}(f(a,b),g(a,f(a,b)) \Rightarrow$
$\Rightarrow E_{\overline{XY} \cap \overline{XZ}}(b,g(a,f(a,b)))$, a onda iz
$\overline{XYZ} \subseteq \overline{XY} \cap \overline{XZ}$ dobivamo $E_{\overline{XYZ}}(b,g(a,f(a,b)))$.
Na ovaj način smo dobili (11) sa $w=g(a,f(a,b))$. Sada, formalno, rezolvirajući (10) i (11) dobijemo dokaz kao na slici 8.



Sl. 8.

5. Zaključak

U ovom radu smo prezentirali metod rješavanja implikacionog problema za višeznačne zavisnosti. Kao i u članku [4], metod se bazira na transformaciji višeznačne zavisnosti u Skolemovu standardnu formu i primjeni rezolucijskih procedura dokazivanja. Primjeri u sekciji 4. ukazuju na potrebu intervencija unutar procedure dokazivanja, a time i na probleme u automatizaciji projektiranja relacionih šema, što s druge strane neki autori, [2], smatraju ultimativnim ciljem istraživanja u teoriji zavisnosti.

Literatura

1. Chang, C.L., Lee, R.C.T.: Symbolic Logic and Mechanical Theorem Proving, Compt Sci. Appl. Math. Academic Press, 1973

2. Gallare, H., Minker, J., Nicolas, J.-M.: Advances in Data Base Theory, vol. 1. Plenum Press, New York, 1981.

Loveland, D.: Automated Theorem Proving: A Logical Basis, North Holland Publ. CO. New York, 1978.

4. Maleković, M.: Implikacicni problem za funkcionalne zavisnosti i mehaničko dokazivanje teorema.

5. Ullman, J.D.: Principles of Database Systems, Computer Science Press, Potomac, MD, 1980.

OCENA PROGRAMSKIH ORODIJ IDA

B. Vilfan, V. Mahnič, T. Mohorič
Univerza Edvarda Kardelja v Ljubljani
Fakulteta za elektrotehniko
Laboratorij za programsko opremo

UDK: 681.326.7:IDA

POVZETEK: V članku podajamo opis in oceno programskih orodij IDA-Baza, IDA-Cosen in IDA-Ekran s posebnim poudarkom na najpomembnejšem izmed njih, t.j. sistem za upravljanje podatkovnih baz IDA-Baza. Karakteristike, po katerih je izvedeno ocenjevanje, so razdeljene na logični nivo, interni nivo, pripomočke za upravljanje podatkovne baze in pripomočke za uporabljanje podatkovne baze s strani končnega uporabnika. Izvedena je tudi primerjava s sorodnim sistemom za upravljanje podatkovnih baz Ultra. V zaključku navajamo še nekaj predlogov za nadaljnji razvoj in izboljšave programskih orodij IDA.

ABSTRACT: The paper presents a description and evaluation of programming tools IDA-Baza, IDA-Cosen, and IDA-Ekran. The special attention is given to the most important of them - DBMS IDA-Baza. The characteristics on which the evaluation is based are divided in logical level, internal level, tools for database management, and tools for end-user database usage. Also the comparison with related DBMS Ultra is outlined. As a conclusion some guide-lines for further development and improvement of IDA programming tools are proposed.

1. UVOD

Sistemi za upravljanje podatkovnih baz postajajo vedno pomembnejša programska orodja, brez katerih si ne moremo več predstavljati razvoja sodobnih, računalniško podprtih informacijskih sistemov. Razen tega pa smo v zadnjih letih priča tudi naglemu razvoju drugih programskih pripomočkov (jeziki za povpraševanje, generatorji poročil, programski generatorji itd.), ki v tesni povezavi s sistemi za upravljanje podatkovnih baz bistveno poenostavljajo postopek programiranja, s tem pa tudi povečujejo produktivnost pri razvoju novih aplikacij.

Sledeč tem tendencam v razvoju uporabniške sistemske programske opreme je DO Iskra Delta razvila več programskih pripomočkov, ki so znani pod skupnim imenom IDA (Iskra Delta Arhitektura). Osrednjo vlogo med njimi igra prav gotovo sistem za upravljanje podatkovnih baz IDA-Baza, predmet naše ocene pa sta še IDA-Cosen (generator cobolskih programov) in IDA-Ekran (generator zaslonskih slik).

Izmed sorodnih produktov, ki jih je moč uporabljati na računalnikih serije VAX-11 in so bili razviti v tujini, je pri nas najbolj znan sistem za upravljanje podatkovnih baz Ultra ameriške firme Cincom Systems.

V tem članku želimo podati kratek opis in oceno programskih orodij IDA ter njihovo primerjavo s sistemom Ultra. Članek predstavlja povzetek rezultatov obširnejše analize obeh sistemov, ki jo je za potrebe DO Iskra Delta opravil Laboratorij za programsko opremo na Fakulteti za elektrotehniko v Ljubljani. Podrobni rezultati omenjene analize so zbrani v posebni študiji z naslovom Ocena programskih orodij IDA.

Poudariti želimo, da je bila analiza (na predlog naročnika) izvršena zgolj na podlagi podatkov, ki so dostopni v priročnikih, nismo pa imeli možnosti, da bi oba sistema preizkusili v praksi. Zato tudi ne podajamo ocen o performansah, temveč ocenjujemo predvsem funkcionalne karakteristike.

2. OPREDELITEV KARAKTERISTIK, PO KATERIH OCENJUJEMO SISTEME ZA UPRAVLJANJE PODATKOVNIH BAZ

Da bi lahko izvršili oceno programskih orodij IDA in njihovo primerjavo z Ultro, smo najprej definirali karakteristike, po katerih bomo ocenjevali oba sistema za upravljanje podatkovnih baz. Razdelili smo jih na 4 skupine:

A. Logični nivo

- konceptualni podatkovni model in njegova sposobnost modelirati stvarnost
- z modelom podprte integritetne omejitve
- uporabniški modeli in njihova sposobnost modeliranja uporabniških pogledov na stvarnost
- fleksibilnost povezave med konceptualnim modelom in uporabniškimi modeli
- logična podatkovna neodvisnost

B. Interni nivo

- struktura datotek in pristopne metode
- fizična podatkovna neodvisnost
- sočasnost dostopa do podatkov
- zaščitni mehanizmi za dostop do podatkov in za obnovo podatkov po nesrečah
- žurnal in njegova uporabnost
- kontrolirana redundantnost

C. Pripomočki za upravljanje podatkovne baze

- generiranje in spreminjanje konceptualne sheme
- generiranje in spreminjanje internih shem
- uglaševanje podatkovne baze z ozirom na povprečni pristopni čas
- migracija podatkov
- nadzor nad delovanjem sistema za upravljanje podatkovnih baz

D. Pripomočki za uporabljanje podatkovne baze s strani končnega uporabnika

- jeziki za poizvedovanje
- vmesniki na visokonivojske gostiteljske jezike
- generiranje zunanjih shem v uporabniških programih
- generiranje zaslonskih slik

3. OCENA PROGRAMSKIH ORODIJ IDA

3.1 Logični nivo

Sistem za upravljanje podatkovne baze IDA-Baza sloni v logičnem pogledu na mrežnem podatkovnem modelu, ki pripada površinskemu tipu podatkovnih modelov in temelji na konceptih

- tip podatkovnega elementa (Data Item Type)
- tip zapisa (Record Type)
- tip seta (Set Type)

kar je v skladu s specifikacijami mrežnega podatkovnega modela, kot ga je specificiral CODASYL, z izjemami:

- v setu lahko nastopa le en tip zapisa kot član;
- vsi tipi zapisov, ki so člani seta, morajo obvezno vsebovati primarni ključ lastnika seta;
- na uporabniškem nivoju se lahko pojavlja določeni zapis kot član le v enem setu, če naj nastopa kot član v dveh ali več setih, ga je potrebno večkratno definirati.

Z opisanim mrežnim podatkovnim modelom lahko predstavimo tipe močnih entitet in njihove povezave s tipi šibkih entitet s pomočjo tipov zapisov, v katerih je ime tipa zapisa običajno ime tipa močne entitete. Povezave med tipi močnih entitet pa so predstavljene s tipi setov, pri čemer ime povezave postane ime tipa seta.

Model podpira dve vrsti integritetnih omejitev:

- iz definicije ključa v tipu zapisa sledi funkcionalna odvisnost šibkih entitet od močne entitete, v katere zapisu nastopajo;
- iz definicije seta sledi integritetna omejitev, da vsaki močni entiteti, ki je predstavljena z zapisom - članom, pripada natanko ena močna entiteta, ki je predstavljena z zapisom - lastnikom.

Podshema, s katero je opisan uporabniški pogled, je podmnožica globalnega logičnega opisa podatkov oziroma konceptualne sheme. Sestavlja jo lista zapisov, ki jih uporabnik namerava uporabljati in so definirani v konceptualni shemi. V okviru vsakega tipa zapisa je možno navesti samo tipe podatkovnih elementov, ki uporabnika zanimajo, ostale pa izpustiti. Za vsak tip zapisa pa lahko navedemo tipe setov, ki jih uporabnik želi uporabljati za dostop do zapisov.

Za uporabniški pogled velja, da mora biti zasnovan na istem podatkovnem modelu kot konceptualna shema, tako da uporabnik nima možnosti izbire podatkovnega modela, ki bi morda bolj ustrezal njegovi aplikaciji. Prav tako je uporabnik vezan na uporabo imen tipov zapisov in imen tipov podatkovnih elementov, kot so definirana v konceptualni shemi.

Logična podatkovna neodvisnost je v pretežni meri pogojena z mrežnim podatkovnim modelom, na katerem je osnovana konceptualna shema. S tega stališča je najvišja, ki jo je možno doseči. Uporabniške sheme in s tem uporabniške programe je treba spremeniti le v primerih, če spremembe v konceptualni shemi zadevajo tipe zapisov in tipe setov, ki jih uporabniški program uporablja.

3.2 Interni nivo

Fizično podatkovno bazo sestavljajo datoteke, imenovane vsebovalniki (containers), ki so pod kontrolo operacijskega sistema in lahko vsebujejo eno ali več logičnih datotek v celoti ali pa delno. Vsebovalnik je razdeljen na logične bloke. Logični blok lahko vsebuje le zapise istega tipa, obsega pa lahko enega ali več fizičnih blokov - sektorjev na disku. Vsak zapis zaseda v logičnem bloku svojo celico (cell), ki je adresabilna.

Seti so fizično predstavljeni s pomočjo kazalcev "naprej" in "nazaj", ki omogočajo dostop do naslednjega oziroma predhodnega zapisa v setu. Žal ne obstajajo tudi kazalci na lastnike setov. Prehod s člana seta na lastnika lahko izvedemo le s pomočjo ključa seta, ki je istočasno tudi ključ zapisa - lastnika.

Spremembe v fizični organizaciji podatkovne baze, ki niso pogojene s spremembami v konceptualnem podatkovnem modelu, se lahko nanašajo na povečanje obsega posameznih datotek, prerazporejanje vhodno izhodnih vmesnikov, razmeščanje vsebovalnikov na različne diskovne enote in spreminjanje velikosti logičnih blokov.

Naštete spremembe ne vplivajo na logično strukturo podatkovne baze in uporabniške programe, zato lahko ocenimo fizično podatkovno neodvisnost relativno visoko.

V sistemu IDA-Baza lahko sočasno deluje nad podatkovno bazo do 52 oz. do 99 uporabniških programov. Konsistentnost podatkovne baze se ščiti z zaklepanjem podatkov. Najmanjši del podatkov, ki se ga da zakleniti za ekskluzivno rabo, je zapis. Uporabnik lahko zaklene tudi več zapisov zaporedi, kar mu omogoča izvesti večfazno ažuriranje brez nevarnosti za konsistentnost podatkovne baze. Pri večfaznem ažuriranje predvideva sistem vključevanje sprememb v podatkovno bazo za vse faze ažuriranja skupaj ali pa njihovo zanemarjanje z ukazoma Pomni (COMMIT) in Pozabi (CANCEL).

Problem mrtve zanke (dead lock) je rešen s časovno omejitvijo zaklenitve zapisa, pri čemer je čas zaklenitve določljiv s strani upravitelja podatkovne baze, za uporabniške programe, vpletene v mrtvo zanko, pa sistem za upravljanje podatkovnih baz izda ukaz Pozabi.

Zaščitne mehanizme za dostop do podatkov lahko razdelimo na dva dela:

Prvi del se nanaša na pristopne kontrole, ki temelje na geslih, in pravice do spreminjanja podatkov (samo branje ali tudi dodajanje, brisanje in modifikacija zapisov). Podeljevanje teh pravic je v pristojnosti upravitelja podatkovne baze.

Drugi del zaščite se nanaša na restavriranje podatkovne baze v situacijah, ko le-ta zaide v nekonsistentno stanje. IDA-Baza uporablja v ta namen dve vrsti beleženja aktivnosti nad podatkovno bazo s pomočjo dnevnikov (log file). Beleženje vrednosti zapisov pred ažuriranjem predstavlja zaščito pred napakami v podatkih ali programski opremi in omogoča, da se postopek večfaznega ažuriranja v primeru napake v uporabniškem programu ali kot posledica mrtve zanke v celoti anulira. Beleženje vseh ažuriranj pa omogoča, da na osnovi kopije podatkovne baze izvršimo restavriranje podatkovne baze v primeru fizičnega uničenja podatkov.

Prvi tip dnevnika (za razveljavljanje sprememb v podatkovni bazi) uporablja sistem za upravljanje podatkovnih baz avtomatično, drugi tip dnevnika (za obnovo podatkovne baze na osnovi kopije) pa uporablja restavracijski program, ki ga sproži upravitelj podatkovne baze.

Sistem IDA-Baza ne podpira namerno uvedene redundantnosti podatkov v smislu avtomatskega ažuriranja vseh obstoječih kopij istega podatka.

3.3 Pripomočki za upravljanje podatkovne baze

Postopek generiranja in spreminjanja konceptualne sheme poteka interaktivno. Komunikacija med sistemom in uporabnikom je realizirana preko ločenih menujev za imenovanje sheme in določitev gesla, za definiranje zapisov in za definiranje setov. Sistem izvaja tudi kontrolo nad posameznimi specifikacijami in preverja njihovo konsistentnost ter je za upravitelja podatkovne baze zelo priročno orodje.

Podobno velja tudi za postopek generiranja in spreminjanja interne sheme, ki pa je omejen predvsem na definiranje vsebovalnikov in vhodno izhodnih vmesnikov.

S sredstvi, ki so mu na voljo v okviru modifikacij interne sheme, ima upravitelj podatkovne baze možnost optimizirati delovanje sistema IDA-Baza na naslednje načine:

- z različnimi načini prirejanja vmesnikov posameznim tipom zapisov;
- z določanjem števila kopij posameznih vhodno izhodnih vmesnikov;
- s prirejanjem optimalne dolžine logičnih blokov glede na dolžino zapisov in pristopne metode;
- s prerazporejanjem vsebovalnikov po različnih diskovnih enotah.

Žal pa upravitelj podatkovne baze nima na voljo trdnih izhodišč za optimizacijo niti možnosti dovolj natanko preveriti uspešnost izvedenih modifikacij, ker ne razpolaga s podatki o pristopnih časih do podatkov za posamezne aplikacijske programe.

Kot poseben primer uglaševanja podatkovne baze navajamo tudi migracijo podatkov, vendar v okviru IDA-Baze te možnosti nimamo na razpolago.

Za izvajanje nadzora nad delovanjem sistema je upravitelju podatkovne baze na voljo poseben kontrolni program, ki omogoča določevanje načina vodenja dnevnikov med delovanjem sistema, ustavitev sistema in pregled nad delujočimi aplikacijskimi programi, njihovimi statusi, načinom uporabe podatkov in zahtevki po podatkih, ki čakajo na servisiranje.

3.4 Pripomočki za uporabljanje podatkovne baze s strani končnega uporabnika

Med pripomočki za uporabljanje podatkovne baze s strani končnega uporabnika zaenkrat najbolj pogrešamo jezik za povpraševanje, ki pa je v zaključni fazi razvoja.

Vmesnik na visokonivojske gosteče jezike je realiziran kot zbirka klicev podprogramov, ki so sestavni del sistema za upravljanje podatkovnih baz, tako da o pravem jeziku za manipulacijo podatkov, ki bi bil vgrajen v gosteči jezik, ne moremo govoriti. Glavne pomanjkljivosti take realizacije "jezika" so:

- obsežen opis podatkovnih struktur in raznih konstant, ki nastopajo kot parametri v klicih podprogramov;
- kompliciran in s programerskega stališča redundantno opisan klic podprogramov;
- razmeroma zapleteno in zamudno pisanje aplikacijskih programov.

Te pomanjkljivosti do neke mere odpravlja programski pripomoček IDA-Cogen, ki skrbi tako za avtomatsko vključevanje zunanjih shem v uporabniške programe kot za generiranje skeleta programov, vendar je njegova uporabnost omejena le na cobolske programe. Z njegovo pomočjo lahko kreiramo praktično celoten aplikacijski program za ažuriranje ali vnašanje zapisov določenega tipa v podatkovno bazo. Tako generiran program je pripravljen za interaktivno delo preko zaslonskega terminala in obsega tudi rutine za prikazovanje vsebine zapisov oziroma za njihovo zajemanje s pomočjo zaslonskih slik.

Naslednji programski pripomoček iz družine IDA, ki olajša pisanje programov, je IDA-Ekran. Namenjen je za avtomatsko generiranje zaslonskih slik in obsega:

- urejevalnik CED, ki je namenjen kreiranju in modificiranju zaslonskih slik;
- podprogram CRT, ki skrbi za komunikacijo med zaslonskimi slikami in uporabniškimi programi.

IDA-Ekran je zasnovan za uporabo iz programov, ki so napisani v visokonivojskem

sostečem jeziku, slike, ki jih podpira, pa so alfanumerične.

Podatke, ki jih v zvezi z opisom slike generira IDA-Ekran, uporablja modul IDA-Cosen pri generiranju ustreznih programov, predvidena pa je tudi vključitev teh podatkov v podatkovni katalog, kar pomeni, da postane slika potem dostopna vsem uporabnikom podatkovne baze.

Zaradi aktualnosti zaslonskih slik in priročnosti njihovega generiranja lahko ocenimo programski pripomoček IDA-Ekran kot zelo koristen dodatek k sistemu za upravljanje podatkovnih baz.

4. PRIMERJAVA Z OZADJEM ULTRA

Primerjava sistemov za upravljanje podatkovnih baz IDA-Baza in Ultra pokaže, da sta si oba sistema po svoji zasnovi v veliki meri sorodna. Ta sorodnost je posledica skupnih izhodišč, saj tako IDA-Baza kot Ultra v svojem bistvu izhajata iz sistema za upravljanje podatkovnih baz Total. Razlike med obema sistemoma pa so posledica različnih poti, ki so jih njuni razvijalci ubrali z namenom, da odpravijo nekatere pomanjkljivosti Totala.

Tako je za IDA-Bazo značilna večja doslednost pri izpolnjevanju specifikacij, ki jih je za mrežni podatkovni model predpisal CODASYL. To se odraža v:

- uporabi koncepta tip seta,
- uporabi konceptov sheme in podsheme,
- uporabi koncepta interne sheme,

česar pri Totalu ne zasledimo.

V nasprotju s tem pa so pri Ultri ohranili Totalov mrežni podatkovni model in sa niso skušali prilagoditi CODASYL-ovim specifikacijam. Osredotočili so se raje na oblikovanje koncepta "logični uporabniški posled", ki omogoča uporabniku, da obdeluje podatke v dvodimenzionalni tabelarični obliki, ne da bi poznal njihovo dejansko strukturo in potrebno navigacijo.

Glede na to, da relacijski sistemi za upravljanje podatkovnih baz, ki so danes na voljo, po svojih performansah še vedno zaostajajo za mrežnimi, predstavlja uvedba logičnega uporabniškega posleda vmesno rešitev, ki zagotavlja:

- enostavnost pri uporabi podatkov (zaradi njihove tabelarične predstavitve napram uporabniku),
- ustrezne performanse (ker dejanska organizacija podatkov temelji na mrežnem podatkovnem modelu),
- visoko stopnjo logične in fizične podatkovne neodvisnosti (ker je logični uporabniški posled definiran kot poseben tip logičnega zapisa, ne glede na to, kje se dejansko nahajajo tipi podatkovnih elementov, ki sa sestavljajo).

Posledica uvedbe logičnega uporabniškega posleda je tudi enostavnejši jezik za manipulacijo s podatki. Vse operacije nad logičnimi zapisi, ki jih program obdeluje, je moč izvršiti samo s štirimi osnovnimi ukazi: GET, INSERT, UPDATE in DELETE. Ti ukazi (kot tudi ostali ukazi jezika za manipulacijo s podatki) so realizirani kot samostojni stavki v okviru gostečega programskega jezika (velja za COBOL in FORTRAN), poseben predprocesor pa jih nadomesti s klici ustreznih rutin sistema za upravljanje podatkovne baze.

Ugotovimo lahko torej, da je bil pri razvoju Ultre velik poudarek namenjen enostavnejši uporabi sistema, prilagajanju uporabniku in večanju produktivnosti pri programiranju.

Seveda tudi pri razvoju programskih pripomočkov iz družine IDA ta področja niso bila zanemarjena, saj je bilo ob osnovnem modulu IDA-Baza razvitih še več orodij, katerih namen je povečati produktivnost pri programiranju. Tako IDA-Cosen (generator cobolskih programov) v precejšnji meri odpravlja pomanjkljivosti zaradi neustrezne realizacije jezika za manipulacijo podatkov pri IDA-Bazi, IDA-Ekran pa je koristen pripomoček za generiranje zaslonskih slik.

Razlika v primerjavi z Ultro je predvsem v tem, da ta orodja še vedno ostajajo zgolj v okvirih tradicionalnega mrežnega podatkovnega modela in uporabe standardnih, postopkovnih programskih jezikov 3. generacije. Prav tu pa so še neizkoriščene možnosti za nadaljnji razvoj programskih orodij IDA.

5. PREDLOGI ZA NADALJNJI RAZVOJ IN IZBOLJŠAVE

Predlogi za nadaljni razvoj in izboljšave so nanizani v takem vrstnem redu, kot si sledi tematika v predhodnih razdelkih.

Razen splošnih slabosti mrežnega podatkovnega modela, ki se jim z njegovo izbiro pač ni dalo izogniti, je v okviru podatkovnega modela moteče naslednje:

Prvi problem so ključi setov, drugi (delno iz njih izvirajoč) pa tip članstva. Ključi setov so sicer preprosto orodje za včlanjevanje zapisov - članov v sete, po logični plati pa so redundantni, saj je pripadnost določenemu setu prikazana tudi s kazalci. Redundantni so tudi v fizičnem posledu, saj se ena in ista vrednost podatkovnega elementa pojavlja pri lastniku in nato še pri vseh članih seta. Članstvo v setu je zato lahko le obvezno, seti pa ne podpirajo poljubnih povezav med člani in lastniki, ki bi se jih dalo realizirati z opcijskim članstvom. Ne najboljša rešitev je tudi to, da so v zunanji shemi večkratno predstavljeni tipi zapisov, ki nastopajo kot člani v večih tipih setov, če so tudi ti zajeti v zunanji shemi.

Na uporabniškem nivoju je možno uporabljati le mrežni podatkovni model, ki programerjem aplikacijskih programov sicer povsem zadošča, za neprogramerje pa bi bilo ugodno dodati (morda v okviru jezika za poizvedovanje) tudi relacijski podatkovni model, ki bi omogočal interaktivno uporabo podatkovne baze brez predhodnega pisanja ustreznih programov. Prvi korak k možnosti njegove uvedbe je bil storjen že z uvedbo ključev setov.

Kontrolni program, ki daje upravitelju podatkovne baze vpogled v dosajanje pri njeni uporabi, bi bilo koristno dopolniti tako, da bi posredoval več informacij o ozkih grlih pri dostopu do podatkovne baze in s tem nudil upravitelju boljšo osnovo za izvajanje optimizacije.

Posebno skrb bi bilo potrebno posvetiti jeziku za manipulacijo s podatki, ki bi moral biti bolj vključen v gostečí jezik. Eksplicitne klice podprogramov sistema za upravljanje podatkovnih baz bi bilo koristno zamenjati z ukazi, ki bi bili podobni ukazom gostečega jezika, in njihovo prevajanje ali vključiti v ustrezni prevajalnik ali pa uporabiti kak predprocesor.

Na osnovi podanih ocen programskih orodij IDA lahko sklenemo, da predstavljajo koristen pripomoček pri izdelavi aplikacijskih programov, pri čemer pa modul IDA-Cosen podpira le cobolske programe. Z ozirom na to, da postaja pomen podatkovnih baz vse večji in da se njihova uporaba širi na vsa področja človekovega delovanja, pa bi veljalo razmisliti o razvoju podobnih modulov tudi za druge visokonivojske jezike.

PODSKUP–ZAVISNOSTI U RELACIONIM BAZAMA PODATAKA I MEHANIČKO DOKAZIVANJE TEOREMA

INFORMATICA 3/86

Maleković Mirko
CVVTŠ „General arm. Ivan Gošnjak", Zagreb

UDK: 681.3.01:519

Podskup-zavisnosti su generalizacija ugrađenih višeznačnih zavisnosti. U ovom radu dokazujemo točnost formalnog sistema za podklasu podskup-zavisnosti. Dokazi se baziraju na reprezentiranju podskup-zavisnosti pomoću standardnih formi i primjeni rezolucijskih procedura dokazivanja.

SUBSET DEPENDENCIES IN RELATIONAL DATABASES AND MECHANICAL THEOREM PROVING: Subset dependencies are a generalization of embedded multivalued dependencies. In this work we have proposed a method for soundness-proving of formal system for a subclass of subset dependencies. The method is based on representation of subset dependencies by standard forms and on application of resolution proof - procedures.

U dizajniranju relacione baze podataka upotrebljavamo specijalne tipove uvjeta, tzv. zavisnosti, kao semantičko sredstvo za izražavanje svojstava podataka. Najčešće upotrebljavane zavisnosti su funkcionalne i višeznačne zavisnosti [1],[2],[3],[5],[6],[7],[8], [9],[10]. Dekompozicija relacione šeme u četvrtu normalnu formu vodi na razmatranje ugrađenih višeznačnih zavisnosti čijom generalizacijom dolazimo do podskup-zavisnosti. U ovom radu dokazujemo točnost formalnog sistema za podklasu podskup-zavisnosti. Navedeni formalni sistem je predložen u [9]. Dokazi se baziraju na primjeni procedura dokazivanja, procedure bazirane na rezolucijskom principu, koje su razvijene u teoriji mehaničkog dokazivanja teorema.
Organizacija članka je kao što slijedi; Osim uvodnog dijela i zaključka, rad ima tri sekcije. U prvoj sekciji uvodimo bazične pojmove, vezane za relacioni model, koje ćemo upotrebljavati u preostalim sekcijama. Pretpostavlja se familijarnost sa teorijom baza podataka na nivou [10]. U sekciji 2. prezentiramo podskup-zavisnosti pomoću Skolemove standardne forme. Primjena procedura dokazivanja razvijenih u teoriji mehaničkog dokazivanja teorema dana je u sekciji 3., gdje se dokazuje točnost formalnog sistema za podklasu podskup-zavisnosti.
Za razumijevanje sekcija 2. i 3. pretpostavljamo poznavanje rezolucijskih procedura dokazivanja na nivou [4].

1. Bazični pojmovi

U ovoj sekciji uvodimo bazične pojmove, vezane za relacioni model, koje ćemo upotrebljavati u članku.
Neka je U_∞ beskonačan skup apstraktnih elemenata. Elemente skupa U_∞ zovemo atributi. Neka je također $\mathcal{F} = \{D_A \mid A \in U_\infty\}$ familija nepraznih skupova, gdje je D_A zovemo domenom atributa A.

Definicija Neka je $R \subseteq U_\infty$ konačan, neprazan skup, $D = \bigcup_{A \in R} D_A$. Tipl t je funkcija $t: R \to D$ sa svojstvom $t(A) \in D_A \; \forall A \in R$.
Sa Tipl(R) označavamo skup svih tiplova nad R.

Definicija Relacija je uređen par (R,r), gdje je $r \subseteq Tipl(R)$ konačan skup. R zovemo relacionom šemom, a r primjerom relacione šeme R.

U slučaju da se relaciona šema R podrazumijeva, relaciju (R,r) ćemo kraće označavati sa r. Uređenje relacione šeme R omogućuje predstavljanje relacije r pomoću tabele; redovi tabele su tiplovi (elementi od r), a stupci su imenovani atributima iz R.

Definicija Familiju skupova $\varrho=\{R_1,..,R_k\}$, gdje su $R_1,..,R_k\subseteq R$ i $\bigcup_{i=1}^{k} R_i=R$, zovemo šemom baze podataka relacione šeme R.

Uobičajeno je da se, u navedenom slučaju, R zove univerzalnom relacionom šemom, a svaki član iz ϱ relacionom šemom. Postupak transformacije relacione šeme R u šemu baze podataka ϱ jeste dekompozicija relacione šeme R.
Da bismo opisali ugrađene višeznačne zavisnosti, trebamo karakterizirati projekciju relacije.

Definicija Neka je $S\subseteq R$ neprazan skup, (R,r) relacija. Projekcija od (R,r) na S, u oznaci $\Pi_S(r)$, je relacija (S,r_1), gdje je $r_1=\{s\in Tipl(S)\mid \exists t\in r:\ t[S]=s\}$. Ovdje je $t[S]$ restrikcija tipla t na skup atributa S

Slijedeći uobičajenu notaciju u teoriji baza podataka, jednočlan skup $\{A\}$ pišemo kao A, uniju skupova atributa $X\cup Y$ kao XY, a komplement skupa X u odnosu na skup R, $R\setminus X$, kao $\overline{X}$.
Sada definiramo pojam ugrađene višeznačne zavisnosti, te pojam podskup-zavisnosti.

Definicija Neka su $X,Y,Z\subseteq R$, $X\cap Z=\emptyset$, $Y\cap Z=\emptyset$ Izraz $X\twoheadrightarrow Y\mid Z$ zovemo ugrađena višeznačna zavisnost.

Definicija Kažemo da $X\twoheadrightarrow Y\mid Z$ vrijedi u relaciji (R,r) ako i samo ako $X\twoheadrightarrow Y$ vrijedi u $\Pi_{XYZ}(r)$.

Uočimo, da ako $X\twoheadrightarrow Y$ vrijedi u r da onda i $X\twoheadrightarrow Y\mid Z$ vrijedi u r. Obrnuto nije istina, pa moramo biti oprezni u iskazivanju uvjeta za r, a da ne ignoriramo ugrađene višeznačne zavisnosti.

Definicija Neka su $X,Y,Z\subseteq R$. Izraz $Z(X)\subset Z(Y)$ zovemo podskup-zavisnost.

Definicija Kažemo da $Z(X)\subset Z(Y)$ vrijedi u (R,r) ako i samo ako
$\forall t_1,t_2\in r(t_1[X]=t_2[X]\Rightarrow\exists t_3\in r(t_1[Y]=t_3[Y]\wedge t_2[Z]=t_3[Z]))$

Poznato je, da vrijedi
$X\twoheadrightarrow Y\mid Z\Leftrightarrow(Z(X)\subset Z(XY))\wedge Z(XY)\subset Z(X)$ tj. za svaku relaciju r koja je primjer relacione šeme R vrijedi: $X\twoheadrightarrow Y\mid Z$ vrijedi u r ako i samo ako $(Z(X)\subset Z(XY))\wedge(Z(XY)\subset Z(X))$ vrijedi u r.

Definicija Neka je r primjer relacione šeme R, a p_z neka je podskup-zavisnost nad R. Kažemo da je r model za p_z ako i samo ako p_z vrijedi u r. Dalje, kažemo da p_z vrijedi u relacionoj šemi R ako i samo ako je svaki primjer od R model od p_z.

2. Reprezentacija podskup-zavisnosti pomoću Skolemove standardne forme

U prethodnoj sekciji smo uveli podskup-zavisnost $Z(X)\subset Z(Y)$, gdje su $X,Y,Z\subseteq R$. Rekli smo da $Z(X)\subset Z(Y)$ vrijedi u r ako i samo ako $\forall t_1,t_2\in r(t_1[X]=t_2[X]\Rightarrow\exists t_3\in r((t_1[Y]=t_3[Y]\wedge t_2[Z]=t_3[Z]))$.
Uvodeći predikat E_T, $T\subseteq R$, gdje nam $E_T(s,t)$ znači $s[T]=t[T]$, podskup-zavisnost $Z(X)\subset Z(Y)$ zapisujemo u obliku

(1) $\forall t_1\forall t_2(E_X(t_1,t_2)\Rightarrow\exists t_3(E_Y(t_1,t_3)\wedge E_Z(t_2,t_3)))$.

Sada imamo, da $Z(X)\subset Z(Y)$ vrijedi u r ako i samo ako (1) vrijedi u r.
Izraz $\sim(Z(X)\subset Z(Y))$ poprima oblik

(2) $\exists t_1\exists t_2(E_X(t_1,t_2)\wedge\forall t_3(\sim E_Y(t_1,t_3)\vee\vee\sim E_Z(t_2,t_3)))$.

Skolemizacijom (1), odnosno (2), nalazimo standardne forme za $Z(X)\subset Z(Y)$, odnosno $\sim(Z(X)\subset Z(Y))$.

$$(3)\ Z(X)\subset Z(Y):\begin{cases}\sim E_X(t_1,t_2)\vee E_Y(t_1,f(t_1,t_2))\\ \sim E_X(t_1,t_2)\vee E_Z(t_2,f(t_1,t_2))\end{cases}$$

$$(4)\ \sim(Z(X)\subset Z(Y)):\begin{cases}E_X(a,b)\\ \sim E_Y(a,t_3)\vee\sim E_Z(b,t_3)\end{cases}$$

3. Točnost formalnog sistema za podklasu podskup-zavisnosti

U ovoj sekciji dokazujemo točnost formalnog sistema, za podklasu podskup zavisnosti, koji je predložen u [9]. Da bismo definirali točnost formalnog sistema moramo uvesti pojam logičke konzekvence.

Definicija Neka je C skup zavisnosti za relacionu šemu R, te neka je c pojedinačna zavisnost. Kažemo da je c logička konzekvenca od C, ili da C logički implicira c, u oznaci $\frac{C}{c}$, ako i samo ako svaki primjer (relacija) od R koji zadovo-

ljava C također zadovoljava i c. Kažemo dalje, da primjer r od R zadovoljava skup zavisnosti C za R ako i samo ako r zadovoljava svaku zavisnost iz C.

Formalni sistem teorije zavisnisnosti se sastoji od pravila (aksioma) koja omogućuju izvođenje novih zavisnosti iz zadanih zavisnosti.

Definicija neka je F_D formalni sistem, C i c kao u prošloj definiciji. Kažemo da C dokazuje c ako i samo ako je moguće upotrebom pravila iz F_D na zavisnosti iz C izvesti zavisnost c. Da je c dokazivo iz C u formalnom sistemu F_D označavat ćemo sa $C \vdash_{F_D} c$.
Sada imamo definiciju točnosti formalnog sistema.

Definicija Za formalni sistem F_D kažemo da je točan ako i samo ako
$\forall C \forall c ((C \vdash_{F_D} c) \Rightarrow (C \models c))$

Pređimo na uvođenje formalnog sistema za slijedeći specijalni slučaj; Neka je $Z \subseteq R$ fiksiran skup atributa, te neka je $ZSD(R) \overset{def}{=} \{Z(X) \subset Z(Y) / X, Y \subseteq R\}$. Elemente skupa $ZSD(R)$ zovemo Z podskup-zavisnosima. Formalni sistem za $ZSD(R)$ se sastoji od slijedeća dva pravila:

SD1 $X \subseteq Y \subseteq R \vdash Z(Y) \subset Z(X)$ refleksivnost
SD $Z(X) \subset Z(Y),\ Z(Y) \subset Z(W) \vdash Z(X) \subset Z(W)$.

Pravilo SD2 predstavlja tranzitivnost za Z podsku-zavisnosti.
Za dokaz točnosti navedenih pravila trebamo dokazati slijedeće teoreme:

$$\text{SD1}' \quad \frac{X \subseteq Y \subseteq R}{Z(X) \subset Z(Y)}$$

$$\text{SD2}' \quad \frac{Z(X) \subset Z(Y),\ Z(Y) \subset Z(W)}{Z(X) \subset Z(W)}$$

Dokaz je kao što slijedi:
Transformacijom $\sim(Z(Y) \subset Z(X))$ u standardnu formu dobijemo:

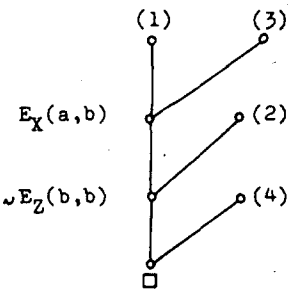
$$\sim(Z(X) \subset Z(X)): \begin{cases} E_Y(a,b) & (1) \\ \sim E_X(a,t_3) \vee \sim E_Z(b,t_3) & (2) \end{cases}$$

$$X \subseteq Y: \sim E_Y(t_1,t_2) \vee E_X(t_1,t_2) \qquad (3)$$

Rečenica (3) je dobivena restrikcijom predikata E_Y na skup X.
Navedeni skup rečenica proširujemo pravilom $\forall T \subseteq R \forall t \in r\ (E_T(t,t))$. Dobivamo rečenicu

$$E_T(t,t) \qquad (4)$$

Dokaz pravila SD1' se sada svodi na dokaz kontradiktornosti skupa rečenica $S = \{(1),(2),(3),(4)\}$. Linearnom rezolucijom imamo slijedeće stablo dokaza:



Sl. 1.

Dokaz SD2' :
Skolemizacijom $Z(X) \subset Z(Y) \wedge Z(Y) \subset Z(W) \wedge \wedge \sim(Z(X) \subset Z(W))$, dobivamo slijedeći skup rečenica:

$$Z(X) \subset Z(Y): \begin{cases} \sim E_X(t_1,t_2) \vee E_Y(t_1,f(t_1,t_2)) & (1) \\ \sim E_X(t_1,t_2) \vee E_Z(t_2,f(t_1,t_2)) & (2) \end{cases}$$

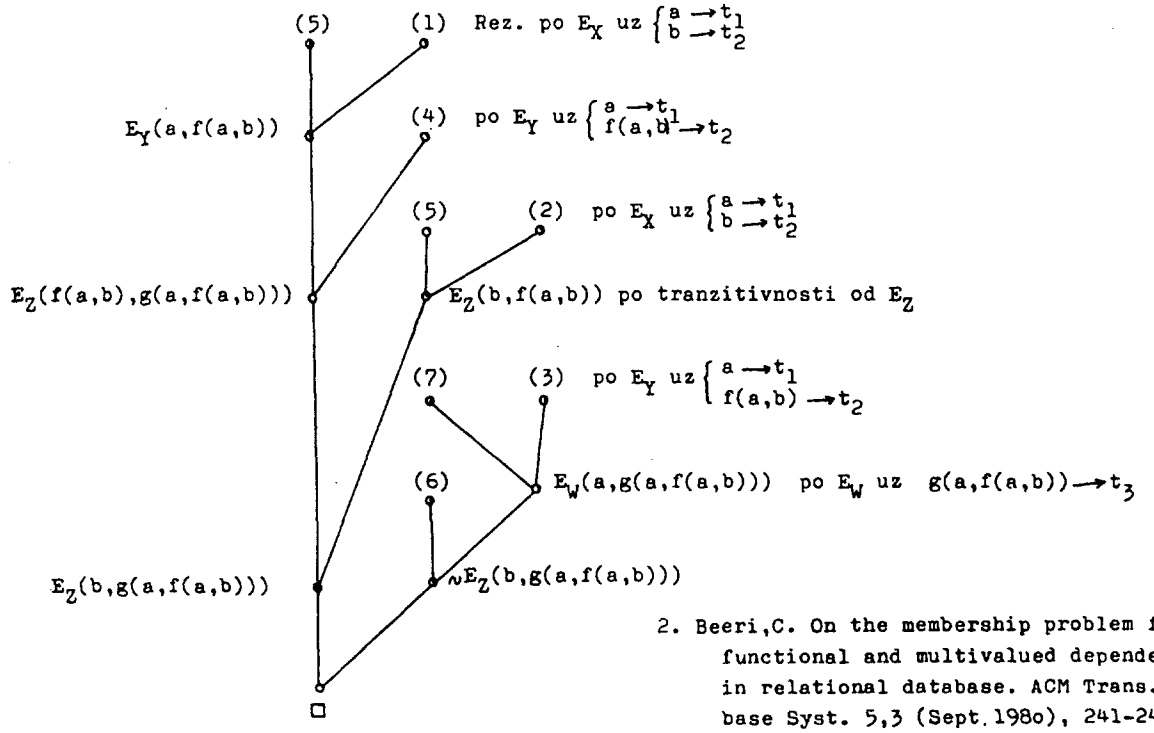
$$Z(Y) \subset Z(W): \begin{cases} \sim E_Y(t_1,t_2) \vee E_W(t_1,g(t_1,t_2)) & (3) \\ \sim E_Y(t_1,t_2) \vee E_Z(t_2,g(t_1,t_2)) & (4) \end{cases} \quad S$$

$$\sim(Z(X) \subset Z(W)): \begin{cases} E_X(a,b) & (5) \\ \sim E_W(a,t_3) \vee \sim E_Z(b,t_3) & (6) \end{cases}$$

Stablo dokaza je dano na slici 2.

Na kraju ove sekcije opišimo implikacioni problem za podskup-zavisnosti.
Neka je $SD \subseteq ZSD(R)$ i $s \in ZSD(R)$
Implikacioni problem jeste da testiramo da li je $SD \models s$*. Postupak testiranja je kao što slijedi; Skolemizacijom rečenica iz SD i rečenice $\sim s$ nalazimo skup rečenica S. Po potrebi skup S proširujemo dodatnim pravilima koja karakteriziraju predikat E_T. Dobiveno proširenje skupa S označimo sa S'. Test za $SD \models s$ se svodi na dokaz kontradiktornost skupa S'. Kontradiktornost skupa S' dokazujemo izvođenjem kontradikcije, u oznaci $\square$, primjenom rezolucijskih procedura dokazivanja. Na kraju, treba naglasiti, da su navedene procedure dokazivanja kompletne tj. ako je zaista $SD \models s$, to će biti potvrđeno izvođenjem kontradikcije. No, ako $\sim(SD \models s)$, ove procedure općenito neće terminirati (semiodlučivost rezolucijskih procedura dokazivanja).

* Ovdje $SD \models s$ označava logičku konzekvencu $\frac{S}{s}$.



Sl. 2.

4. Zaključak

U ovom radu smo tretirali podskup-zavisnosti u relacionim bazama podataka. Dokazali smo točnost formalnog sistema, za podklasu podskup-zavisnosti, predloženog u [9]. Dokazi se baziraju na procedurama dokazivanja koje su razvijene u teoriji mehaničkog dokazivanja teorema. Predloženi pristup ukazuje na mogućnost testiranja ekvivalentnosti danih skupova zavisnosti, odnosno redundantnosti danog skupa zavisnosti. Navedeno predstavlja korak u smjeru automatizacije dizajna relacionih šema, što se smatra ultimativnim ciljem teorije zavisnosti u relacionim bazama podataka. Semiodlučivost rezolucijskih procedura dokazivanja zahtjeva razvoj komplementarnog pristupa u rješavanju implikacionog problema. Teorija Armstrongovih relacija, razvijena u [6], pruža takvu mogućnost. Interakcija rez. procedura dokazivanja i procedure bazirane na teoriji Armstrongovih relacija ostaje kao otvoren problem za dalje istraživanje.

Literatura

1. Armstrong,W.W., and Delobel,C. Decompositions and functional dependencies in relations. ACM Trans. Database Syst. 5,4. (Dec. 1980), 404-430.
2. Beeri,C. On the membership problem for functional and multivalued dependencies in relational database. ACM Trans. Database Syst. 5,3 (Sept. 1980), 241-249.
3. Biskup,J. On the complementation rule for multivalued dependencies in database relations. Acta Inf. 10,1 (Jan. 1980), 93-105.
4. Chang,C.L., and Lee,C.R.T. Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York, 1973.
5. Fagin,R. Multivalued dependencies and a new normal form for relational databases ACM Trans. Database Syst. 2,3 (Sept. 1977), 262-278.
6. Fagin,R. Horn clauses and database dependencies. J.ACM 29,4 (Oct. 1982),952-985.
7. Mendelzon,A.O. On axiomatizing multivalued dependencies in relational database. J.ACM 26,1 (Jan. 1979), 37-44.
8. Sagiv,Y.,Delobel,C.,Parker,D.S., and Fagin R. An equivalence between relational database dependencies and a subclass of propositional logic. J.ACM 28,3 (July 1981), 435-453.
9. Sagiv,Y., and Walecka,S.F. Subset dependencies and a completeness result for a subclass of embedded multivalued dependencies. J.ACM 29,1 (Jan-1982), 103-117.
10. Ullman,J.D. Principles of Database Systems. Computer Science Press, Potomac, Md., 1980.

Peter Kolbezen, Branko Mihovilović
Institut „Jožef Stefan" Ljubljana

UDK: 681.519.7

V programih, pri katerih obstaja možnost sočasnega izvajanja operacij, je lahko število pomnilniških stanj preveliko, da bi bil pregled nad njimi praktično obvladljiv. Takšni primeri so pogosti pri horizontalnih mikroprogramih, ki dopuščajo večjo stopnjo paralelizma. Pričakovati je, da bo programiranje visokoparalelnih mikroprogramov po principu krmiljenja s "tokom podatkov" mnogo lažje. V takšnih primerih mora biti programer pozoren le na podatkovne odvisnosti, medtem ko je zanj eksplicitni zapis poteka kontrole nepomemben. Članek obravnava, kako se lahko nekatere tehnike, ki se sicer uporabljajo pri viskoparalelnih superračunalnikih, aplicirajo tudi na mikroprograme.

USE DATA FLOW PRINCIPLES IN MICROPROGRAMMING. In the data flow programming style the programming task for highly parallel microprograms appears to be much easyer. The programmer has to think only in terms of data dependencies, while producing his code. This paper describes how some of the techniques applied for highly parallel computers to be used for microprograms.

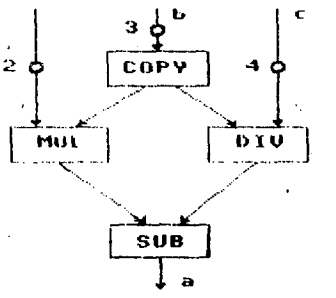
1. UVOD

Napor, ki ga je potrebno vlagati pri načrtovanju vse obsežnejših mikroprogramov sodobnih, zmogljivih računalnikov, vse bolj narašča. V primerjavi s programiranjem na nivoju zbirnika se pojavljajo dodatne težave, ki rastejo s prizadevanjem za maksimalno izkoriščenost vseh možnih virov. Največjo učinkovitost v tem pogledu je mogoče doseči z uvajanjem čim popolnejšega sočasja pri izvajanju mikrooperacij. Programer naj bi v takšnih primerih uporabljal jezik, ki bi mu omogočal dovolj učinkovito opisovanje in dober pregled nad sočasnim izvajanjem operacij načrtovanega procesa.

Obstajajo podobnosti na področju interpreterjev in paralelnih računalnikov: oba področja se ukvarjata z maksimalnim izkoriščanjem večkratnih virov. Zato je še posebej pomembno slediti raziskovalnim dosežkom na področju superračunalnikov, saj so le ti najlažje uporabljivi tudi pri načrtovanju mikroprogramov in njih pripadajočih interpreterjih.

2. DF MIKROPROGRAMIRANJE

Nov princip krmiljenja s tokom podatkov /2/ je mogoče prilagoditi tako, da je primeren tudi za mikroprogramiranje. V primerjavi s konvencionalnim izvrševanjem programa obstaja bistvena razlika v metodi kontrole. Programi, ki so krmiljeni s tokom podatkov, imajo asinhrono kontrolo, kar pomeni, da je pogojenost izvajanja operacije lokalnega značaja. Operacija se izvrši takoj, ko so dosegljivi vsi vhodni podatki. Med izvrševanjem operacije se vhodni podatki zajemajo v vhodnem pomnilniškem vmesniku, rezultati pa se odlagajo na izhodnem pomnilniškem vmesniku. Procesno enoto navadno sestavlja zelo velika množica (tudi 100 ali več) procesorjev. Sposobna je izvrševati vse potrebne operacije ter je vključena med vhodno in izhodno podatkovno potjo, to je med obema vmesnikoma.



Slika 1. Program krmiljenja s tokom podatkov, ki izračuna izraz a = 2b-b/c

Na sliki 1 je prikazan takšen princip kontrole na asemblerskem nivoju. Glede na pravilo, da se prične operacija izvajati takoj, ko so na vhodu prisotni vsi podatki velja, da se operacija COPY izvrši s prisotnostjo vrednosti b na vhodu operatorja. Ta vrednost se prenese preko vhodne poti na procesor, ki izvrši operacijo COPY. Kot rezultat se pojavita na izhodu operatorja dve enaki vrednosti, ki izpolnjujeta pogoj, potreben za pričetek sočasnega izvajanja operacij MUL in DIV. Oba paralelna tokova sta (po istem pravilu) sinhronizirana na vhodu operatorja SUB. Ta namreč čaka na izvajanje operacije toliko časa, dokler nista dosegljiva oba podatka: iz operatorja MUL in iz operatorja DIV.

Programi, ki so krmiljeni s tokom podatkov, kažejo le odvisnost med podatki. Tako je v našem primeru operacija SUB odvisna le od rezultatov operacije MUL in DIV. Kontrola operacij ni oddvojena od podatkov. Ti se ne shranjujejo na neke določene lokacije, ampak le začasno v pomnilniške vmesnike med operatorji. Pri novem načinu se ne določa mehanizma krmilnega toka, niti ne shranjevanje podatkov in stanj pomnilnika. Konkurenčni programski tok je izražen in krmiljen izključno s podatkovnim pretokovnim programskim grafom (data flow program graph).

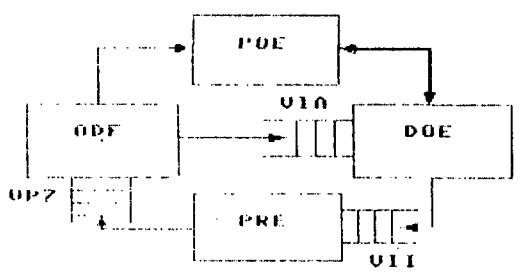
Obstaja že več jezikov za opisovanje podatkovno pretokovnih programov. Jezik EDDA /3/ je na primer grafični jezik, ki je na videz podoben grafu na sliki 1. Medtem, ko ta jezik zahteva grafični terminal, ter za razvoj programov in editeranje uporablja tehniko menijev, sta jezika Val in Id tekstualno podatkovna pretokovna jezika. Vsi ti jeziki so visokonivojski, ki v svojih različicah dosegajo tudi nivo strojne kode. Nobeden od njih pa sprva ni bil namenjen mikroprogramskemu nivoju. Veliko napora pa je bilo že vloženega v razvoj takšnega programskega jezika, ki bi bil uporabljiv in dovolj učinkovit tudi za mikroprogramerja. Lepi uspehi so že doseženi pri mikroprogramskih podatkovno pretokovnih jezikih, ki so podobni jeziku EDDA.

V konvencionalnih programih je izvršitev naslednje operacije odvisna samo od izvršenosti predhodnih operacij. Podatki se pojavljajo v obliki referenc pomnilniških lokacij (predstavljene z imeni). Programer mora zagotoviti, da so ustrezni podatki razpoložljivi na referenčnih lokacijah vselej, ko se prične izvajati operacija, ki te podatke potrebuje. Pri običajnih programih mora biti programer seznanjen s stanjem pomnilnika v času izvajanja vsake instrukcije, ki zaseda pomnilnik. V programih, pri katerih obstaja možnost sočasnega izvajanja operacij, kot na primer v horizontalnih mikroprogramih, pa je pogosto število pomnilniških stanj preveliko, da bi bil možen dober pregled nad vsakokratnim stanjem pomnilnika.

Pričakovati je, da bo postalo načrtovanje visko paralelnih mikroprogramov z novim načinom programiranja s tokom podatkov bistveno lažje, saj mora programer pri pisanju kode misliti le na podatkovne odvisnosti.

DF programi kažejo še druge prednosti: poleg relativno lahkega programiranja so programerju paralelne operacije vidnejše brez dodatnega napora, ne obstajajo pa tudi podatkovne odvisnosti, ki bi bile skrite v kodi.

DF programi se ne morejo izvajati direktno na mikroprogramskih interpreterjih običajnih arhitektur, kot bi bilo sicer zaželjeno. Prepad med jezikom in materialno opremo je mogoče premostiti z ročnim prevajanjem DF programov v konvencionalno mikrokodo.



Legenda:
POE...pomnilniška enota s celicami M1,...,Mm
DOE...dostavna enota (Fatch Unit)
PRE...procesna enota s procesorji P1,...,Pp
ODE...odlagalna enota (Up-date Unit)
VIA...vrsta instrukcijskih adres
VII...vrsta izvršljivih instrukcij
VP2...vrsta podatkovnih znakov (data token)

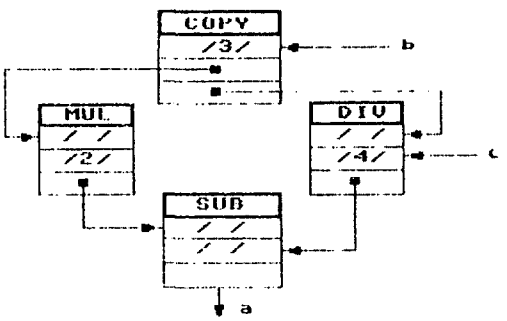
Slika 2. Paketni komunikacijski računalnik

Obstaja pa tudi druga pot do rešitve nakazanega problema: prevajalniki za mikrokodo se lahko načrtajo tako, da omogočajo direktno izvajanje DF kode. To pa je možno le, če tečejo na DF arhitekturi, ki je načrtovana po Dennisu /1/. Takšna arhotektura je prikazana na sliki 2.

Pomnilniška enota hrani DF program s štirimi instrukcijami nam že znanega programa a = 2b-b/c. (Glej sliko 3!).

Instrukcije se nizajo v okvir (frame), imenovan tudi "šablona" (template). Ta vsebuje operacijske kode in vmesniški prostor pripadajočih vhodnih podatkov, kamor se ažurno nalagajo. Med drugimi pomožnimi podatki je tudi kazalec, ki kaže na eno ali dve naslednji šabloni.

Brž, ko se na vhodu šablone COPY pojavi vrednost "b", je operacija COPY pripravljena na izvajanje. To pripravljenost bo nakazala tako, da bo poslala svojo adreso v vrsto adres instrukcij, ko instrukcija doseže glavo vrste, izbere dostavna enota enega od procesorjev procesne enote in napolni šablono izbranega procesorja. Rezultat se naloži v dva znaka (okvirčka). Oba bosta vsebovala enako vrednost "b", toda z različnimi pomožnimi adresami (MUL in DIV): Odlagalna enota sprejme ti dve vrednosti in ju vstavi v dve šabloni operatorjev MUL in DIV, ki sta s tem pripravljena na izvajanje vsak svoje instrukcije. Adresi obeh instrukcij sta poslani v vrsto adres instrukcij. Vsaki je dodeljen svoj procesor, ki pričneta instrukciji izvajati skoraj istočasno.



Slika 3. Vsebina pomnilnika pri kodu
a = 2b-b/c

Instrukcije lahko izvaja več procesorjev hkrati. Procesorji so lahko enaki, vsak od njih pa mora v takšnem primeru izvajati vse potrebne mikrooperacije. Lahko pa obstajajo tudi grupe enakih procesorjev, od katerih je vsak procesor iste grupe sposoben izvajati le mikrooperacijo, ki je karakteristična za grupo. Torej izvaja vsak procesor iste grupe le eno vrsto mikrooperacije.

Opisani tip mikrokodnega interpreterja bo DF mikrokodo z večjo ali manjšo stopnjo paralelizma izvajal "popolnoma" asinhrono.

Kadar se med procesiranjem pojavi istočasno več poti, bo interpreter avtomatično izbral takšno število procesorjev, da bo izvajanje mikroprograma kar najbolj učinkovito. Zaradi asinhronega izvajanja tega interpreterja bo avtomatično tudi cevljenje mikrooperacij, če je povzročeno z eno makrooperacijo ali, če je povzročeno s sekvenco večih identičnih (vektorskih) makrooperacij. Programer je s tem rešen napora, ki ga sicer mora vložiti za ugotavljanje časovne slike in kontrole sočasnih dogodkov.

Odprto ostaja še vprašanje, kako zahtevna je materialna oprema takšnega DF mikroprogramiranega sistema v primerjavi z interpreterji navadnega von Neumann ovega tipa.

Ugotovljene so naslednje bistvene razlike:

- Kapaciteta pomnilnika je večja, ker se pomnilniške celice (v enoti POE) med izvajanjem programa ne uporabljajo za večkratno shranjevanje.

- Sistemu je dodana posebna odlagalna enota DOE (Up-date Unit), ki skupaj z običajnim pomnilniškim adresiranjem ugotavlja veljavnost instrukcij in jih uvršča v vrsto.

- Zanj je značilno dodeljevanje procesorjev, ki se nahajajo v procesni enoti PRE. Za dodeljevanje skrbi posebna dodeljevalna enota DOE (Fetch Unit).

Če upoštevamo dejstvo, da je cena teh dodatnih funkcijskih enot porazdeljena na vse procesorje, katerih število je sicer veliko, so pa dobro izkoriščeni, je DF interpreter relativno cenen. Izkoriščenost procesorjev je dokaj različna, da pa se povečati, če teče na interpreterju več mikroprogramov hkrati. Ta možnost obstaja, če se več mikroprogramov -pomešanih med seboj- sočasno izvaja. Zagotovitev te tega pa ne zahteva od programerja kakšnega dodatnega napora.

3. ZAKLJUČEK

Glede na gornje ugotovitve je mogoče zaključiti, da kaže mikroprogramski interpreter za DF jezike s programerskega vidika naslednje lastnosti:

- Programerju ni potrebno posebej skrbeti za optimalno izkoriščanje virov. Ti se namreč določajo in optimalno izkoriščajo povsem avtomatično.

- Za programerja so pomembne le podatkovne odvisnosti, medtem ko je zanj eksplicitni potek kontrole programa nepomemben.

- Programerju ni potrebno upoštevati programska stanja, niti upravljati s pomnilnikom.

Glede na zgoraj opisane lastnosti je dovolj očigledno, da je obravnavana arhitektura za interpretacijo mikrokode ugodna povsod tam, kjer je ob prisotnosti možnega paralelizma v obsežnejših mikroprogramih pomembna velika učinkovitost izvajanja mikroprogramov in s tem kar najboljši izkoristek stroja.

4. LITERATURA

/1/ Dennis, J. B., "Data flow supercomputers", IEEE Computer, vol. 13, no. 11, pp. 44-56, Nov. 1980

/2/ Ackerman, W. B., "Data Flow Language", IEEE-Computer, vol. 15, no. 2, Feb. 1982

/3/ Trattnig, W., Kerner, H., "EDDA, a very high level programming and specification language in the style of SADT", IEEE-CS's Proc. of the 4th Int. Computer Software & Applications Conf., pp. 436-443, Oct. 1980

NOVE RAČUNALNIŠKE GENERACIJE

O JAPONSKEM PROJEKTU RAČUNALNIKOV NOVE GENERACIJE

Ljubo Jurak

Projekt razvoja računalnikov nove generacije se je pričel na Japonskem leta 1979. Na predlog ministerstva za mednarodno trgovino in industrijo (MITI) v Tokiu je "Japan Information Procesing Development Center" osnoval komite za študij in raziskave računalnikov nove generacije. Na osnovi dvoletnih raziskav je MITI pripravilo nacionalni projekt za razvoj računalnikov nove generacije ("Pete generacije računalniških sistemov") in ga 14. aprila 1982 uradno objavilo.

Desetletni projekt je razdeljen na tri faze. V prvi fazi je predviden razvoj prototipnega osebnega računalnika za programiranje v jeziku, podobnem Prologu. Prolog je programirni jezik, posebej primeren za opis postopkov pri umetni inteligenci : učenju, logičnem sklepanju itd. Tak računalnik naj bi po hitrosti sklepanja nekajkrat presegel sposobnosti najboljših današnjih sistemov, ki razumejo Prolog. Po obsegu znanja pa bodo na ravni današnjih ekspertnih sistemov. Ta faza naj bi trajala 3 leta.

V drugi fazi trajajoči 4 leta, se bodo raziskave usmerile k večjim problemom : nadaljnemu razvoju prototipnih računalnikov in povezave množice računalnikov v sočasno in vzporedno delujoč sistem. V tem delu je predvideno tudi nadaljevanje vseh spremljajočih raziskav iz prve faze, z uporabo programske opreme, razvite v prvem delu.

V zadnji fazi bodo vso razvito opremo - programsko in aparaturno povezali v enoten sistem. Končni cilj je stroj za logično sklepanje, ki bo nekajtisočkrat hitrejši od današnjih. Uporabiti ga bo mogoče v zelo različne namene.

Časovni plan

81 - 84 začetna faza	85 - 88 osrednja faza	89 - 91 zaključna faza
Razvoj osnovne računalniške tehnologije in planiranje	Razvoj podsistemov	Razvoj celotnega sistema

Zato, da bi dosegli te cilje, naj bi po japonskem načrtu arhitekture nove generacije računalnikov učinkovito podpirale logični sklep kot osnovni korak in vzporedno procesiranje. Na tem temeljijo ostale osnovne funkcije v sistemih nove generacije, to so baze znanja, mehanizmi sklepanja in relacijske baze podatkov. Japonci menijo, da je od sedanjih programirnih jezikov za te namene Prolog najprimernejši. Osnovni del Prologa je zato izbran za strojni jezik nove generacije računalnikov.

1. O Prologu

Prva implementacija Prologa je bila narejena leta 1972 na univerzi v Marseillu. Prvi implementaciji so sledile nove. Največji napredek je pokazala implementacija Prologa na računalniku DEC-10, ki vključuje poleg interpreterja še prevajalnik. DEC-10 Prolog je tako postal nekakšen neuradni standard za Prolog. V osemdesetih letih so se vrstile nadaljne implementacije. Najpomembnejša je Quintus Prolog, ki je izboljšana verzija DEC-10 Prologa. Prolog je bil izbran kot osnova za strojni jezik računalnikov nove generacije.

Prolog je enostaven in učinkovit programirni jezik, osnovan na simbolični logiki. Služi za logično programiranje. Podobno kot LISP je Prolog interaktiven jezik, v osnovi razvit za simbolično procesiranje podatkov. V splošnem je program v Prologu zbirka procedur, vsaka procedura pa je sestavljena iz klavzul.

Prolog lahko opazujemo iz dveh aspektov. Proceduralni ali operativni je bolj konvencionalen način in opisuje zaporedje stanj med izvajanjem. Deklarativni, kjer program opazujemo kot zaporedje stavkov, ki opisujejo problem. V splošnem so v Prologu vsi objekti imenovani termi. Neformalno si lahko predstavljamo terme kot okrajšave naravnega jezika.

Deklarativen aspekt Prologa omogoča učinkovito, jasno in hitro programiranje. Program lahko razbijemo v majhne, neodvisne enote. V Prologu programiramo tako , da :

-- deklariramo dejstva o objektih in odnose med njimi

-- definiramo pravila o objektih in odnose med njimi

-- sprašujemo o objektih in odnosih med njimi

Tipična področja za programiranje v Prologu so simbolično procesiranje , obravnavanje objektov in relacij med njimi.

2. Prva faza japonskega projekta razvoja računalnikov nove generacije

Osnovni koncept projekta je rekonstruirati aparaturno in programsko opremo računalnikov na osnovi logičnega sistema, imenovanega predikatna logika. Z drugimi besedami: računalnik naslednje generacije naj bi bil stroj za predikatno logiko. Lahko bi ga imenovali stroj za logično sklepanje, ker je logični sklep osnovna operacija predikatne logike.

Današnji računalniki uporabljajo strojni jezik. Strojni jezik določa arhitekturo računalnika. Programska oprema je zgrajena na osnovi strojnega jezika. Značilnosti von Neumanovega tipa strojev so intenzivno predstavljene s strojnim jezikom.

V konceptu nove generacije računalnikov je

ekvivalent strojnemu jeziku "kernel language" (KL). KL je osnovan na predikatni logiki in predstavlja nov strojni jezik. Nov strojni jezik zahteva novo aparaturno in programsko opremo. Aparaturna oprema se v osnovi projektira za vzporedno procesiranje ali asociativno iskanje. Osnovna operacija bo logični sklep. Današnji tipi von Neumanovih arhitektur so grajeni za sekvenčno procesiranje in iskanje po naslovih. Torej bo vzporedni računalnik z asociativnim sklepanjem neka vrsta ne-von Neumanove arhitekture.

Programska oprema se gradi s kombiniranjem osnovnih funkcij sklepanja, ki jih bo izvajal aparaturni del. "Kernel language" KL sodi kot neproceduralni jezik med visoko razvite programske jezike. Na osnovi KL se razvijajo uporabniško usmerjeni programirni jeziki. Ti jeziki bodo omogočali procesiranje baz znanj in razumevanje naravnih jezikov.

Zgradili so sekvenčne stroje za sklepanje, ki služijo za orodja pri nadaljnem razvoju. Eden takšnih strojev je "Personal Inference Machine" (PSI). "Kernel language 0" KL0 je strojni jezik sekvenčnih strojev za sklepanje. PSI stroj uporablja KL0. V osnovi je KL0 razširjena verzija Prologa. KL0 je strojni jezik, Prolog pa je uporabniški jezik. Ker je programiranje v strojnem jeziku neudobno, je razvit programirni jezik "Extended Serial prolog" (ESP). ESP je prijeten za uporabo in vključuje modularne ter makro funkcije. ESP bi lahko imenovali makro-zbirni jezik za KL0. Operacijski sistem na PSI se imenuje SIMPOS. Vsi moduli operacijskega sistema SIMPOS so napisani v jeziku ESP. Kritiki jezikov predikatne logike postavljajo vprašanje, ali je mogoče te jezike uporabiti za pisanje kontrolnih programov, kot so operacijski sistemi. ESP se je na tem področju zelo obnesel in z njim napisan operacijski sistem je učinkovit. SIMPOS se razvija in trenutno na njem še ne tečejo večji aplikativni programi. KL0 je bil razvit na začetku projekta, zato odraža omejitve sekvenčnega procesiranja. Zato so razvili KL1, ki vključuje vzporedno procesiranje in ostale izboljšave glede na KL0. Pri razvoju KL1 je bil v veliko pomoč konkurentni Prolog, ki je razširjena verzija Prologa. Konkurentni Prolog je primeren za objektno orientirano programiranje in simuliranje vzporednih dogodkov.

Za pomoč pri razvoju so izdelali PSI sekvenčni stroj za sklepanje in Delta stroj za poslovanje z relacijskimi bazami.

==========

SERIJSKI STROJ ZA SKLEPANJE

==========

Za osnovni jezik računalnikov nove generacije je bil izbran logični programirni jezik. Specificira splošno ogrodje programske in aparaturne opreme. Za dosego cilja, računalnikov nove generacije, je potrebno orodje, nov računalniški sistem, ki nudi raziskovalcem učinkovito programsko okolje za logično programiranje. Ta novi računalniški sistem se imenuje "Serial Inference Machine" (SIM) in uporablja strojni jezik KL0. Osnovan je na logičnem programiranju in sekvenčno izvaja KL0.

Projekt SIM je podprojekt v okviru japonskega projekta računalnikov pete generacije. Projekt SIM vključuje razvoj aparaturne in programske opreme, PSI - računalnik, lokalno mrežo, programski in operacijski sistem SIMPOS.

1. Organizacija SIM sistema

Aparaturna oprema

Sistem sestavlja PSI in lokalna mreža ICOT-Net. Glavni del je zelo hiter procesorski sistem, ki je osnova za PSI. Nadalje obsega specializirano aparaturno opremo za V/I operacije (kot je obdelava slik).

Programska oprema SIM

Programska oprema SIM sistema je SIMPOS, programski in operacijski sistem. Osnovni operacijski sistem obsega funkcije za poslovanje s procesorjem, pomnilnikom in V/I enotami. Oblikuje abstrakte kot so : procesi, dinamična področja, pretok, zapletene V/I operacije (sistem mnogoterih oken), sistem datotek in sistem mreže. Programski sistem obsega programske sisteme za procesiranje sistemskega jezika, ESP, editor, očiščevalec in koordinator.

1.1 Razvoj strojnega jezika

Razvoj SIM sistema se je pričel z razvojem osnovnega jezika KL0. KL0 je osnovan na logičnem programiranju. DEC-10 Prolog iz Edinburga (1983) in Prolog II iz Marseilla (1982) nista primerna za strojni jezik ali jezik za opis sistema. Zato je bilo potrebno spremeniti kontrolne strukture, podatkovne tipe in vgrajene predikate osnovnega nivoja. Te spremembe omogočajo napisati osnovni operacijski sistem in potrebne aplikativne programe.

Enostaven "cut" operator DEC-10 Prologa ne zadošča za interpreter, očiščevalec, procesiranje napak itd. Zato je dodan večnivojski "cut" operator. Nadalje sta dodani operaciji "bind-hook" in "exception-hook", ki sta podobni "freeze" operaciji v Prologu II.

Podatkovni tipi

Potrebni so dodatni podatkovni tipi : nizi in vektorji. Ti podatkovni tipi so dodani in vgrajeni so predikati za poslovanje z njimi. Vektorji se uporabljajo za kontrolne tabele operacijskega sistema.

Vgrajeni predikati osnovnega nivoja

Dodanih je mnogo posebnih predikatov za dostop in kontrolo aparaturnih komponent, kot so: aparaturni registri za kontrolo in testiranje CPU, pomnilnika in V/I enot. Ti predikati se pogosto uporabljajo v osnovnem operacijskem sistemu, predvsem pri poslovanju s pomnilnikom in krmilniki perifernih enot. K tem predikatom so še dodani predikati za običajne aritmetične operacije s celimi in naravnimi števili. KL0 je projektiran tako, da doseže nivo DEC-10 Prologa. S tem se pridobi več prostora za uvajanje firmvera in aparaturnih mehanizmov za povečanje hitrosti izvajanja.

2. Glavne komponente SIM

2.1 Osnovna aparaturna oprema

Arhitektura PSI

(1) Arhitektura pomnilnika : vse spominske celice so dolge 40 bitov in vsebujejo 8 bitov označbe in 32 bitov za podatkovno polje.

(2) Mikroprogramirana kontrola: prevedeno interno kodo KLØ izvaja mikroprogramiran interpreter.

(3) Mehanizem strojnega sklada : glavni pomnilnik je razdeljen v 256 logično neodvisnih področij od katerih je vsako lahko uporabljeno kot sklad ali polnilno področje. Prirejanje strani se izvaja glede na trenutne zahteve.

(4) Podpora za multiple procese: aparaturna oprema podpira do 63 procesov.

(5) V/I vodilo je standardno IEEE 796 vodilo.

(6) LAN je tipa CSMA/CD podobno Ether-netu. Hitrost prenosa je 10 Mbps.

2.2 Aparaturna oprema PSI

(1) Hitrost izvajanja: 30 K LIPS (logičnih sklepov na sekundo)

(2) Mikroprogramski spomin: 64 bitov X 16 K besed

(3) Cas strojnega cikla: 200 nsec

(4) Pomnilnik: 40 bitov X 16 Mb besed

(5) Hitri vmesni pomnilnik: 40 bitov X 4 K besed X 2

(6) Tehnologija : TTL za CPU in NMOS za glavni pomnilnik

2.3 Razširjena aparaturna oprema

Hitri procesorski modul

(1) Strojni jezik: bazira na instrukcijskem setu orientiranem na sklad. Vključuje 170 vgrajenih predikatov ter podpira KLØ in ESP.

(2) Interpretacijski mehanizem: bazira na kopiranju struktur in uporablja 3 sklade.

(3) Arhitektura pomnilnika : vse spominske celice so dolge 36 bitov in vsebujejo 4 bite označbe in 32 bitov podatkovnega polja ali 7 bitov označbe in 29 bitov za podatke.

(4) Mikroprogramska kontrola : strojni jezik interpretira mikroprogramiran interpreter

(5) Poslovanje s pomnilnikom: glavni pomnilnik je razdeljen v 8 logično neodvisnih področij.

(6) V/I je projektiran za povezavo na PSI prek nadzornega procesorja.

(7) Hitrost izvajanja: 200 K LIPS.

(8) Mikroprogramski spomin: 80 bitov X 11 K besed.

(9) Strojni cikel: 100 nsec.

(10) Velikost pomnilnika: 36 bitov X 64 Mb besed.

(11) Hiter vmesni pomnilnik: 36 bitov X 8 K besed.

(12) Tehnologija: CML za CPU in NMOS za glavni pomnilnik.

2.4 Sistem programske opreme

Profil sistema SIMPOS :

(1) Osebni operacijski sistem z multiprocesno podporo.

(2) Modularna struktura je objektno orientirana.

(3) Komunikacija s strojem poteka prek podsistema oken, ki vključuje V/I z japonskimi pismenkami. Pismenke so predstavljene s 16 biti.

(4) Podsistem mreže omogoča uporabniku, da uporablja oddaljene izvore; procese in datoteke.

(5) Programirni sistem vključuje knjižnico v kateri je prevajalnik ESP in povezovalnik, interpreter in očiščevalnik, struktuiran editor itd.

SIMPOS sestavlja programski sistem (PS) in operacijski sistem (OS). OS sestavljajo : jedro, nadzornik in podsistem za V/I medije. PS sestavljajo podsistemi - eksperti. Te podsisteme nadzirajo uporabniki, potrebna pa je koordinacija med podsistemi in procesi. To opravlja podsistem koordinator. Ostali podsistemi so:

okna (OS)	očiščevalnik/interpreter (PS)
datoteke (OS)	editor (PS)
mreža (OS)	knjižnica (PS)

Določeni deli SIM se še vedno razvijajo. Prvi cilji pa so že doseženi. To je računalnik PSI, mreža ICOT-Net in programska oprema SIMPOS.

2.5 Proizvodnja PSI

Proizvodnja aparaturne opreme PSI se je pričela maja 1982 z detaljnim projektiranjem pomnilniškega sistema in V/I vodil. Podrobna specifikacija je bila izdelana konec junija 1983 in takrat se je pričela tudi proizvodnja. Za tem so opravili več modifikacij in izboljšav. Prvi PSI sistem vključno z V/I enotami je bil izdelan konec leta 1983.

Pri razvoju PSI je kot orodje služil mini računalnik PDP 11/23, ki je bil priključen na DEC 2060 prek DEC-Neta. Mini računalnik so imenovali SVP (supervisor procesor za PSI) in je služil za odkrivanje napak v firmveru in aparaturni opremi. Kasneje je bil uporabljen za odkrivanje napak v osnovnih komponentah sistema SIMPOS.

2.6 Razvoj firmvera

Najprej je bilo potrebno napisati mikro-zbirnik in zmogljiv mikro-simulator za pisanje in popravljanje mikro-programov. Splošni mikro-zbirnik bi lahko bil napisan v Prologu. Zato je Prolog osnovni del zbirniškega procesorja. Na ta način je možno opisati arhitekturo s Prolog programom. Mikro-simulator je napisan v Pascalu.

KLØ obsega več kot 100 vgrajenih predikatov. Za razvoj osnovnih delov mikro-interpreterja je bilo potrebno veliko časa. Kodiranje in

testiranje mikro-programov je bilo izvrseno na DEC 2060 z uporabo mikro-zbirnika in simulatorja. Večina mikro-programov za SIMPOS je bila izdelana marca 1984. Celotno število mikro-programskih korakov je okrog 12 K.

2.7 Aparaturna konstrukcija in implementacija PSI

PSI omogoča :

(a) Učinkovito izvajanje logičnega programirnega jezika KL0, ki je strojni jezik PSI.

(b) Arhitektura aparaturne opreme podpira SIMPOS operacijski sistem, razvit za PSI.

(c) Velikost pomnilnika in hitrost sta dovolj velika za izvajanje velikih programov. (16 Mb pomnilnika in hitrost 30 LIPS)

(d) Zelo zmogljive interaktivne V/I naprave : bit-mapiran ekran, miška itd.

(e) Lokalna mreža (LAN) za interno komunikacijo med PSI in skupna uporaba perifernih naprav.

Za učinkovito izvajanje logičnih programov ima PSI :

(a) Adaptirano aparaturno opremo (ICOT razvoj), ki povečuje hitrost izvajanja.

(b) Fleksibilen mikroprogramiran sekvenčni kontroler z velikim vpisovalnim kontrolnim spominom.

(c) Možnost evaluacije aparaturne opreme in firmvera za merjenje dinamičnih karakteristik in zbiranje statističnih podatkov.

2.8 Arhitektura PSI

Format besede

Besedo sestavlja 40 bitov, 8 bitov za "tag" označbo in 32 za podatke. Označba vsebuje 2 bita za "zbiranje smeti" in 6 bitov, ki predstavljajo tip podatka :

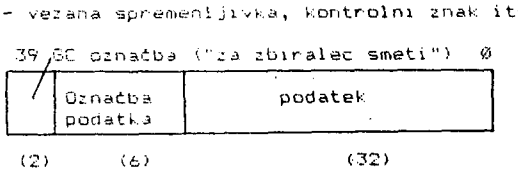
-- nedefiniran simbolični atom

-- celo ali realno število

-- sklad/polnilni vektor, niz, koda, vgrajena koda

-- lokalna/globalna spremenljivka, lok./gl. referenca

-- vezana spremenljivka, kontrolni znak itd.

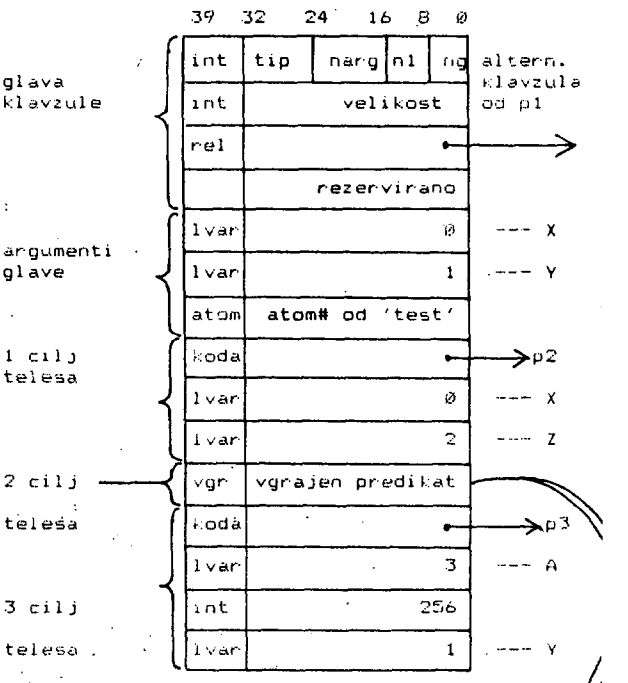


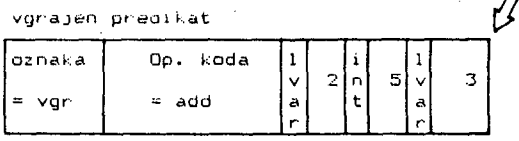
Strojni ukazi

KL0, logični programirni jezik, katerega specifikacije so skoraj enake kot DEC-10 Prolog definira funkcije strojnih ukazov PSI računalnika. Strojni ukaz, predstavljen na sliki je enostavno preveden iz KL0 izvornega programa.

Strojni ukaz izvaja firmverski interpreter, ki zmore unifikacijo in avtomatsko vračanje. Strojna instrukcija KL0 je predstavljena z glavo klavzule, argumenti glave in z nekaj cilji. Ti cilji vsebujejo uporabnikove predikate in vgrajene predikate. Večina vgrajenih predikatov ima kompakten format, ki vsebuje v eni besedi kodo operacije in največ 3 operande.

```
p1(X,Y,test)      :- p2(X,Z),      add(Z,5,A),
p3(A,256,Y)
```





Slika : predstavitev strojnega ukaza.

Vsak od argumentov ima 3 bite za oznako in 5 bitov za podatek. Ko se vgrajen predikat izvaja, se odvisno od kode operacije direktno pokliče ustrezen firmverski program. Če nastopajo kot argumenti cela ali realna števila, so dovolj majhna, da jih predstavimo s 5 biti. Večino vgrajenih predikatov lahko shranimo v eno besedo. Ta predstavitev je dokaj učinkovita, saj prihrani prostor v pomnilniku in skrajša čas izvajanja.

2.9 KL0

Specifikacije KL0 so :

(a) osnovan je na podzbirki DEC-10 Prologa.

(b) ima razširjene kontrolne funkcije.

(c) ima kontrolne funkcije za strojno opremo.

Podzbirka DEC-10 Prologa je zato, ker KL0 ne vsebuje enakih vgrajenih predikatov za interno poslovanje z bazo, predikate za V/I kot so read/write. Te predikate nadomeščajo uporabniški predikati, napisani z uporabo predikatov za kontrolo funkcij strojne opreme. Kontrolne funkcije strojne opreme ustrezajo direktnim strojnim operacijam za poslovanje z registri, pomnilnikom in V/I vodilom.

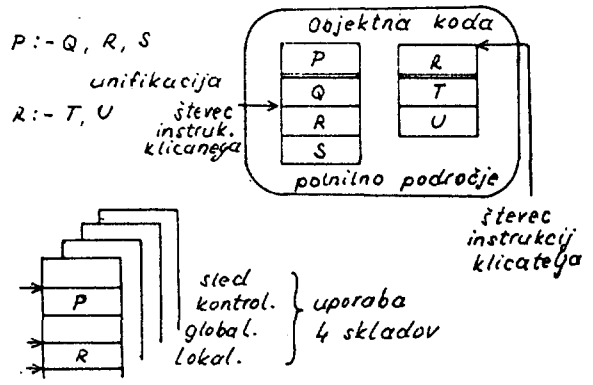
Vsi sistemski in uporabniški programi so napisani v logičnem programirnem jeziku ESP, ki je jezik za opis sistema in uporabniški jezik PSI računalnika. Pred izvajanjem je potrebno prevesti programe v KL0.

2.10 Izvajalno okolje za KL0

Za izvajanje KL0 programov uporablja interpreter 4 sklade:

-- lokalni sklad
-- globalni sklad
-- kontrolni sklad
-- sklad sledi
-- in polnilno področje.

Polnilno področje se uporablja za shranjevanje strojnih ukazov in vektorjev. Vzdrževanje skladov in kontrola izvajanja sta enaka kot pri DEC-10 Prologu. Edino lokalni sklad DEC-10 Prologa je na PSI razdeljen na kontrolni in lokalni sklad, ker je potreben neodvisen kontrolni okvir za razširjene kontrolne strukture.

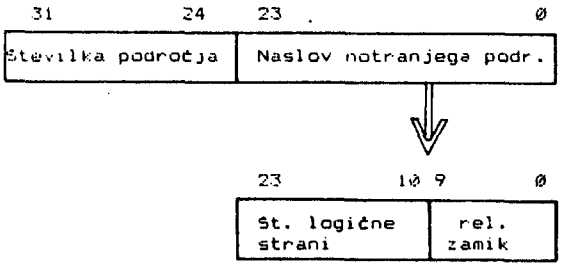


Na sliki je prikazano izvajalno okolje KL0 med izvajanjem unifikacije. Strojne instrukcije za klavzule in kazalci se nahajajo na polnilnem področju. Kakor pri DEC-10 Prologu, se sestavi skupina celic v okvir, glede na klicatelja in klicanega. Ti okviri se shranijo v lokalni sklad za spremenljivke in v globalni sklad za spremenljivke v strukturiranih podatkih. Da dosežemo določeno celico v okvirju, uporabimo relativni zamik od baze okvirja. Kontrolni sklad služi za shranjevanje okvirjev, ki vsebujejo informacijo pri vračanju po verigi, kakor tudi kazalce okolja za nadaljevanje s povratne točke.

Sklad sledi se uporablja za shranjevanje naslovov celic, ki jih moramo sestaviti pri vračanju. Dostop v sklad se vrši na osnovi kazalca začetka sklada. Ti kazalci : kazalci na ukaze, kazalci na baze okvirjev in kazalci na začetek sklada sestavljajo izvajalno okolje KL0.

2.11 Predstavitev naslovov

Za izvajanje KL0 programov so potrebni 4 skladi in polnilno področje. PSI potrebuje konkurentno izvajanje več procesov in razdeljevanje instrukcijskih kod med procese. Da zadosti tem zahtevam, je naslovni prostor razdeljen v logično neodvisna področja, kjer je vsako področje predstavljeno s številko. Področje je lahko prirejeno enemu od skladov ali polnilnemu področju in skupno več procesom. Področje služi za shranjevanje kode in skupnih prostorov za spremenljivke. Zato je naslov predstavljen z 8 bitno številko področja in 24 bitnim notranjim področjem naslova (glej sliko 4.)



Področje : logični naslovni prostor max. 16 M besed.

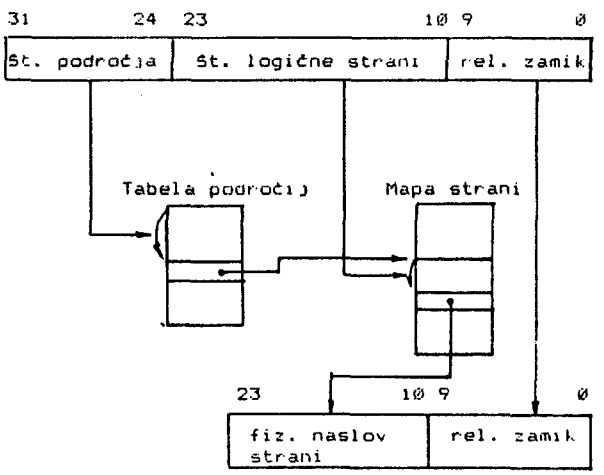
Naslovni prostor (32 bitov): sestavlja 256 področij.

Slika : predstavitev naslova

PSI ima do 256 področij, vsakemu je možno prirediti 16 M besed spomina.

2.12 Pretvarjanje naslovov

PSI ima največ 16 M besed realnega spomina. Za prirejanje in sproščanje področij v spominu je izdelan mehanizem za pretvarjanje naslovov. Fizični pomnilnik se uporablja v straneh po 1 K besed. Strani se na zahtevo prirejajo področjem, sprošča pa jih "pobiralec smeti". Slika 5 ponazarja mehanizem pretvarjanja naslovov, ki je izveden z dvema tabelama, eno za mapo baz strani in drugo za mapo strani.



Slika : mehanizem pretvarjanja naslovov.

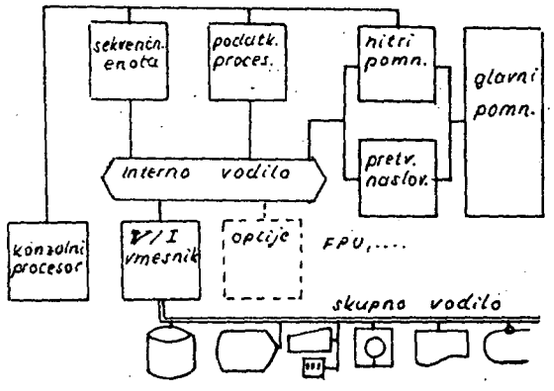
2.13 Multipli procesi

Editorji, prevajalniki in uporabniški programi, se izvajajo na PSI kot ločeni procesi. Vsak od ten procesov ima svoj status, ki vključuje KLØ izvajalno okolje in strojno kontrolno informacijo : prioriteto procesorja za procesiranje prekinitev. Okolje in informacije so zbrane v tabeli, ki se imenuje kontrolni blok procesa (PCB). PCB neaktivnega procesa je shranjen v lokalnem pomnilniku procesorja, medtem ko je PCB procesa v obdelavi porazdeljen med registre procesorja. Vsebina trenutnega PCB se spremeni, kadar se izvede preklop procesa. Preklop procesa se sproži s prekinitvijo ali z različnimi vgrajenimi predikati. Zaradi omejenega števila področij je največje število procesov 63.

2.14 Prekinitve

V PSI je adaptiran vektorski sistem prekinitev. Za vsak izvor prekinitve je pripravljen prekinitveni vektor. Vsakemu vektorju je prirejen register, ki identificira proces. Ko se zgodi prekinitev, se proces preklopi na ustrezen proces iz registra, ki ga zatem izvede firmver. Obstaja 8 nivojev prekinitev, 2 za zunanje in 6 za notranje prekinitve. PSI ima tudi nemaskiran sistem pasti za poslovanje z napakami, ki se zgodijo med izvajanjem programov. "Zbiralec smeti" (Garbage collector) se sproži kot neodvisen proces na osnovi posebne pasti. Določene prekinitve imajo višjo prioriteto od zbiralca. Te nujne prekinitve obravnavajo vzporedni procesi, ki uporabljajo nekaj posebnih področij za sklade in polnilno področje. Zbiralec smeti teh posebnih področij ne procesira.

2.15 Konfiguracija sistema



Slika: Konfiguracija sistema.

Slika predstavlja konfiguracijo PSI sistema. CPU vsebuje sekvenčno kontrolno enoto za procesiranje podatkov, spominsko enoto s hitrim vmesnim pomnilnikom in enoto za pretvarjanje naslovov, vmesnik na V/I vodilu. Medsebojno so enote povezane z notranjimi vodili. V/I sistem PSI je standardno IEEE-796 vodilo in več V/I enot.Konzolni procesor je priključen na CPU za vzdrževanje, inicializacijo in za podporo pri odkrivanju napak. Če je potrebno močnejše orodje za odkrivanje napak, je na CPU mogoče priklopiti mikroračunalnik PDP 11/23 plus.

2.17 V/I enote

-- bit-mapiran ekran (1200 X 900 točk)
-- optična miška, tastatura
-- trdi disk (37 Mb X 2)
-- gibki disk (1 Mb X 2)
-- lokalna mreža (LAN)
-- linijski tiskalnik

Priključimo lahko poljubne periferne enote, ki imajo standarden vmesnik IEEE 796. Na V/I vodilu je 512 K bytov vmesnega pomnilnika, ki se uporablja pri prenosu podatkov na sekundarne pomnilne enote ali v LAN. Ta vmesni pomnilnik uporablja programska oprema kot hitri vmesni pomnilnik pri prenosu podatkov z diskov. Bit-mapiran kontroler ima neodvisen pomnilnik za sliko in lastne rasterske funkcije. V ta pomnilnik se lahko shrani več kot 10 polnih ekranskih slik. Shranjevanje okenskih slik zelo zmanjša obremenitev V/I vodila.

ESP - RAZŠIRJEN SERIJSKI PROLOG(Extended Serial Prolog)

PSI računalnik, na katerem deluje sistem SIMPOS,ima strojni jezik KLØ, podoben Prologu. ESP prenesen v KLØ se direktno izvaja. Vse značilnosti ESP izhajajo iz KLØ. Razmerje ESP proti KLØ je podobno kot Flavors proti Lispu.ESP omogoča logično programiranje. Je objektno orientiran in vključuje makro ekspanzije.

Logično programiranje

ESP vljučuje značilnosti logičnega programiranja KLØ. Pri prenosu unificira parametre in ima vgrajen mehanizem preiskovanja AND-OR dreves najprej v globino, ter avtomatsko vračanje. Vsak Prolog program lahko skoraj neposredno prevedemo v ESP.

Objektna orientacija

Z gledišča logičnega programiranja je v ESP objekt predstavljen z zbirko aksiomov. Z uporabo zbirke aksiomov se skuša dokazati določeno predpostavko o objektu. Objekti so sestavljeni v razrede. ESP program je sestavljen iz ene ali več definicij razredov. Niz aksiomov povezanih z objektom je v osnovi unija aksiomov, definiranih v razredu in aksiomov definiranih v superrazredu tega razreda. Zraven logičnih značilnosti jezika, ima KLØ značilnosti podobne Lispu, kot so : cons, rplaca itd.

Makro ekspanzije

ESP vključuje zelo fleksibilen mehanizem makro ekspanzij. Kjer je makro poklican, se nadomesti z ekspandirano obliko. Makroje lahko kličemo tudi iz ostalih jezikov, ki imajo to možnost.Na ta način lahko vstavljamo določene cilje v klavzule. Natančna pozicija vstavka je odvisna od tega, kje se pojavi klic makroja, v glavi klavzule ali v telesu cilja. Na osnovi tega mehanizma se lahko v glavi klavzule ali telesu cilja pojavijo kot argumenti aritmetični izrazi.

1 Časovno odvisna stanja

Programi komunicirajo z objekti izven

računalnika : V/I enotami, drugimi računalniki v mreži, z uporabnikom za terminalom. Zunanji objekti imajo lahko časovno odvisna stanja, ki so zanimiva za programe. Zato mora sistem zgraditi modele časovne odvisnosti zunanjega sveta.

Naivna implementacija

V čisti logiki je časovna odvisnost predstavljena z logičnimi relacijami med časovnimi periodami in ustreznimi stanji. Take relacije so same po sebi permanentne in niso časovno odvisne. Časovno neodvisna predstavitev je logično elegantna, toda naivna implementacija, ki ni učinkovita.Razlog neučinkovitosti je dejstvo, da je dokaj težko ugotavljati tiste relacije informacij, ki programu niso več potrebne. Če uporabimo enostavno obliko baze podatkov, recimo tako kot jo uporabljajo trenutne verzije Prologa, potrebujemo ogromno pomnilnika in upočasnjujemo sistem.

Programiranje v realnem času

V klasičnih operacijskih sistemih konvencionalnih računalnikov se modelira čas zunanjega sveta direktno s časom obdelave. Časovno odvisna stanja zunanjega sveta so predstavljena s časovno odvisnimi stanji obdelave. Tak način imenujemo programiranje v realnem času. Nemogoče je restavrirati predhodno stanje, ko je določena obdelava izvršena.Če obdržimo relacije med časovnimi periodami in stanji v podatkovni bazi, lahko programiramo v realnem času. Zato je v ESP uvedena notacija časovno odvisnih stanj.

2. Objekti in razredi

Metode

Objekt je v ESP predstavljen z nizom aksiomov. To je v osnovi enak koncept, kot "svetovi" v nekaterih Prologih. Isti klic predikata ima lahko različne semantike, če ga uporabimo na različnih nizih aksiomov. Pri mehanizmu svetov se niz aksiomov določi ob izvajanju. V ESP se določi z objektom, podanim s prvim argumentom klica. Predikati s takšno semantiko, ki je odvisna od argumenta, se imenujejo metode, kakor v ostalih objektno usmerjenih jezikih.

Klice metod razlikujemo od klicev vgrajenih ali lokalnih predikatov tako, da postavimo pred klic ":" (primer ":odprt(Vrata)". V primeru ima spremenljivka "Vrata" vrednost, ki je objekt povezan z nizom aksiomov, v katerem se nahaja predikat "odprt".

Razredi

Posamezen objekt pripada razredu objektov. Razred objektov je enostavno razred, ki definira skupne lastnosti skupini podobnih objektov. Objekt, ki pripada nekemu razredu je temu razredu prirejen. Ena ali več definicij razredov sestavlja ESP program. Definicija razreda obsega različne atribute razreda, vključno z nizom aksiomov, ki so povezani z objektom prirejenim razredu. Razred se lahko obravnava kot objekt, ki predstavlja določen niz aksiomov.

Sestavi objektov

Objekt lahko ima časovno odvisne spremenljivke stanj, ki jih imenujemo sestav objekta. Sestavi niso logične spremenljivke, ampak za logično programiranje konstantne vrednosti. Vrednosti sestavov lahko odčitamo z navedbo imena po metodi "get-slot" definirani v nizu objektu ustreznih aksiomov. Vrednosti sestavov definirajo del aksiomov iz niza povezanih z objektom. Prireditve istemu razredu imajo ista imena sestavov. Vrednosti sestavov lahko spreminjamo z metodo "set-slot". To ustreza spreminjanju niza aksiomov predstavljenih z objektom. Ta značilnost je podobna ukazoma "retract" in "assert" v DEC-10 Prologu, le da je način spreminjanja pri Prologu dokaj omejen. "Assert" in "retract" je mogoče uporabiti samo pri interpretiranju programov, ne pa v prevedenih programih. ESP uporablja kratko notacijo za poslovanje s sestavi. Na primer "X!a" nam da vrednost sestava X in "X!a := V" priredi X-u vrednost V.

Mehanizem prirejanja

V ESP je več mehanizmov, ki so podobni Flavors sistemu. Definicija razreda je lahko naravna, ki definira eden ali več superrazredov. Če ima superrazred nadaljni superrazred, je ta tudi superrazred originalnega razreda. Superrazredi razredov tvorijo drevesno strukturo. Hirearhija razredov in vse prirejene relacije med razredi se določijo statično pri prevajanju, medtem ko se pri Prologu določijo ob izvajanju.

Prirejanje metod

Niz aksiomov povezan z razredom je unija aksiomov definiranih v razredu in tistih, ki so definirani v superrazredu. Nekateri superrazredi in podrazredi vsebujejo aksiome za isti predikat. Nizi aksiomov superrazredov so enostavno sestavljeni, aksiomi za isti predikat pa povezani z operatorjem OR.

Dokler se obravnava ista logika, lahko sestavimo semantično mrežo na osnovi IS-A hirearhije. Če se uporablja ista logika, zaporedje z OR povezanih aksiomov nima vpliva.Ima pa vpliv, če se obravnavajo izven računalnika. Zato ESP predvideva specifikacijo zaporedja lastnosti, kar je isto, kot zaporedje preiskovanja aksiomov, kadar imajo različni razredi aksiome za isti predikat.

Lastnosti sestavov

Razredi imajo sestave definirane v definicijah razredov in v definicijah superrazredov. Če imajo sestavi enako ime, se privzame samo en sestav s tem imenom. Za sestave objektov se lahko uporabi "PART-OF" hirearhija na osnovi IS-A hirearhije.

Predpostavimo, da je ključavnica del vrat. Najprej podamo definicijo enostavnih vrat brez ključavnice. Potem definiramo razred s_ključavnico, tako da ima prirejen sestav z razredom ključavnica. Nazadnje definiramo razred vrata_s_ključavnico, ki vsebuje oba razreda, razred vrata in razred s_ključavnico. Vrata s ključavnico so vrata in prav tako objekt s ključavnico. V našem primeru smo definirali razred s_ključavnico kot ločen razred. To je bolje, kot da bi razred vrata_s_ključavnico direktno vseboval razred vrata. Tako v ESP v celoti izkoristimo možnost večkratnega vsebovanja. Razred s_ključavnico lahko za tem uporabimo za definiranje nadaljnih razredov, na primer okno_s_ključavnico.

3. Nemonotonija

Pri mehanizmu prirejanja mora biti niz aksiomov podrazreda superniz tistih iz superrazreda. Torej mora biti predikat uspešno dokazan v superrazredu uspešno dokazan tudi v podrazredu. Tak mehanizem prirejanja je monoton. Zaradi monotonije vrata_s_ključavnico ne morejo biti podrazred od vrat, ker vrata_s_ključavnico ne

moremo odpret, če so zaklenjena, vrata pa lahko vedno odpremo. Za vzdrževanje enostavnega mehanizma prirejanja mora biti v našem primeru razred enostavnih vrat brez ključavnice podrazred vrat_s_ključavnico. Projektant hierarhije razredov mora misliti pri razredu vrata na to, da se lahko zgodi, da bo nekdo potreboval vrata_s_ključavnico. Take razširitve je zelo težavno predvideti. Kadar gradimo z uporabo monotonega znanja, je mnogo lažje razvijati programe.

Operator "Cut"

Nemonotonija je v ESP vpeljana z dvema mehanizmoma. Eden je nekakšna razširjena verzija dobro znane "cut" operacije. V KL0 vgrajen predikat poreže alternative do podanega nivoja vgnezdenih klicev predikata. Ena najpomembnejših uporab "cut" operatorja je pri metodi prekrivanja. To je zamenjava aksioma definiranega v superrazredu s tistim, ki je definiran v svojem podrazredu. "Cut" operator postavimo v definicijo aksiomovega podrazreda, ki poreže do takega nivoja, da odreže aksiom iz superrazreda. Ker se tako prekrivanje pogosto uporablja, se "Cut" simboli (!), ki nastopajo na vrhnjem nivoju klavzul metod avtomatično prevedejo v večnivojske "Cut" ukaze.

Negativno znanje predstavimo s prekrivanjem in eksplicitnim pogreškom. Na primer :ptice letijo, pingvini so ptice, vendar pingvini ne letijo. Z uporabo večnivojskega "cut" skupaj s pogreški nastane podobna kontrolna struktura kot "catch" in "throw" v določenih Lisp sistemih. Kontrolna struktura je nepogrešljiva v mehanizmu obravnavanja napak, ki je potreben v skoraj vseh delih operacijskega sistema.

Metoda prirejanja

Klavzule metod v definiciji razreda so razvrščene v 3 kategorije : principielne klavzule, pred-demonske klavzule in po-demonske klavzule. Demonske klavzule se sintaktično označujejo s kvalifikatorjem "before" ali "after" pred klavzulo. Principielne klavzule v definiciji razreda za isto ime predikata in z isto ariteto oblikujejo principielni predikat, kakor niz klavzul oblikuje predikat v običajnem Prologu. Podobno formirajo pred-demonske klavzule pred-demonski predikat.

Metoda se uvede s predikatom "method". Telo predikata "method" sestavlja AND kombinacija naslednjega drevesa:

-- AND kombinacija klicev vseh pred-demonskih predikatov definiranih v prirejenih razredih. Zaporedje je tako kot je bilo prirejanje.

-- OR kombinacija klicev vseh principielnih predikatov, definiranih v prirejenih razredih v zaporedju prirejanja.

-- AND kombinacija klicev vseh po-demonskih predikatov definiranih v prirejenih razredih v obratnem zaporedju prirejanja. Zaporedje je obrnjeno zato, da se pred in po-demonski predikati pravilno vgnezdijo.

```
method_predicate(A1,.....,Am) :-

    b1(A1,.....,Am), ... ,bn(A1,....,Am),
   (p1(A1,.....,Am), ... ,pn(A1,....,Am)),
    a1(A1,.....,Am), ... ,a1(A1,....,Am).
```

Slika : kombinacija metode

Vsi klici si delijo iste argumente. Kadar ima razred n superrazredov (vključno z originalnim razredom) in ima vsak razred k pred-demonske, principielne in po-demonske predikate bk pk in ak pripadajoče isti metodi, potem bo predikat metoda kakor na sliki.

Raba demonov

Predpostavimo primer, da ima razred vrata_s_ključavnico metodo "odprt", ki uspe samo takrat, če so vrata odkljenjena. Imeli smo razred "vrata", ki je imel metodo "odprt", toda to je vedno uspelo. Razred s_ključavnico moramo definirati tako, da ima pred-demonsko klavzulo za "odprt", ki testira stanje ključavnice in uspe samo takrat, če je odklenjena. S povezavo dveh klavzul definiramo izbran razred vrata_s_ključavnico. V tem primeru bo metoda "odprt" razreda vrata_s_ključavnico :

```
     vrata_s_kljucavnico                    s_kljucavnico
odprt                          :- odprt                  ,
     metoda                                    pred-demon

                                               vrata
                                          odprt
                                               princip
```

Principielni predikat v splošnem predstavlja glavni posel metode. V gornjem primeru igra to vlogo :

```
                                               vrata
                                          odprt
                                               principielni
```

Če obstajajo pred-demoni, testirajo če je objekt v primernem stanju, da sprejme sporočilo oz. če so argumenti podani ob klicu metode ustrezni. V našem primeru:

```
                                               s_ključavnico
                                          odprt
                                               pred-demonski
```

testira stanje ključavnice v vratih. Zatem demoni testirajo vrnjene vrednosti. Kadar principielni predikat vrne neko vrednost se spremenljivke unificirajo v dane argumente, kot je to običajno pri logičnih jezikih. Potem ko demoni prejmejo nekaj argumentov se prične testiranje.

Mehanizem demonov se uporablja v različnih delih sistema SIMPOS. Predvsem v podsistemu oken, ker zahteva zelo kompliciranо kontrolo. Brez nemonotonega mehanizma bi bilo projektiranje SIMPOS znatno bolj kompliciranо delo.

4. Makroji

Ena od največjih težav pri programiranju z logičnimi programirnimi jeziki je to, da v osnovi jeziki ne dovoljujejo notacij funkcij, razen v posebnih primerih (npr.: aritmetični izrazi v DEC-10 Prologu). Če želimo podati vsoto X in Y kot argument predikata P, moramo zapisati :" add(X,Y,Z),P(Z) ".Uvedba makro ekspanzij v ESP omogoča notacijo funkcij v obliki P(X+Y). To je bolj čitljivo, posebno če je izraz daljši.

Meta jezik

Makroji služijo za pisanje meta programov in opisujejo kake strukture se naj prevedejo v kake programe. Najtežji del projektiranja makro ekspanzij je izbor meta jezika za meta programe. Obstajata dve široko uporabljani družini jezikov, v katerih se uporabljajo makroji. Lisp in njemu podobni jeziki ter zbirniki. Makroji v Lispu so zmogljivejši in

lažje uporabljivi kot v zbirniku. To pa zato, ker je Lisp sam po sebi meta jezik v Lispu, v zbirnikih pa je meta jezik bistveno drugačen jezik s specializiranimi funkcijami. Lisp je lahko sam po sebi meta jezik, ker se podatki obravnavajo kot programi. V ESP je izbran ESP za meta jezik. Z vgrajenim primerjanjem vzorcev in možnostjo logičnega sklepanja so definicije makrojev celo bolj fleksibilne kot v Lispu. V različnih jezikih z možnostjo makro ekspanzij se klic makroja enostavno nadomesti z ekspandirano obliko. Tak mehanizem je zadosten v Lispu in njemu podobnih jezikih,ne zadošča pa v Prologu in njemu podobnih jezikih. Na primer: makro, ki ekspandira cilj:

p(a,f(x+y))

v zaporedje ciljev :

add(X,Y,Z),p(a,f(Z)

ne pa v :

p(a,f(add(X,Y))

ne more biti definiran z mehanizmom enostavne zamenjave. Popoln format makro definicije v ESP je prikazan na sliki.

Pattern ==> Expansion

when Generator where Checker

:- Condition.

Slika : makro definicije.

Vzorec, ki unificira s "Pattern" ekspandira v "Expansion" edino takrat, kadar "Condition" uspe. V tem trenutku se razbijeta "Generator" in "Checker" v ekspandirano obliko in se postavita na ustrezno mesto :

-- če se klic makroja pojavi v telesu cilja, se "Generator" postavi izpred in "Checker" za ciljem, ki vključuje klic.

-- če se klic makroja pojavi v glavi klavzule, se "Generator" doda na konec telesa in "Checker" na začetku telesa.

Na primer makro definicija :

X + Y ==> when add (X,Y,Z) .

X + Y je vzorec "Pattern", Z je "Expansion" in "add (X,Y,Z)" "Generator". "Checker" in "Condition" sta v tem primeru prazna. Ista definicija se lahko uporabi na dva načina. Klavzula :

increment(M,M+1)

ekspandira v klavzulo:

increment(M,N) :- add(M,1,N)

medtem, ko telo cilja:

...,p(M+1),....

ekspandira v zaporedje ciljev:

..,add(M,1,N),p(N),..

V zapletenih makro definicijah se "Condition" lahko uporabi za odločitev, ali se klicni vzorec ekspandira ali ne. Lahko se uporabi za računski del od "Expansion" s pisanjem spremenljivk v "Expansion" in prirejanjem spremenljivk v "Condition". S tem je omogočena enostavna optimizacija, ker se vrednosti konkretnih izrazov izračunavajo v fazi prevajanja.

5. Uporaba ESP prevajalnika

Od avgusta 1984 je na "main-frame" računalniku na razpolago prečni prevajalnik iz ESP v KLØ. S povezovanjem objektne kode z majhnim "runtime-support" sistemom napisanim v KLØ, se programi lahko izvajajo direktno na PSI računalnikih. Očitna je potreba po objektni usmerjenosti. Objekt je predstavljen z vektorjem, shranjenem v enem od polnilnih področij. Prvi element vektorja je kazalec na opisnik objekta, ki vsebuje naslova dveh tabel. Opisnik je razdeljen med prireditve, ki pripadajo istemu razredu. Torej je za objektno orientiran mehanizem klicanja potrebna 1 beseda. Ostali elementi vektorja so za shranjevanje vrednosti sestavov objekta.

Prva tabela, na katero kaže opisnik objekta se imenuje tabela metod. Tabela metod povezuje atome predikatnih imen s predikati metode. Druga tabela se imenuje tabela sestavov, ki povezuje atome imen sestavov z relativno pozicijo sestava znotraj objekta.Tabeli sta predstavljeni kot KLØ predikata. Klice objektno orientirane metode prevede KLØ prevajalnik v klice runtime podprogramov z atomi imen metod in originalnimi argumenti.Runtime podprogram preveri prvi originalen argument, ki je vektor za predstavo objekta. Zatem preveri prvi element vektorja, ki je opisnik objekta. Zatem ponovno preveri prvi element, ki je tabela metode. Tabela metod se pregleda z uporabo imena metode in s številom argumentov kot ključem, z uporabo kode objekta, ki je predikat metode. Končno se ta predikat pokliče z originalnimi argumenti.

V večini primerov so nekateri klici predikatov v mehanizmu objektno orientiranih povezav redundantni. Na primer, če metodo sestavlja samo eden principielni predikat, ni potrebe za predikat metode, ki vsebuje demonsko kombinacijo. Te optimizacije opravi prevajalnik med prevajanjem.

Kar se tiče hitrosti izvajanja, sedanja verzija ESP ni zadovoljiva. Klici metod so 3 do 4 krat počasnejši, kot direktni klici KLØ predikatov. Procesiranje sestavov ima enak balast. Izvori balasta so klici "runtime-support" podprogramov in preiskovanje tabel sestavov.

Planirane izboljšave

Planira se vgradnja novih predikatov v KLØ za zmanjšanje balasta. Klici runtime podprogramov bodo nadomeščeni z vgrajenimi predikati in funkcijo podprogramov bo izvajala strojna oprema. Dostopi do sestavov bodo prav tako prevedeni v klice vgrajenih predikatov. Mehanizem preiskovanja tabel metod je dokaj neučinkovit zaradi v KLØ vgrajenega mehanizma indeksiranja klavzul. Preiskovanje se lahko pospeši s podporo firmvera. Z uporabo firmvera bodo tabele shranjene v hitrih vmsnih pomnilnikih, kar bo zmanjšalo potrebe po centralnem pomnilniku.Ob podpori planiranega firmvera bodo klici metod dvakrat hitrejši. Izboljšave pri dostopanju v sestave bodo še večje, ker posegi ne bodo vključevali klicev predikatov.

Nadaljna optimizacija

Za zmanjšanje balasta pri preiskovanju tabel so zelo učinkoviti hitri vmesni pomnilniki (cashe), v katere se shranijo imena najbolj pogosto klicanih metod, njihovi prvi argumenti in ustrezni naslovi kod predikatov. Hitri vmesni pomnilnik bi moral biti dovolj velik, da bo razmerje zadetkov zadovoljivo. Potrebna bo posebna izvedba, kot je na primer asociativni pomnilnik.

6. Izkušnje s SIMPOS

Preliminarne verzije treh najnižjih podsistemov SIMPOS : jedro, nadzornik in V/I podsistem so kodirane v ESP in v celoti testirane na PSI. Trenutno se preverjajo izboljšave teh podsistemov in programskega sistema SIMPOS.

Med razvojem SIMPOS je postalo jasno, da mehanizem večkratnega prirejanja zelo olajša deljenje kode. Na primer funkcije za modul nadzornik, ki poslujejo s tabelami, imeniki, procesi in tokovno orientirano komunikacijo med procesi, je mogoče uporabiti direktno v modulih na višjem nivoju.

Spreminjanje programskih modulov je dokaj enostavno. Moduli SIMPOS so ločeno razviti, kljub temu, da so tesno povezani. Pri spremembi je v večini primerov dovolj ponovno prevajanje. Včasih se je zgodilo, da so bili povezani moduli napačnih verzij. Zato je skoraj nepogrešljivo avtomatsko poslovanje z verzijami modulov. Mehanizem makro ekspanzij omogoča bolj čitljive programe. Posebej ker dovoljuje aritmetične izraze kot argumente.

Ostale aplikacije

Kljub malo izkušnjam z ESP, so se lastnosti jezika pokazale kot zelo dobre. Če bi bil SIMPOS kodiran direktno v KL0, bi bilo projektiranje mnogo težje.Ker se analizira odnos povezav med razredi statično med prevajanjem, je nepogrešljiva programska podpora za avtomatsko sledenje verzij. Programska podpora za sledenje verzij bo dodana v knjižnico SIMPOS.

==========

SEKVENČNI PROLOG RAČUNALNIK

==========

PEK je eksperimentalni računalnik, konstruiran za izvajanje Prolog programov. Uporablja LSI bit-rezine za sekvenčnik in ALU , ter mikroprogramirano kontrolo. Da omogoča veliko hitrost izvajanja Prolog programov, vključuje vezja za unifikacijo in vračanje. Razvit je enostaven Prolog interpreter. Zmogljivost stroja je okrog 40 LIPS (logičnih sklepanj na sekundo), kar je enako, kot zmogljivost DEC-10 Prologa na računalniku DEC 2060.

1. Uvod

Prolog je logični programirni jezik in se največ uporablja na področju umetne inteligence. Mehanizem izvajanja v Prologu se zelo razlikuje od konvencionalnih jezikov. Zato je pomembno raziskati arhitekture za izvajanje Prolog programov.

PSI je konstruiran za raziskave in razvoj. Ima posebno arhitekturo : strojne sklade in multi-procesne funkcije. PEK vključuje specifične strojne funkcije, kot je vezje za avtomatsko sledenje, za avtomatsko sproščanje itd. Da bi dosegli hitrost 100 LIPS na enem procesorju, mu je potrebno dodati posebne strojne funkcije. Sekvenčni Prolog računalnik sta projektirala Evan Tick in David Waren. Uporablja cevenje (pipelining), reduciran instrukcijski set, prepleten pomnilnik. S tem se poveča hitrost izvajanja Prolog programov. PEK je eksperimentalen računalnik za raziskovanje arhitektur, ki bi omogočile hitrejše izvajanje Prolog programov.

Sekvenčno izvajanje

Znano je, da je za učinkovito izvajanje logičnih programov potrebna velika mera vzporednosti. Pomembno pa je raziskati kolikšna je največja prepustnost stroja pri sekvenčnem izvajanju. Zato je v PEK računalniku malo vzporednosti, razen :

-- prenos podatkov je izveden v 3 poljih: okvirnem, označbenem in polju vrednosti.

-- podatkovna področja so ločeno razporejena v spominskih modulih, kot so skupni pomnilnik, pomnilnik procesa, vodilni sklad itd.

-- vodoravni format mikro-instrukcij simultano nadzira strojne module.

-- sproščanje prireditev spremenljivkam izvaja podsekvenčnik.

-- branje instrukcij teče po načinu cevenja.

Mikroprogramska kontrola

PEK uporablja mikroprogramsko kontrolo in Prolog interpreter je napisan v mikro-kodi. Prevajalnik pretvarja Prolog programe v mikro-kodo.

Metoda deljenih struktur

PEK uporablja deljene strukture, tako da je za procese, ki potrebujejo dodatne informacije predvidena posebna strojna oprema.

Ostale lastnosti

Ker so podatkovna področja ločeno razporejena po pomnilniku, je možno v ta področja posegati s kompleksim načinom naslavljanja, ki ga izvaja strojna oprema :

-- indeksno naslavljanje za pomnilnik procesa

-- naslavljanje prek sklada za strojni sklad

-- post-inkrementalno za naslovne registre

-- izračunavanje naslovov za celice spremenljivk in globalni sklad

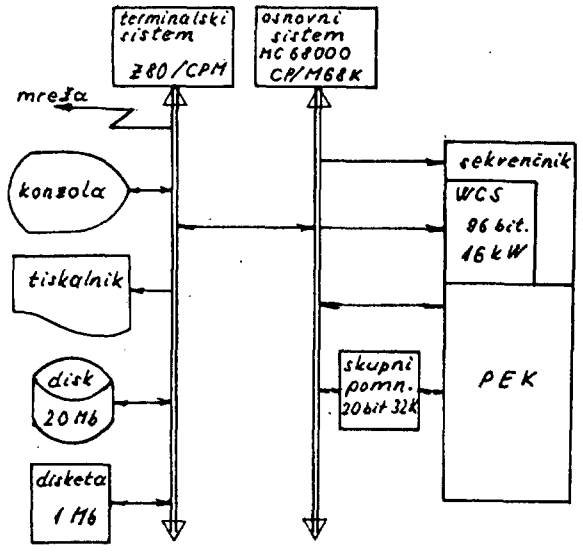
PEK vključuje specialno strojno opremo za unifikacijo in vračanje:

-- primerjalna vezja

-- vezje za avtomatsko sledenje

-- vezje za samodejno sproščanje

2. Konfiguracija sistema



Slika: konfiguracija PEK sistema.

Sistem sestavlja osnovni procesor MC 68000 in PEK. Vse V/I enote so priključene na osnovni procesor prek Z-80. Osnovni procesor inicializira PEK in podpira V/I operacije med izvajanjem Prolog programov.

PEK kontrolira osnovni procesor prek CML registra (izvajanje HALT, STEP itd.) Ostali registri za komunikacijo so ICR ("input commands" osnovnega računalnika k PEK) in OCR ("output commands" od PEK k osnovnemu računalniku).

Strojna oprema

Strojna oprema obsega 5 plošč:

-- CCU
sekvenčnik, WCS, vmesnik za osnovni procesor MC 68000

-- ALU , "by-pass" kontrolerji, pomnilnik procesov, strojni sklad

-- Unifikacijska plošča :
globalni / sklad, vodilni sklad, primerjalno vezje, sprostitveno vezje.

-- Skupni pomnilnik :
ima dva naslovna registra,dva registra za čitanje in dva za pisanje.

-- Plošča za evaluacijo sistema :
ura za čas izvajanja, števec mikroinstrukcij.

Format besede

Beseda je sestavljena iz 14 bitov dolgega okvirnega polja in 20 bitov dolgega polja za terme, da je omogočen prenos molekul. Polje terma je razdeljeno na 4 bitno označbo in 16 bitno polje za vrednost.Uporabljene so naslednje označbe:

-- celo število
-- atom literal
-- nedefinirano
-- globalna spremenljivka
-- lokalna spremenljivka
-- prosta spremenljivka
-- struktura (term)
-- struktura (seznam)
-- konec strukture
-- klavzula
-- koda

Seznami so predstavljeni z enodimenzionalnimi vektorji. Imajo ločeno označbo in povezano shranjene elemente. Na koncu vsakega seznama je dodana posebna označba EOS (end of structure), ki določa dolžino seznama. Sestavljeni termi so sestavljeni na podoben način kot seznami.

Spominski moduli

Spominski moduli so razvrščeni glede na njihovo uporabo in se lahko procesirajo vzporedno. Spominska integrirana vezja imajo pristopni čas 55 nsec.

-- WCS vsebuje mikro program 96 bitov X 16 K besed.

-- skupni pomnilnik: 20 bitov X 32 K besed

-- pomnilnik procesa: 20 bitov X 16 K besed

-- globalni sklad

-- vodilni sklad: 14 bitov X 16 K besed pomnilniški sklad

-- strojni sklad: 34 bitvo X 4 K besed

-- registerske datoteke: 34 bitov X 16 besed

Interna vodila

R, S in Y vodilo v širini 34 bitov.

Sekvenčnik

Sekvenčnik sestoji iz 4 Am 2909 A 4 bitnih rezin LSI. Minimalni izvajalni čas je 120 nsec za izvajanje instrukcije in 160 za večino vejitvenih ukazov.

Aritmetična logična enota

Vsebuje 9 4 bitnih rezin LSI, ki so razdeljene v 3 bloke, glede na format besed. ALU operacije so neodvisne za vsako polje kakor tudi rezultati operacij. Nadalje je dodanih 9 LSI kot eksterna register datoteka, kjer je shranjenih 32 registrov.

Obhodni kontrolerji

Predvidena sta dva obhoda. Eden za prenos podatkov iz R vodila k Y vodilu in drugi za prenos podatkov od S vodila k Y vodilu. Uporaba obhodov omogoča na primer prenos podatkov od izvornega vodila k namembnemu vodilu in simultane operacije ALU.

Pomikalniki

Uporabljena sta dva. Levi za pomikanje spodnjih 14 bitov polja terma v polje okvirja. Desni pomika okvirno polje ali označno polje v polje vrednosti.

Primerjalno vezje

Z njim se primerjata dva atoma z eno instrukcijo.

Avtomatično določanje naslovov za celice spremenljivk

Naslovi celic spremenljivk se izračunavajo

avtomatično z uporabo dveh registrov. Če je spremenljivka prosta ali vezana, se ugotovi iz indikatorja in ni potrebno odčitati vrednosti iz globalnega sklada.

Avtomatsko sledenje

Vračanje zahteva sproščanje prireditev oz. resetiranje vrednosti spremenljivk v nedefinirano stanje. Pri prireditvi se mora naslov celice shraniti na globalni sklad. PEK izvede to operacijo strojno. Kadar se izvede operacija pisanja na globalni sklad, se naslov avtomatično zapiše na vodilni sklad.

Avtomatično sproščanje

Sproščanje se v PEK vrši avtomatično in ga izvaja sekvenčnik. Kadar je vodilo prosto, se izvaja sproščanje vzporedno z glavnimi operacijami sekvenčnika. Pobiranje z vodilnega sklada in zapisovnaje v globalni sklad se prekrivata, kar poveča hitrost operacije.

Čitanje struktur v cevnem načinu

Kadar se izvaja Prolog unifikacija med dvema elementoma strukture, PEK vsebuje dva naslovna registra (AD1, AD2), dva pisalna registra (WR1, WR2) in dva čitalna registra (RD1 in RD2). Kadar se čita RD1 ali RD2, se Ad1 ali Ad2 avtomatično poveča za 1 in prične se operacija čitanja naslednjega podatka. Podatek z naslednjega naslova se vpiše v RD približno 250 nsec kasneje. Okvirno polje čitalnega registra ima enako vrednost kot okvirno polje, shranjeno v naslovnem registru.

3. Razvojna programska oprema

Mikro-zbirnik

Mikroinstrukcija PEK računalnika je horizontalna, dolga 96 bitov in vsebuje 24 polj. V začetku je bil napisan zbirnik, ki je specificiral mnemonike za vsako polje. Ugotovili so, da so programi težko čitljivi. Zato so razvili podprocesor, ki vključuje klice makro funkcij s primerjavo vzorcev .

Na PEK se uporablja Am 2925 časovni generator, ki lahko generira 8 vrst 4 faznih časov. Izbira časa se vrši s 3 bitnim CYC poljem v mikroinstrukciji. Na ta način lahko mikro-zbirnik pri zbiranju izbere najprimernejši cikel.

Napisana sta dva programa, ki ugotavljata dolžino cikla. Prvi ugotavlja dolžino cikla tako, da razdeli operacijski čas za vsako enoto na enote 40 nsec. Drugi ugotavlja dolžino cikla tako, da poišče najdaljšo podatkovno pot in skrajša čas cikla za 15 % glede na prvi program. Čas cikla se lahko skrajša za nadaljnih 40 % pod posebnimi pogoji: če je bil izbran naslov z globalnega sklada v predhodni mikro instrukciji, potem se zveča hitrost čitanja z globalnega sklada za 96 nsec.

Monitor/očiščevalnik

Monitor je razvit na osnovnem procesorju MC 68000 in služi za odkrivanje napak v strojni opremi. Monitor obsega ekranski editor, mikro-zbirnik, razbirnik in dodatke kot so: postavljanje prekinitvenih točk, izpisovanje vsebine registrov itd.

4. Interpreter

Prolog jezik je kompatibilen z DEC-10 Prologom. Razvit je enostaven Prolog interpreter za merjenje zmogljivosti sistema. V sedanji verziji so vse spremenljivke obravnavane globalno in niso implementirane zahtevne funkcije, kot je indeksiranje klavzul ali optimizacija rekurzije. Za testiranje sta bila izvedena z interpreterjem 2 Prolog programa. Prvi je dodajal seznam n elementov v prazen seznam, drugi pa invertiral seznam elementov.Povprečen cikel je znašal okrog 170 nsec, torej je hitrost okrog 40 K LIPS.

Literatura

FIFTH GENERATION COMPUTER SYSTEMS 1984

Proceedings of the International Conference on Fifth Generation Computer Systems 1984
Tokyo, Japan, November 6-9, 1984

```
**************************************
*  Podatkovno pretokovni procesor    *
*              µPD7281               *
**************************************
```

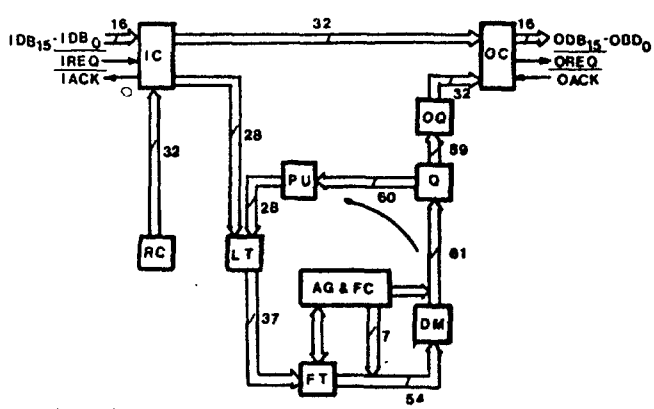
Načrtovalci sistemov za procesiranje slik so običajno prisiljeni poiskati kompromis med hitrostjo in fleksibilnostjo sistema. Sistem, ki ga sestavljata miniračunalnik za procesiranje slike in masovni pomnilnik za shranjevanje slik, je okoren in počasen, kar je nesprejemljivo za delo v realnem času. Z dodatkom posebne materialne opreme se hitrost procesiranja poveča, vendar vsaka sprememba programske opreme narekuje spremembo materialne opreme - fleksibilnost sistema se zato zmanjša.

Omenjeni razkorak med hitrostjo in fleksibilnostjo sistema je moč omiliti z uporabo programabilnega slikovnega procesorja, ki se odlikuje s podatkovno vodeno arhitekturo. Primer takšnega procesorja je NEC µPD7281, katerega moč temelji na krožno organizirani pipeline arhitekturi ter bogatem naboru ukazov. µPD7281 je prvi VLSI čip, ki deluje po načelih podatkovno pretokovne arhitekture. Ta omogoča večjo učinkovitost procesorja v mnogih večprocesorskih aplikacijah, kot sta procesiranje slik ter razpoznavanje vzorcev na področju umetne inteligence, kjer se uporabljajo algoritmi za dvodomenzionalno konvolucijo, povečavo, pomanjšavo in rotacijo. Njegova učinkovitost postane očitna predvsem pri procesiranju slik v realnem času, kjer dodobra izkorišča vsebovane vzporednosti uporabljenih algoritmov. µPD7281 ni uporaben le pri procesiranju slik, temveč tudi pri zahtevnih numeričnih izračunih, kot so matrično množenje, matrično vektorsko množenje, aritmetika s plavajočo vejico ter izračuni transcendentnih funkcij v realnem času.

V nasprotju s von Neumannovimi procesorji, procesira µPD7281 pakete (tokens), ki so nosilci operandov in med izvrševanjem ne potrebuje dostave ukazov. V von Neumannovih procesorjih je vsak korak sestavljen iz dostave, dekodiranja ter izvršitve ukaza, pri čemer se vsi trije deli izvršijo zaporedno. Tako je izračun A = B + C je sestavljen iz dostave ukaza "seštej B in C ter shrani v A" v procesor, iz dekodiranja ukaza, dostave operandov B in C v procesor, seštetja obeh operandov ter prenosa rezultata A v pomnilnik.

Podatkovno pretokovni procesor ne pozna dostave ukaza. Namesto tega vsebuje 'grafni' pomnilnik (LT in FT), v katerega se pred začetkom izvrševanja vpiše programski podgraf. Pretok podatkov se vrši s pomočjo paketov, ki vsebujejo polje z naslovom procesorja, identifikator, podatkovno ter krmilno polje. Notranja krožna pipeline organizacija (Slika 1) omogoča procesni enoti neprekinjeno delovanje s hitrostjo 5 MHz. Procesna enota vsebuje 17 x 17 bitni množilnik ter ALU, ki omogoča ostale standardne aritmetično logične operacije. Nabor ukazov je širši kot pri večini klasičnih von Neumannovih procesorjev.

Komunikacija z okolico poteka s pomočjo vhodne in izhodne enote (IC in OC). Glavni procesor pošlje paket procesorjem µPD7281. Iz naslovnega polja paketa µPD7281 ugotovi, če je paket namenjen njemu ter ga v tem primeru sprejme, izloči naslovno polje in pošlje v 'obtok' - najprej v LT. Paket, ki ni namenjen danemu procesorju, se nespremenjen pošlje v istem ciklu preko izhodne enote (OC) naslednjemu µPD7281. Procesor je tako za 'tuje' pakete praktično transparenten. Na ta način vsak procesor zbira 'svoje' pakete. Med 'obtokom' po µPD7281 paket še nekajkrat spremeni vsebino in dolžino. Identifikator sprejetega paketa omogoča določitev pripadajoče povezave v opisu programskega podgrafa. Vsebina identifikatorja je naslov lokacije v LT (povezava). Iz LT pride paket z novim identifikatorjem (ki je vsebina naslovljene lokacije v LT) in vstopi v FT, kjer se nahajajo opisi točk programskega podgrafa. Del identifikatorja je naslov lokacije v FT. Vsebina naslovljene lokacije v FT opisuje točko programskega podgrafa in je sestavljena iz dveh delov. Prvi del je koda operacije in je del novega identifikatorja ob izstopu paketa iz FT - namenjen je procesni enoti. Drugi del pa posredno opisuje oznake povezav, ki vstopajo v točko in služi AG&FC pri generiranju naslova lokacije v DM. Ta naslov se doda kodi operacije v okviru novega identifikatorja. Tako spremenjen paket vstopi v 'paketni' pomnilnik DM. Če predstavlja ta paket zadnjega izmed operandov, se skupaj s svojim parom (ta že čaka v DM), preko vrste Q prenese v procesno enoto PU, sicer pa se podatkovno polje paketa shrani v naslovljeno lokacijo v DM. Vpis ter branje iz paketnega pomnilnika potekata sočasno z izvrševanjem v procesni enoti. Če operacija zahteva en sam operand, se paket prenese iz FT direktno preko vrste Q v procesno enoto PU. Procesna enota sestavi paket, ki vsebuje rezultat in identifikator, ki je oznaka izhodne povezave. Paket ponovno vstopi v LT, ter nato v FT. Če procesor ne vsebuje nobene operacije, ki bi potrebovala ta paket, prenese AG&FC ta paket preko vrste Q in izhodne vrste OQ v izhodno enoto OC.



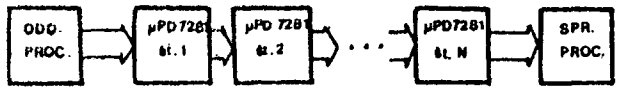
IC: Input Controller
OC: Output Controller
LT: Link Table
FT: Function Table
DM: Data Memory
Q: Queue
PU: Processing Unit
OQ: Output Queue
AG&FC: Address Generator and Flow Controller
RC: Refresh Controller

Slika 1

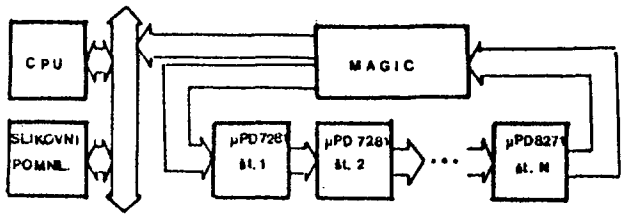
Npr., za izvršitev operacije A = B + C mora procesor prejeti paketa, ki nosita vrednosti B in C, šele nato lahko izvrši operacijo seštevanja. Vhodna paketa lahko prispeta v poljubnem zaporedju, saj je procesor sposoben razpoznavati pakete. Iz opisa podgrafa, ki je vnešen v procesor pred izvrševanjem, le ta ugotovi, da paketa B in C pripadata preko operacije '+' paketu A, zato paketa B in C združi in pošlje v procesno enoto. Vrednost, ki je rezultat izvršitve v procesni enoti, se vstavi v paket in opremi z oznako A. Če je A vhodni paket neke nove operacije, ki se mora izvršiti v istem µPD7281, se shrani v paketni

pomnilnik, dokler ne prispe v procesor tudi paket, ki nosi vrednost drugega operanda. Tedaj se opisani postopek ponovi. Pravilna izbira predhodno vpisanega podgrafa zagotavlja neprekinjeno delo procesorja.

Procesorji µPD7281 se povezujejo v večprocesorski sistem na dva osnovna načina: kaskadni (Slika 2a) in krožni (Slika 2b). Za krožno arhitekturo je v razvoju tudi podporni čip MAGIC (Memory Access & General bus Interface Chip).



(a)



(b)

Slika 2

Opravljena testiranja opravičujejo uporabo večprocesorske arhitekture z µPD7281. Tako na primer rotacija binarne slike 512 X 512 zahteva 0.6s pri krožni povezavi treh procesorjev; en procesor porabi za isto nalogo 1.5 s. Za izračun funkcije cos(x) potrebuje en procesor 40µs, kaskada treh procesorjev pa 15µs.

J. Šilc in B. Robič