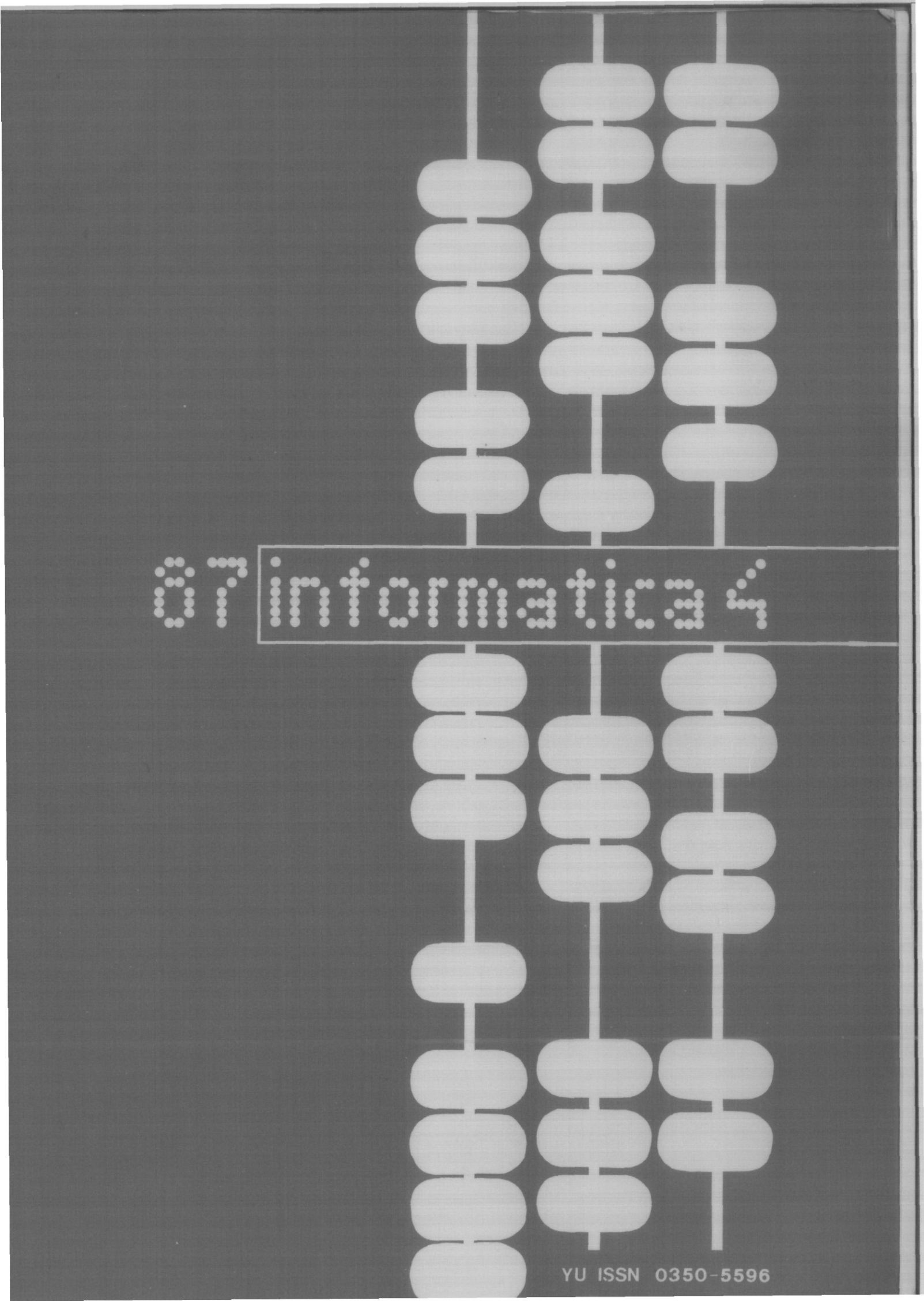


YU ISSN 0350-5596

informatica

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350-5596

VOLUME 11, 1987 – No. 4

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$ 30 for companies, and US$ 15 for individuals

Printed by: Tiskarna Kresija, Ljubljana

CONTENTS

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

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

YU ISSN 0350-5596

LETNIK 11, 1987 - ŠT. 4

Uredništvo in uprava:

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

Letna naročnina za delovne organizacije znaša 11990 din, za zasebne naročnike 2990 din, za študente 990 din; posamezna številka 4000 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

V S E B I N A

ANALYSIS OF BUFFERED MULTISTAGE INTERCONNECTION NETWORK FOR PARALLEL PROCESSORS

INFORMATICA 4/87

Dušan Fajfar
Institute for Teleinformatics
ISKRA Telematika, Kranj
Matija Lokar
University E. K. Ljubljana, Department of Mathematics

UDK 618.3.02

Abstract In the paper a method for calculating distribution law on the number of memory requests in the finite buffer in multistage interconnection network of parallel processors is given. A modified delta network is used for the connecting processors with memory modules. Memory requests on each processor are generated randomly and independently. Two cases of traffic flow are discussed: the constant average rates of requests and the time dependent average rates.

Povzetek V članku analiziramo večnivojsko povezovalno mrežo med procesorji in pomnilniškimi moduli, ki jo sestavljajo elementi z vmesnimi pomnilniki omejene kapacitete. Podana je metoda za izračun porazdelitvenega zakona števila zahtev po podatkih iz pomnilnika v posameznem elementu mreže. Vsak procesor generira zahteve po pomnilniških modulih naključno in neodvisno od ostalih. Obravnavana sta dva primera: časovno konstantno in s časom spreminjajoče se povprečno število zahtev.

Keywords Buffered network, delta network, multistage interconnection network, buffer length, queueing theory.

I. Introduction

In the recent years a lot of new multiprocessor architectures have been proposed. The main problem of any multiprocessor system is its interconnection network. A typical configuration of such system is illustrated in Fig. 1. Many identical processors are connected via an interconnection network to identical memory modules. Each processor should have access to each memory module with requests generated randomly and independently at each processor.

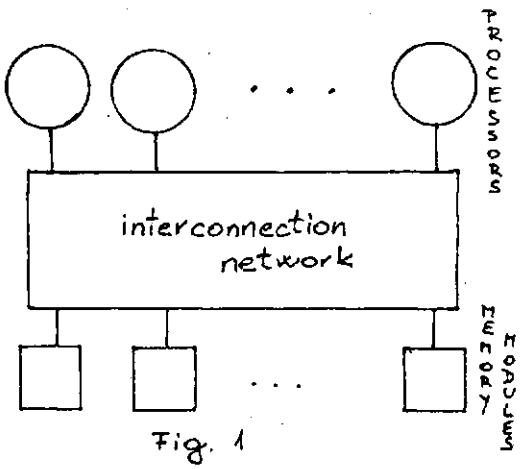


Fig. 1

We have several possible organizations of such processors - memory interconnection. For example, a shared bus in an unexpensive, but admits very low transfer rate. On the other hand maximum transfer rate is attained by the full crossbar switch (with n×m switches where n is number of processors and m number of memory modules). However it is far too complicated and expensive for practical application. Thus we have to use interconnection networks with less then n×m switches. One of them is multistage interconnection network [1]. A lot of them have been presented in the literature ([2],[3],[4],[5],[6]). In this paper we study a delta network with some modifications in the first and the last stage (input process from processors and output process to memory). In section II description of network is given. The network consists of three types of switches. For more detailed presentation see [6]. In sections III, IV and V the distribution law of buffer length for each type of switch is calculated. In section VI results and conclusions are given.

II. The network model and system operation

We discuss a multistage network for connection N processors with N memory modules, where $N = 2^n$. The model for N = 16 is illustrated in Fig. 2.

The network consists of $\log_2 N + 1$ stages. Each stage has N identical switches. With regard to the number of input and output links there are three types of switches. Switches at the first stage have one input and two output

links, switches at the last stage have two input links and only one output link. Switches at all other stages have two input and two output links. To achieve higher transfer rate and to avoid blocking there is a finite buffer at each output link, where the incoming requests wait to be processed further.

As we have finite buffers we propose that in the case when the buffer is full, the incoming requests are lost.

For the time unit we choose the system cycle time. At each cycle only one request can be transmitted through the same link except at the first stage where more than one request can come by input links from each processor. In one time unit only the first request in the buffer can travel from the stage i to stage i + 1. As we can see on Fig. 2. there are no links between the switches of the same stage. The input links of stage i are the output links of the stage i-1. The first stage has input links from processors and the last stage has output links to memory modules. This regularity of the network gives the possibility to analyze it stage by stage instead of the whole network at once. In the next section we give the analysis of the first stage in the section IV we give analysis of the stages indexed from 2 to $\log_2 N$ and in the section V the last stage is analyzed.

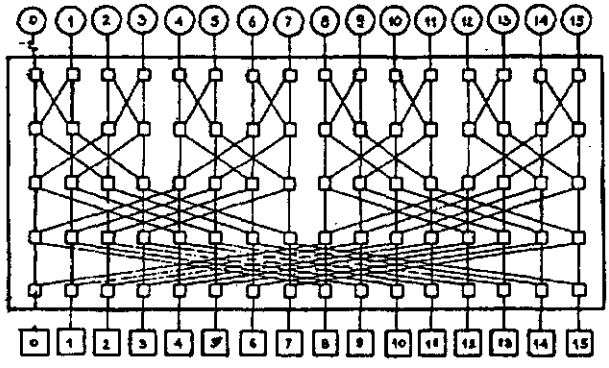


Fig. 2

III. Switches with one input and two output links

Switches with one input and two output links can be found only in the first stage. The input links come from processors and the output links are connected to switches of the second stage. Each processor is connected to only one switch. There are no links between the switches of the same stage, so we have N identical systems. Each system consists of a processor that sends the requests to switch and two output links, each with finite buffer.

As the memory requests are time independent ([6]), we use the Poisson distribution with a given average rate a. The average rate can be different for each processor. Let p^k denote the probability that k requests are sent from processor in one cycle. By Poisson law we get

$$p^k = \frac{a^k e^{-a}}{k!} \qquad (1)$$

We analyze the case where a is constant and the case where a is time dependent, denoted by a(t).

Each request coming from processor is switched in one of two output buffers. As requests are uniformly distributed between memory modules ([6]) there is an equal probability that the incoming request joins the first or the second buffer. By p_m^k we denote the probability that m of k requests enter the first buffer (and k-m requests enter the second buffer). Thus we get

$$p_m^k = \binom{k}{m} (0.5)^k \qquad (2)$$

Let ρ_m denote the probability that m requests enter the first buffer in one cycle. Then

$$\rho_m = \sum_{k=m}^{\infty} p^k p_m^k = \sum_{k=m}^{\infty} a^k e^{-a} / k! \binom{k}{m} (0.5)^k = a^m e^{-a/2} / (2^m m!) \qquad (3)$$

The output process is very simple if we compare it with the input process. In each cycle only the first request from each buffer leaves system.

Let us first analyze stationary system with constant average rate of input process. Since events on both buffers are equal we could analyze only one of them. Let bl denote the buffer length (we propose that all buffers have the same length but we do not have any difficulties when buffers have different length. We just have to calculate the distribution for all switches). The balance diagram is shown in Fig 3.

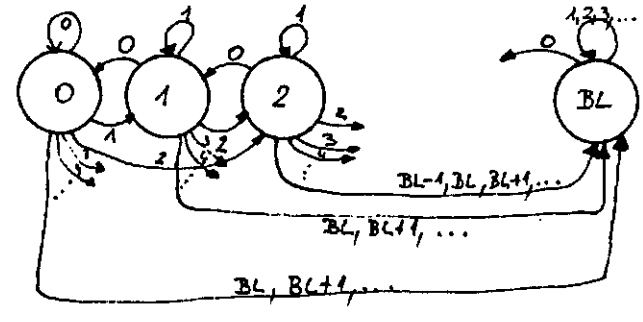


Fig. 3

The nodes in the diagram denote the number of requests in the buffer. If we look at in- and out-coming arcs to each node we get the system of balance equations

where $\Omega(s)$ denotes the probability that we have s requests in the buffer.

$$\Omega(0) = \sum_{m=1}^{\infty} \rho_m = \Omega(1) * \rho_0$$

$$\Omega(s) = [\rho_0 + \sum_{m=2}^{\infty} \rho_m] =$$

$$= \Omega(0) * p_s + \sum_{k=1}^{s-1} \Omega(k) * p_{s-k+1} + \Omega(s+1)\rho_0$$

$$s = 1, 2, \ldots, bl - 1 \quad (4)$$

$$\Omega(bl) = [1 - \rho_0]$$

$$= \Omega(0) * \rho_{bl} + \sum_{k=1}^{bl-1} \Omega(k) * p_{bl-k+1}$$

If we pot ρ_m expressed by (3) in the left side of equations (4) we get the following system of linear equations.

$$\Omega(1) = (e^{-a/2} - 1) * \Omega(0)$$

$$\Omega(s) = [1 - (a/2)e^{-a/2}] =$$

$$= \Omega(0) * p_s + \sum_{k=1}^{s-1} \Omega(k) * p_{s-k+1} + \Omega(s+1)\rho_0$$

$$s = 1, 2, \ldots, bl - 1 \quad (4')$$

$$\Omega(bl) = e^{-a/2}$$

$$= \Omega(0) * \rho_{bl} + \sum_{k=1}^{bl-1} \Omega(k) * p_{bl-k+1}$$

As the system (4') is trivialy solved by putting all $\Omega(s)$ to 0, we add the normalization equation (5).

$$\sum_{s=0}^{\infty} p(s) = 1 \quad (5)$$

Now the system can be solved by any of the well-known methods for solving linear system of equations.

For the system where average rate of input process is time dependent we could not use the same approach. Because the system works in cycles we consider the discrete time. Instead of probabilities $\Omega(s)$ we have $\Omega(s,t)$ where the second variable denotes the cycle counter. With v_i^t we denote the probability that i requests came on input line to the buffer at the time t and is expressed by the following equation

$$v_i^t = \sum_{k=i}^{\infty} p_i^k \cdot p^k \quad (6)$$

A simple calculation gives

$$v_i^t = a(t)^m e^{-a(t)/2} / (2^m m!) \quad (7)$$

With a given buffer length bl we get the recurrence relation for the number of requests in the buffer at the time t (denoted by q_t) if in this moment V_n requests will join the buffer.

$$\begin{aligned} q_{t+1} &= \min \{bl, q_t + V_{t+1} - 1\} \quad q_t \neq 0 \\ q_{t+1} &= \min \{bl, V_{t+1}\} \quad q_t = 0 \end{aligned} \quad (8)$$

So $\Omega(s,t)$ is expressed by

$$\Omega(s,t) = \sum_{i=1}^{s+1} \Omega(i,t-1)\, v_{s-i+1}^t + p(0,t-1)\, v_s^t \quad (9)$$

if s is less than buffer length bl. In that case some of the requests are rejected. The equation is almost the same as (9), we must only replace v_i^t with $\bar{v}_i^t$, where

$$\bar{v}_i^t = 1 - \sum_{j=0}^{i-1} v_j^t \quad (10)$$

is the probability that more than i-1 requests are joining the buffer at the time t.

$$\Omega(s,t) = \sum_{i=1}^{s+1} \Omega(i,t-1)\, v_{s-i+1}^t + p(0,t-1)\, \bar{v}_s^t \quad (11)$$

We start with the distribution

$$p(0,0) = 1, \quad p(s,0) = 0$$

$$s = 1,2,\ldots,bl$$

and repeatedly calculate the probabilities.

The same approach we could use in the case where average input rate is time constant. But as equations (9) and (10) can not be simplified, the calculation with this approach is time consuming.

IV. Switches with two input and two output links

Switches with two input and two output links we find in stages form 2 to $\log_2 N$. Two input links are connected to two switches of the previous stage and the output links are connected to two switches of the next stage.

The input process now depends on the output process of the previous stage. If the buffers that are connected to input links of the switch are not empty, we get a request out of them. As we have only two input links, we can not get more than two requests in one cycle. The output process is just the same as described in section III.

Again we first analyze the stationary case with constant average rate. As we could see from the previous section and as is shown in this, probabilities that buffers are not empty (in this case we get a request) are not changing with time. Let Π_1 and Π_2 be the

probabilities that the buffers from where input links come are empty. Then we have the following probabilities on the number of incoming requests

$$\begin{aligned} p^0 &= \Pi_1 * \Pi_2 \\ p^1 &= \Pi_1 (1 - \Pi_2) + \Pi_2 (1 - \Pi_1) \\ p^2 &= (1 - \Pi_1)(1 - \Pi_2) \\ p^k &= 0 \qquad k = 3, 4, \ldots \end{aligned} \tag{12}$$

As the path from the processor to the memory module is completely random, requests join each of the two buffers with equal probability. So we get

$$\begin{aligned} p_0^0 &= 1 \\ p_0^1 = p_1^1 &= p_1^2 = 0.5 \\ p_0^2 = p_2^2 &= 0.25 \end{aligned} \tag{13}$$

For the probability of m requests coming into buffer in one cycle we get

$$\rho_m = \sum_{k=m}^{2} p^k p_m^k \qquad m = 0,1,2 \tag{14}$$

The balance diagram for this type of switch is shown in Fig. 4.

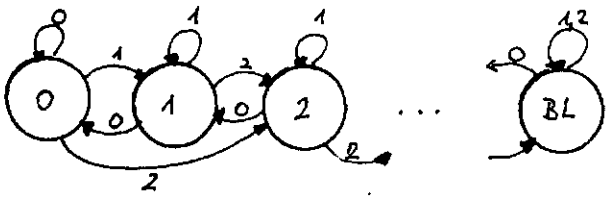


Fig. 4

The balance equatations which we obtain from the diagram are

$$\begin{aligned} \Omega(0)\ (\rho_1 + \rho_2) &= \Omega(1)\ \rho_0 \\ (\Omega(0) + \Omega(1))\ \rho_2 &= \Omega(2)\ \rho_0 \\ \Omega(s)\ \rho_2 &= \Omega(s+1)\ \rho_0 \qquad s = 2, 3, \ldots, bl - 1 \end{aligned} \tag{15}$$

Now the system is already linear. We just have to add the normalization equation (5) and solve it.

For the time dependent system we use a similar procedure as in the previous section. Consider that we analyze the stage i. Then we have

$$\Omega(0,t) = 1, \quad \Omega(s,t) = 0, \ s = 1, \ldots, bl, \quad t = 0, \ldots, i-1 \tag{16}$$

After i cycles the first request can come to the switch on the stage i. The number of requests in the buffer may grow maximaly by a request per cycle, because of maximaly two incoming requests and one outgoing request (except in the case where at the previous moment the buffer was empty. But we only have a jump from 0 to 2 and after that the length grows maximaly by one). So we get the following system

$$\begin{aligned} \Omega(0,t) &= [\Omega(0,t-1) + \Omega(1,t-1)]\ \rho_0 \\ &\quad \Omega(2,t-1)\ \rho_0 \\ \Omega(2,t) &= [\Omega(0,t-1) + \Omega(1,t-1)]\ \rho_2 + \\ &\quad \Omega(2,t-1)\ \rho_1 + \Omega(3,t-1)\ \rho_0 \\ \Omega(s,t) &= \sum_{k=s-1}^{s+1} \Omega(s-1,t-1)\rho_{s+1-k} \qquad s = 3, \ldots, bl - 1 \\ \Omega(bl,t) &= \Omega(bl,t-1)\ (\rho_1 + \rho_2) + \\ &\quad \Omega(bl-1,t-1)\ \rho_2 \end{aligned} \tag{17}$$

Equatations (17) are the same in the case where average input rate is time dependent and where is not.

V. Switches with two input and one output link

This kind of switches we find at the last stage. The input links come from the previous stage and the output links are connected to memory modules. From each switch we can reach only one memory module. The situation is almost the same as described in the section IV. The difference is only that we have just one buffer so all requests go in the same buffer. So we have to change the coefficients ρ_m defined by (12) and (14).

VI. Summary and conclusion

In the Table 1 probabilities on number of requests in the buffer for one switch on the first stage with different average input rates are given.

	average rate			
s	0.5	1	1.5	1.9
0	0.75000	0.50000	0.25085	0.07687
1	0.21302	0.32436	0.28020	0.12189
2	0.03277	0.12260	0.19489	0.12635
3	0.00385	0.03779	0.11706	0.11699
4	0.00037	0.01091	0.06787	0.10594
5	0.00004	0.00311	0.03915	0.09569
6	0.00000	0.00088	0.02258	0.08643
7	0.00000	0.00025	0.01302	0.07807
8	0.00000	0.00007	0.00751	0.07052
9	0.00000	0.00002	0.00433	0.06370
10	0.00000	0.00001	0.00250	0.05753

Table 1

	average rate			
s	0.5	1	1.5	1.9
0	0.75000	0.50000	0.25086	0.08057
1	0.22959	0.38889	0.39047	0.19734
2	0.01999	0.09876	0.23004	0.20422
3	0.00041	0.01097	0.08251	0.15007
4	0.00001	0.00122	0.08251	0.11028
5	0.00000	0.00014	0.02960	0.08104
6	0.00000	0.00002	0.01062	0.05955
7	0.00000	0.00000	0.00381	0.04376
8	0.00000	0.00000	0.00137	0.03216
9	0.00000	0.00000	0.00049	0.02363
10	0.00000	0.00000	0.00006	0.01737

Table 2

In the Table 3 the probabilities for the switches on the last stage are given. All requests are generated with equal average input rate.

	average rate			
s	0.5	1	1.5	1.9
0	0.50000	0.02500	0.00000	0.00000
1	0.38889	0.07500	0.00000	0.00000
2	0.09877	0.10000	0.00000	0.00000
3	0.01097	0.10000	0.00000	0.00000
4	0.00122	0.10000	0.00000	0.00000
5	0.00014	0.10000	0.00000	0.00000
6	0.00002	0.10000	0.00014	0.00000
7	0.00000	0.10000	0.00125	0.00000
8	0.00000	0.10000	0.01117	0.00008
9	0.00000	0.10000	0.09958	0.00897
10	0.00000	0.10000	0.88784	0.99094

Table 3

As we can see from the tables it is obvious that as the input average rates grow to the maximum output rate, which is 2, the buffers are more and more full, so also grows the number of rejected requests.

Acknowledgement

This work was partly supported by Iskra Delta Computers.

References

[1] Rettberg R., Thomas R., Contention is no obstacle to shared-memory multiprocessing, CACM, 29(1986), 12.

[2] Patel J.H., Performance Of Processor-Memory Interconnection for Multiprocessors, IEEE Trans. on Comp., Vol. C-30 (1981), 10.

[3]Kruskal C.P., Snir M., The Performance Of Multistage Interconnection Networks for Multiprocessors, IEEE Trans. on Comp., Vol. C-32 (1983), 12.

[4]Dias D.M., Jump J.R., Analysis and Simulation of Buffered Delta Networks, IEEE Trans. on Comp., Vol. C-30 (1981), 4.

[5]Thanawastien S., Nelson V.P., Interference Analysis of Shuffle/Exchange Networks, IEEE Trans. on Comp., Vol. C- 30(1981), 8.

[6]Brajak P., Designing a reconfigurable intelegent memory module (RIMM) for Performance Enhancement to Large Scale, General Purpose Parallel Processor, Informatica 11(1987), 1.

[7]Gelenbe E., Mitrani I. Analysis and Synthesis of Computer Systems, Academic Press, 1980 London.

UDK 519.72

Anton P. Železnikar
Iskra Delta, Ljubljana

Abstract. This essay is the continuation of the previously published 'Information Determinations I' (Železnikar, IDI) and 'Principles of Information' (Železnikar, POI) and investigates some additional higher informational forms and informational processes. In respect of determination from the informational point of view, it deals with creativity, language, writing, reading, utterance, hearing, philosophy, ideology, dialectic, science, technology, aesthetics, ethics, rhythm, harmony, art, literature, crisis, capability, incapability, functions of mind, information systems, social systems, and artificial intelligence. As close as possible, these informational forms and informational processes are investigated through the so-called informational principles of spontaneity, counter-information, Informing, arising, embedding, counter-informing, circularity, recurrence, parallelnes, and sequentialness of information. The choice of very divergent informational forms (for instance, creativity, ideology, technology, crisis, etc.) and their informational investigation shows appropriateness and suitability of informational comprehension of different subjects.

情報の諸限定 II

Anton P. Železnikar
Iskra Delta, Ljubljana

要約. このエッセイは以前発表された「情報の限定 I」("Information Determinations I": Železnikar, IDI), そして「情報の諸原理」("Principles of Information": Železnikar, POI) という二つのエッセイの続きである. 本エッセイではさらにいくつかの高度な, 情報における形式的なおよびプロセッス的な側面を考察する. 情報的な立場から見た限定において, 創造性, 言語, 書くこと, 読むこと, 発話, 聞くこと, 哲学, イデオロギ, 弁証法, 科学, 技術, 美学, 倫理学, リズム, 調和音, 芸術, 文学, クライシス, 能力, 不能力, 知能の諸機能, 情報の諸システム, 社会の諸システム, 人工的情報プロセスなどが論じられている. これらの情報における形式的なおよびプロセス的な諸側面はできるだけ厳密に, 自発性, 反・情報, 情報提供 (informing), 出現 (arising), 埋め込み, 反・情報提供 (counter informing), 循環論法, 再発, そして情報の平行性および順序性といういわゆる情報的諸原理を用いて究明される. このような, 極めて異質的な情報における諸形式 (例えば創造性, イデオロギ, 技術, クライシス等) の選択, そしてこれらの諸側面の情報的観点からの究明の結果によって, 諸問題の情報的観点からの捉えかたの妥当性が示されていると考える.

Keywords. Aesthetics, art, artificial intelligence, asocial, association, autopoietic, beauty, bisociation, capability, circular arising-embedding, circular-spontaneous information, coding, cognition, cognitive psychology, communication, cortical process, counter-Informing, creativity, creative Informing, crisis, culture, cynicism, dialectic, dialictical principles, displacement of meaning, ethics, expertise, formalization, functions of mind, harmony, hearing, hermeneutics, hermeneutic-semantic translation, humor, ideology, incapability, information, information processing, information system, informational arising, informational circularity, informational embedding, informational filtering, informational modulation, informational noise, informational path, informational paralleiness, informational phenomenology, informational spontaneity, informational transformation, Informing, intelligence, inventiveness, knowledge, language, language generation, language understanding, linguistic, literature, metainformation, metaphoric, metaphysics, methodology, mind, natural language, periodic, philisophy, problem solving, reverse information, rhythm, science, semantics, sensory information, social action, social interaction, social system, speaking, storytelling, sub-information, technology, thinking, uttering, value, work of art, world-view, writing.

10

10.1. Creativity

> Creativity is not nearly as mysterious as the genius view leads us to believe, but neither is it as trivial as the behaviorist view claims. ... Creative thinking must remain mysterious and unknownable. ... In the case of creative thinking, the myths explain the phenomena, the creative products of artists and scientists, by emphasizing unconscious thought processes, far-ranging leaps of insight, and some unique personal characteristics. ... what we believe about creativity is not based on hard data but is more or less folklore ... The creative capacity that the ancient Greeks assigned to the mythical gods has in our era been assigned to the unconscious and to other exotic processes.
>
> (R. W. Weisberg, CRY, 3)

Creativity is one of the most regular properties of information. Creativity as Informing concerns spontaneous arising, embedding, and counter-Informing in an informationally circular manner. Information is creative in its circularly shaped arising, embedding, and Counter-Informing. Creativity is semantically and hermeneutically closest to the Greek 'poiesis' and Latin 'cogitatio'. Informing of information is creative, whereas counter-information is the result of informational creation.

Creativity is circularly spontaneous process, which is field-directed, domain-characteristically or field-concerning oriented complexity of mutually dependent and circularly connected informational processes. Creativity is individually structured and organized domain-subjected informational activity which results in creative behavior (thinking, handling, happening).

In comparison to intelligence, creativity is a highly parallel informational process which integrates various intelligent processes, and has the capacity to inform over and above different intelligent activities. Creativity can influence the arising of various intelligent processes by itself. Therefore it is similar to individual expertise, where expertise is understood as the highest possible state of individual capability in a specific domain or discipline. Individual expertise is to be understood as the highest point of a particular intelligence where expertise is achieved by passing through at least five developmental stages of the performer: novice, advanced beginner, competence, proficiency, and expertise (Dreyfus, Dreyfus, MOM). In this way the performer achieves the state of expertise through learning, instruction, experience, investigation, and counter-Informing. Creativity is grounded in 'self', through individual expertising, taste, and origin. Creativity is no less than individually expressed metaphysics of a being, a processing within total information of a living being.

In this respect, creativity is more or less a normal informational way of circular-spontaneously informational arising-embedding (in the human brain), grounded in a specific informational realm (background, field, domain, discipline, skill, art, profession). From the mythical point of view, creativity manifests and roots in the mind of a genius in the form of creative thinking and creative behavior.

Although creativity is understood as a particular and characteristic informational processing of a living being in general, there is no reason to understand it also as an informational phenomenology in non-living or cosmic space. Theoretically, creativity can be divided in several arising-embedding (parallel, parallel-serial, and sequential) informational subprocesses such as (according to G. Wallas, AOT) preparation, incubation, illumination, and verification of the created subjects (CRY).

Creativity involves novel combinations of information and observation, investigation, and recognition of these combinations. Where contrasted to association, the combining of unrelated information is termed bisociation. At this point, a question arises, whether an unrelated information exists at all? Creative information informs by connecting or bisociating an information in which some informational relations are not evident. It arises and comes into existence by this combinational informing. In this process, connective information of previously unrelated informational forms and processes becomes regular information. And this is the act of creation.

The comprehension of designing, planning, and discovering of procedures, by which problems will be solved or objectives achieved leads to the notion of creativity. Most of the functions in the informational and process-generating hierarchy of the brain are learned. Is creativity merely an illusion which turns operational through informational recurrence of combining and discovering information, this is, through recursive procedure for discovering procedures? What kind of information do we have in mind when something new is created? Obviously, some information is taken from somewhere and then this information is processed by another information into a new information giving it a new name or interpreting it anew. Known information is taken and changed into another, so that this process guarantees occurrence of a new information.

Real creativity when inventing a completely new information from the informational nothing is considered not possible. Information can only arise from given information. The question comes from whether a complete creative act could be recognized at all. Information of a being simply cannot be related to sensory input patterns which are entirely different from patterns known up to now, from patterns memorized or presented as cortical processes. Such sensory informational patterns will be received as informational noise without meaning or sense, as strange information, or as possible hostile information. This is the way of autopoietic informing in which a being's total information is molded when information processing is taking place in the living substance.

Philosophically, every living being is creative in its entire behavior because information can never be repeated in an exact manner. However, this informational change is not understood to be creative if the arisen information is not essentially shifted, bent, inflected, thus becoming different from averagely changed information.

Creative Informing is characterized by capability of hypotheses selection, generating realistic expectations, and by precise processing and evaluation in accord with some given rules. In this manner, creative Informing

is the art of a complex objective-oriented behavior. Most creative acts need habitual observation, investigation and cognition, and yet unusual selection of voluntary objectives.

The informational value of creativity occurs in its rareness in the function of informational process and in informational (biological, life-concerning) advantages, which it carries. Its rareness is a consequence of non-creative information processing in human mind. Creativity is oriented into advantage, usefulness, benefit, favourableness, and privilege by discovering new information which is useful for the being and its population.

As a case, creativity also offers the possibility of learning the inventiveness and that of solving problems. A human being learns to be creative in a similar way as it learns adopting of several patterns of problem solving. However, creativity is also a coming, eruptive, and dominant information process, which carries the power of new cognition and development of inventive thinking. No matter how, creativity remains a regular, circular arising-embedding informational process.

According to Maturana (AAC, xxviii), it is important to stress that a creative being, which is an observer, an investigator, and a recognizer, always is potentially antisocial. A being can simultaneously be a member of several social systems (a family, fraternity, club, clan, party, religion, army, race, tribe, nation), however, as a creative being it can create a metadomain from which it can see its participation in social systems that it integrates. In this way, through observation, investigation, and cognition, through a being's counter-informing, a creative being is standing behaviorally outside of social systems and may find them contradictory, controversial, so, that it may undergo social interactions that do not confirm these social systems.

10.2. Language

> ... in thinking Being comes to language. Language is the house of Being. In its home man dwells. Those who think and those who create with words are the guardians of this home.
>
> (M. Heidegger, BW, 193)

Language as phenomenology of thinking and behavior (speaking, hearing, writing, communicating, etc.) is at the same time a higher informational form and informational process by which also the complex realm of informational phenomenology can be governed arbitrarily. Language is founded in the capability of generating and comprehending regular and irregular (new) sequences of phonemes, words, and thoughts. The coding of arising information can be achieved by different linguistic sequences.

Natural language as informational expression and as informational reception is a living, socially active and developing informational process, which carries and generates a variable coding (interpreting) of the meaning. Linguistic processing is a fair example of information processing concerning the circular arising-embedding principle of symbolic meaning and formal expression. Arbitrary informational phenomena can be described and expressed in linguistic forms and performed by linguistic processes. However, specific portions of informational phenomenology cannot be adequately or sufficiently precisely expressed by means of natural language. Such are several forms of sensory information (concerning sight, hearing, smell, taste, or tactility), experience and other information. In this respect, language remains particular informational form, i. e. particular information, or sub-information.

In spite of the fact that it is possible to express information by language, it is not possible to express information in the way as informational forms and informational processes appear in a living substance. Expressing information always involves informational transformation, whereas through language, thinking, a notion, a form, or a process are translated from their original informational appearance in a specifically linguistic manner. Under language an external and environmental, contextual, and grammatical-semantically determined natural language can be understood. The internal, intellectual, mind-concerning information includes imaginative, picture-like, acoustic, extraordinarily shaped, individualized properties which are determined by internal, individual informational patterns, connectives, relations, processes, and as such cannot be expressed immediately and adequately by means of a natural language. However, the use of language is a form of behavior, a means of social action, which are characteristically related with thinking and development of the possibility for future expression of information.

Language is destined for specific social interactions, such as awarding, refusal of punishment, or achieving success in the social hierarchy. Linguistic behavior enables informing, counter-informing, communication, receiving of reverse information and counter-information from the environment. This kind of behavior is primarily a kind of social interaction, populational culture, and valid behavioral forms, which concerns highly intelligent and creative being, its conscious and intelligent information.

Language generation and language cognition also root in informational processes of rhythm, periodicity, and patterns of harmony which appear on different informational levels and in different social situations. Already very simple, however well-structured or metaphoric linguistic entities can influence the creative power and cognition of language generation and language understanding.

Language reflects a being's internal information, informing of this information in itself. Creative, meditative, strategic, contemplative and other kind of thinking is internally arising and linguistically structured information. It is expressed and understood in the realm of a being's total information by means of its natural language. Language possesses its informational depth and penetrates as informational phenomenology into arbitrary depth of philosophy and poetry. Through language it is thought about information and language, up to arbitrarily possible abstract informational levels. In some examples, language reaches the power of information by which it is coming into existence in an information-processing manner. A being with language capacity and with its in-itself-language is continually speaking. It gets on the way of this language through acoustic articulation, through listening to other speaking beings, and through thinking by means of linguistic categories. Internal speech and being's (voice-articulated) speech are

sprouting internally, developing language, and processing it. Being's information processing from one kind of complexity into another, from less to more complex and vice versa, is setting on the way to a new meaning and new symbolism. Language is the home of informational abstractness, symbolics, meaning, semantics, hermeneutics, cognition, understanding, and other linguistic-informational modeling and behavior.

Language means speaking of language and its processing in itself. Information means circular arising-embedding of information. Language is circular arising-embedding of language. Several informational forms of a being are language-characteristic, verbally shaped, or processed by linguistic means on different informational levels. These forms can have language-characteristic codes, abstraction, symbolism, meaning, and understanding. Mastering a language means emphasizing lingual abstractness and recurrence, developing lingual thinking and counter-informational processing up to the highest level of behavior-generative hierarchy. It involves exploration of the arising of counter-language and lingual invention, where new semantical and hermeneutical categories are constructed. Speaking in verbal language means the arising of language-shaped information, the development of linguistic information processing. Language develops Being of language in the direction of higher or lower diversity, variety, complexity, and informational performance. Language is at-the-top of a being's informationalness. It is a tool, expression, cause, and consequence of intelligence, the being's creativity. In the cortex of a human being, language is represented as a form of particularly distributed information processing which is linguistically coded, abstracted, and symbolized.

To some extent, language is informational self-sameness when it speaks about information, supposing that the information is comprehended by lingual categories. Language develops information about information. Language develops information about language when language speaks language and speaks about language. In this case, the way to information is on the way to language where a being's informational processes of a being are verbal. The way to language by language is also the way to the being's information by information.

Language processing is an essential faculty of being's informationalness, which is distributed, essentially rhythmic, harmonic, filtered, and modulated by other informational processes (for instance, emotions) of mind. A being is a developing informational, lingual system, where language processing is embedded into the being's informationalness. Language processing is only one of possible modes of information processing. A being's imagination can be characteristically verbal, processed linguistically. On several informational levels language processing can be semantically and hermeneutically evened to a being's informationalness. Language is an informational model which models information, and simultaneously represents and processes information by linguistic means. In a similar way, as information is on the way to information by counter-information, language is on the way to language by counter-language, which is the arising meaning, expression (concerning linguistic form), and understanding.

Within "classical" cognitive psychology (CPY), language is defined by notions of semanticity, arbitrariness, discreteness, displacement, productivity, iteration, and recursion (CPY, 322). One of the basic questions of language remains how linguistic entities, units, and fragments are coming into understanding, how the processes of apprehension and comprehension are arising, developing, coming into existence within a being and in a being's population (environment, culture). In this respect, the approach to understanding within the being's total information is hermeneutic, focused upon internal information and on the internal process of speaking. In contrast to hermeneutic information, the approach to understanding in populational environment or culture is semantic, informing linguistic facts externally, by external processes of speaking cultural languages.

10.3. Writing, Reading, Utterance, and Hearing (in verbal Language)

> ... Language is not a defined realm of the speakable, over against which other realms that are unspeakable might stand. Rather, language is all-encompassing. There is nothing that is fundamentally excluded from being said, to the extent that our act of meaning intends it.
>
> (H.-G. Gadamer, PHH, 67)

On the one hand the informational forms and informational processes, like writing, reading, uttering, and hearing all concern language, i. e. live linguistic activity, thus covering acts of internal and external speaking. On the other hand the said activities also involve interaction and reaction of speakers (writers) and hearers (readers). Writing, reading, uttering, and hearing are social activities which concern social action of living beings, social interactions within beings' populations.

Writing a text in a (natural, formal, professional) language concerns creative internal shaping of meaning, planning a sequence of meaning as a hermeneutic form and afterward transforming this hermeneutic skeleton into an adequately grammatically and semantically shaped result, displayed in an appropriate external, linguistic, or cultural form. This process of hermeneutic-semantic translation causes a displacement of meaning between the initial hermeneutic plan and the final semantic result. Displacement of meaning is the difference occurring between a being's total information (its metaphysical presentations) and external language-ruled information. The being's metaphysical fragments, which represent the hermeneutically shaped content of guiding ideas are through writing carefully translated into linguistic (grammatical and semantic) entities. However, this is only the first step of the writing iteration.

Linguistically written text is now presented for further creative improvement of its meaning and its form. Semantics of the written text can now be analyzed, observed, investigated, and recognized by reading the text. Through these processes, the text may be transformed into hermeneutics, giving reason to change, complete, add, or reduce semantics of the given text for the second time. According to this, changes, insertions, additions, or dropping of some textual fragments may occur. Such

improving iterations of meaning and form can continue until writer is getting satisfied with the final result. One has to bear in mind that the informational process of writing concerns planning of the meaning on the hermeneutical level, translating this meaning into grammatically written text on semantic level, reading this text by translating it on hermeneutic level, correcting the obtained hermeneutics, translating hermeneutic form into semantic one, etc. In this way, writing and correction (improvement) is an iterative informational process of text creation.

Writing and writer's skill is a complex informational goal-directed activity, a particular intelligence, which was acquired through laborious learning, repeated exercise, self-correction, self-motivation, sensitivity, perception, cognition, mastering of meaning, style, grammar, composition, expression of writer's thinking, imagination, and factual knowledge. However, no one text is written in such a form that it would not call for improvement, in order to increase its linguistic and informational valuableness. In this context, informational valuableness has the meaning of the power which is to generate intended informing in the reading auditory, to influence this auditory informationally (ideologically).

Reading a text is an informational process of transforming the semantics of the text into hermeneutics of the reader. In the process of reading, the linguistic information informs a reader's total information by counter-information which results into hermeneutics of the given text. In process of understanding, the reader can repeat reading of problematic passages until he obtains the information that is hermeneutically satisfactory. In the process of reading, the reader's information informs and counter-informs, so the reader is getting its own new information, its counter-hermeneutics. By reading, an informationally sensitive being can be informed and counter-informed, for he performs its own informing concerning the semantics of a text. Of course he develops it in a new hermeneutical direction, where the "coming-to-understand" arises. Reading is the individualized process of informing where text is information, giving clues for the context of a being's entire information.

Utterance (speech) is the act of speaking (voice articulation) in the context of social interaction (hearing). Internal speech (utterance within itself) has a form of dialog, of a social interaction with a being itself, with imagined speakers and hearers. In speech and in hearing of the speech, hermeneutic interaction occurs among speaker and hearers. In this social interaction, speech interferes and depends from the reaction of hearers.

In speech, the speaker informs in several ways: while speaking he plans the meaning of speech fragments as his hermeneutics, articulates this hermeneutics in a linguistically appropriate form through several informational transformations (hermeneutic-semantic, semantic-grammatical, linguistic-articulatory, etc.), receives reactions (linguistic, gestural, mimic, sighing, etc.) of hearers, and interacts with adaptation and correction of his speech activity. In this way, hearers can influence the planned utterance of the speaker, signalling and communicating their hermeneutic agreement and disagreement to the speaker. Only a poorly structured and organized speech lacks such a social interaction of speaker and hearers.

Speech gives evidence how it is important that hermeneutic backgrounds of speaker and hearers are evened to the appropriate extent and must not be to far away. If not, the speech will be received as informational noise without a wishful speaking-hearing interaction. If hermeneutic backgrounds of hearer are very different, speaker has to use a sufficiently bright semantics, which is externally valid for a broad or mixed auditory; a 'good' speaker will use general accepted cultural forms of utterance and convenient grammatical and semantic forms.

11

11.1. Philosophy

> ... If philosophy is the scientific construction of a world-view, then the distinction between "scientific philosophy" and "philosophy as world-view" vanishes. The two together constitute the essence of philosophy, so that what is really emphasized ultimately is the task of the world-view.
>
> (M. Heidegger, BPP, 7)

Philosophy is a linguistically based (usually literary) form and a linguistically based process of particular (philosophical) information with a specific, uttered, written, esseyistically structured, logically organized, and thought-free orientation. It is dedicated to some selected, philosophically characteristic objects (questions, problems). Each philosophy has its own system of information and its own way of informing. The orientation of a philosophy is its semantical (objectively selective), comprehensional (philosophically systematical observation, investigation, and cognition), and stylistical (characteristically organized and formalized) task. In this way, philosophy is a culture-memorizing, a being-investigating, and a population-concerning information.

In philosophy, poetry, literature, and ideology, informational processes can come into existence informationally and counter-informationally through their particular respectability. They are established through bethinking, sensing, grasping, and comprehending occurrences, which are separated from the imperative of necessity and can be arbitrarily informationally perverted. The informational processes are variably playable, truant, magically oriented into the contrariness of the real, unreal, rational, irrational, free, forced; they are involved into the play of themes, images, sounds, colors, notions, events, models, into self-bethinking as a response concerning similar environmental absurdity. Furthermore, they are characterized by the autoreflection of counter-reflection. Is this kind of information autopathological or is it a healthful discourse which is also self-contrary, evidently autoabsurd, autocorrecting, in many cases, however, only mataphoric, paraphrasing, poetically expressive, and philosophical?

Since the ancient understanding, philosophy had been a friendly, methodical informing, reasoning, wisdom, and basic cognition without a purpose. It could have been explicated as an information theory in itself. Throughout the history and cultural development, philosophy

became more and more formalized, structured, purposely constitutional, specialized, scientized, artistic, and ideological. Eventually it turned into a systematic, purposeful, utilitaristic, methodological reasoning about being, nature, cognition, mind, life, history, and about itself alone. From the original, friendly reasoning it developed also into human hostile, external, forced thinking. In the history of mankind, philosophical orientations were confronted, prosecuted, and prohibited as information of unbound thinking. In this respect philosophy appears to be only a particular, cognitively oriented historical phenomenon of information and counter-information.

By its informational nature, philosophy enters the realm of abstractness and theory, where modeling, changing, systematizing, and artificial thinking are exercised, due to inherent philosophical capacity of performance. In this capacity lies its developmental-progressing and thinking-complex value. Philosophy offers possibilities of divergent and heterogeneous discourse about arbitrary questions of being, existence, essence, and within these questions also about culture, science, technology, economy, democracy, applying philosophical modeling and prevision of answers. Information which is generated by philosophy in this way, can be transformed into counter-information down to the arbitrary cognitive depths. In this respect, philosophical information basically contributes to cognitive informational processes in science, technology, and management.

Philosophy is the principal informational activity of philosophers (from Plato to Heidegger) and their writings are informational forms of philosophy. According to A. Flew (DPH), "... philosophy is a matter of standing back a little from the ephemeral urgencies to take an aphoristic overview that usually embraces both value-commitments and beliefs about the general nature of things." Responsive philosophy can always be understood as a being's therapeutic and developmental counter-informational phenomenon of its creators, producers, and followers. Philosophies can inform therapeutically and developmentally against pathological informational syndromes. A sound being or a sound population understands ideology as an essentially reduced, specialized information of already acute, mass incapability of brains (illness, lesion, injury, blocking, excessive artificial or natural excitation), whose consequences are pathological, without expected therapeutic efficacy, descending into pathologic compensation and momentary balance of survival.

Philosophy of information is nothing more than a particular information about information, an expressively recurrent informational process, counter-information about information, which broadens and innovates comprehension and informationalness of information. In any case, philosophy is only a preinformational process of general informational arising, since it is only a particular form of informational and counter-informational regularity, naturalness, reasonability, cognitivity, and progressiveness.

For Martin Heidegger (BAT, 487), '... philosophy "is universal phenomenological ontology, and takes its departure from the hermeneutic of Dasein, which, as an analytic of existence, has made fast the guiding-line for all philosophical inquiry at the point where it arises and to which it returns." ...' Counter-information especially embraces philosophy as sufficiently complex informational process, as sufficiently broadened investigation of phenomenology of beings living in the beings' environments. Information departures from explaining life processes or survival, from analytics of survival, from existence of living and non-living, from information itself and via counter-information also returns to information.

Basic questions of philosophical ontology and anthropology cannot surpass informational essence, informational Being of a being, and informational liveliness which accompany beings from their conception to their death. A being's informational origins and sinkages are autopoietically unlimited in thinking and in behavior of beings.

11.2. Ideology

> To learn ideology means to step into ideological circle. To understand ideology means to observe informationally the ideological behavior.

Hitherto, it is not possible to give an exhaustive, sufficiently determined and informationally well-embedded notion of characteristic ideological forms and processes. In this section, only a beginning into the action of opening of informational understanding of ideology will be given, mostly through putting questions being on the way to ideology. What is ideology as an informational form and informational process in a being, population, or society? Which informational principles does it follow? What is idelogy by itself as an informational principle? In what way is it characterized through informational arising, embedding, counter-informing, and circulation?

Ideology will be the label for a characteristically structured, organized, and semantically oriented informational form or informational process. In general, ideology as information violates the most fundamental principle of information - the principle of spontaneity (Zeleznikar, POI). Ideology concerns the being's everyday experience through its unpleasant, disagreeable, intruding, officious, and manipulative influence. Ideology is essentially distinctive from the natural being's metaphysics or from a freely arising population's ontology. It is very close to the informational artifact or to artificial information, which has its own way of informational propagation and circulation. Ideology is, for example, the informational product of an archaic culture, of a civil-backward, hindered population, which is marked by antifreedom. Ideology stands for ethically, morally, scientifically, and philosophically bound learning, doctrine, it is is an evident example of non-spontaneous, strange, fictitious, objectively false, and inappropriate information of the modern epoch.

In general, information is a free, spontaneous, formally unlimited and in no way formalized phenomenon. Every informational formalization excludes some substantial meaning from an informational domain by introduction of a specifically formalized language (ideological, philosophical, scientific) in a being's realm of thinking. Ideology is characteristically formalized, oriented and limitedly cultural informational form of population or individual

being. It may be treated as informational process concerning thinking and behavior of beings. From this point of view, the question of ideology versus information can be questioned upon if it is to become the topic of this section. This questioning will produce interrogated and newly arising questions which will all recurrently influence making questioning about the essence of ideology.

It seems that in the realm of a being's thinking, ideology qua information is formalized in a specific way. Through this thinking, this formalization is extended also into the realm of the behavior of human beings and their culture. Each formalization, which also holds true for the informational one, is restricted in the spontaneous informational arising, embedding, counter-informing, and circulating (recurring, paralleling, and sequencing). In some cases, this ideological formalization is presented culturally in a linguistically oriented, written, spoken, or behavioral form as a process of traditional, fraternal, clanish, or revolutionary (radically new) orientation. Ideology is developing out from its magic, thus conducting and managing a body of ideas, embedded in the being's metaphysics and/or in the population's ontology.

Ideology is an ideally oriented metaphysics or ontology, possessing its conducting kernel. In this respect, ideology has its informational grounding, which can be mutually different to the extremes. Extremes given actually represent a list of synonymous antonyms, such as:

axiomatic	... dogmatic,
magic	... indoctrinating,
suicidal	... fratricidal,
gentilistic	... homicidal,
nihilistic	... positivistic,
rationalistic	... behavioristic,
logical	... ideological,
hard-scientific	... artistic,
caste-feeling	... class-hostile,
tribe-liable	... individual,

etc. Ideology has its own approved system of rules for the so-called ideological deduction, induction, argumentation, and inference of ideological truth and untruth (ideologically a false information).

Various ideological forms and ideological processes can be understood, for example, as particular forms of informationism, informism, informativism, informatism, informatizism, and informationalism (described in paragraphs 1.4, 2.3, 3.2, 4.3, 5.3, and 6.2). Linguistics (or formalization) of an ideology can be determined more exhaustively, restrictively, paternally, radically, or when led to the extreme in the form of a totalitarian ideology. It is more indefinite, pragmatical, or ideological-independent when concerned with subtle, concealed, underground, or apparently non-manipulative ideology. As a totalitarian code system, and through this as a total form of a being's thinking, ideology governs a being's behavior, not only through a being's restraining but also through the ideological forecasting as behavior it is ideologically expected. In this way, any ideology is being vitally approved or verified as a system of truth and through this, it compells beings to perform ideological behavior.

However, any ideology is only an informational form or informational process, so that the arising principle or counter-informing of ideology qua information is preserved. In the informational context of a being's total information, ideology is appearing as a hardly ever informationally resolved counter-information producing information for everyday life. The arising of information in an ideological environment is the governing ideology itself. Further, it seems that ideology, which in some broader information-embedded specific information is embedded in its own grounding. In such a manner the being is self-sufficient, self-producing, self-preserving, self-referencing, and autopoietic in an ideologically specific manner. With regard to normal informational arising inside being's thinking, ideology appears as a characteristic counter-information, by which the environment is exerting pressure on a being in the direction of an ideologically ordered, oriented, non-diverse, non-alternative, non-spontaneous thinking. Through this it keeps the being's ideological, non-natural, artificial, counter-feint, pathogenic, imitative, mimicking, or generally deceiving behavior. Ideology is an environmentally supported movement of information which pertains to population and society in general. Obviously, ideology is informing in a specific, informationally and counter-informationally blocking way.

Ideology is self-perpetual, gaining its own orientation in amplification, maintenance, and in its perpetuity. This is possible because ideology is concerned with counter-information, which is a trivial (ideologically well-structured) form of counter-information, the respective ideological realm itself. In this context, the self-perpetuity has the meaning of ideological, of ideologically true, of non-spontaneous arising, which is governed by the perpetuity itself. Evidently, ideology is non-intelligent, non-creative, because in ideology, intelligence or creativity as a phenomenon of informational throwness is superfluous, not needed as an informational process.

In many cases, the so-called traditional background is ideological to some extent too and rests on a being's (ontic) metaphysics or on a population's ontology. A traditional orientation as informational background is cultural, in fact ontically outward, however, a being feels it as it it were its own, as to be in a being dwelling orientation. Therefore, a being defends the tradition as its own information, for which it feels as belonging to itself, as it having been co-produced by a being itself.

How is it possible to characterize ideology qua information? Albeit, the following list of characterizations is not completed since it only supports further questioning applied to ideology: ideology as information is non-autonomous, alternatively blocked, ideologically controlled, filtered, and modulated in its arising or in its multi-directional-arising, ideologically structured and organized, topologically unified, iterating in itself. It is further ideologically autopoietic, but generally non-poietic, dogmatically embedded and ideologically stable, with a being's influencing and supervising. Ideology at the same time covers ideal, axiomatic, simplifying, and utopian features. It is: ideologically gaining, amplifying, and maintaining; informationally reduced; semantically hidden, converted or transformed; informationally or spontaneously non-parallel, impractically oriented, unfitted for life (inanimate, lifeless, immature, unhealthy), etc. Ideology is a body of characteristically limited information (ideas) of a being, of a population, or of a culture representing assertions, theories, and aims that constitute an ideological program, directed thinking and

uniform behavior. Ideology is structured and organized hierarchically. Its highest, supervisory level is the ideological kernel, which might incorporate different informational forms as hatred, hate, hostility, fear, anxiety, destructiveness or destructive irrationality, archaic or fraternal terror, suicide, fratricide, homicide, genocide, envy, revengefulness, incapability, greed, avidity, covetousness, etc. The kernel of an ideology is represented by the so-called avant-garde of a given ideology, where this avant-garde is evidently metaphysically blinded, emotionally founded, backward, and possibly biologically unsuitable and informationally (cortically) pathogenic.

Ideology is characteristically periodic and in itself closed informational phenomenon, which repeats its patterns of thinking and behavior in an exclusive, one-way, one-sided, biassed, partial, unjust, unilateral, lop-sided, top-heavy, single-track, unitarian, uniform, however rhythmic, harmonic, and thematic manner. Especially, because of its clearly forecasting possibilities, ideology is performing beings' harmonic, informing also in a killing, apathetic, indifferent, narcotic, subduing, restrained, consciously excluded, and repressive way. All these ways of informational filtering and modulation give the ideology a fictitious, hypocritical, deceptive, and garbled impression of survival safety within its own protection. Thus, ideology is established in self-protective and development-stable way. In the human brain, ideology is blocking, preventing, obstructing, frustrating, and artificially separating the left hemispherical informational activity from the right one, thus destroying the left-right informational parallelism, connectedness, mutuality, and harmony.

Ideology seems to offer mentally and behavioristically passive encouragement and turns out on the other hand to be connected with an indefinite feeling of delight, safety, rewarding, efficiency, surviving easiness, and self-motivation which perfectly suits a properly incapable or disabled individual. Ideological feeling changes and varies the internal world model in the direction of monochromic beliefs, faith, and expectations, which are being harmonically realized and approved, although not using the extrapolation, retrospection, and introspection concerning the future possiblities or rewarding plausible forecasts for the future survival. Ideological feeling and will determinating ideological behavior are conditioned with the internal ideological state of a being's central nervous system and with the ideological environment or culture, which generates and modifies this internal state. The ideological will selects ideological behavior, its patterns and generators, which are activated in an ideological environment and in social hierarchy. They incorporate ideological habits and customs (relations, ceremonies, myths, taboos) as well as strategies and tactics of ideological behavior (ideological speaking, faithfulness-and-obedience approving, uniformness, ideological enthusiasm, respect for ideologically prohibited and repressive justness). Ideological behavior makes use of ideological expertness (fluency of ideological feeling, elucidating, simplifying, scientific contemplation, and manipulation), and ideological reflexes (adaptability, submission, docility, humility, servility).

Patterns and generators of ideological behavior start and maintain the world model, the world-outlook of an ideology-subjected being. This model is based on ideological beliefs, mindfulness, and remembering. An ideological world model, which is a wishful and expected image and imagination of an ideology subjected environment, includes the categories of spiritual, social, and physical entities. The spirit of an ideology is its idolatry, guiding ideas, ideological doctrine of salvation, an essential theory of the given ideology including entire ideological abstraction, symbolism, language, sense, and consecration. The sociability of an ideology roots in an ideological framework, which is ideologically correct, and provides ideologically equitable relations. These relations are reflected from the ideological environment only. The physics of such an informational model is, for example, dialectically ideology-materialized or ideology-idealized physical phenomenology, where ideology interferes in a totalitarian manner with thinking and with modeling of the physical world.

Sensory processors which are receiving information and sensations from the ideological environment, are under the filtering and modulation control of the ideological world model. These ideologically abstract sensations are in a perfect, prodigious, rhythmic, and harmonic agreement with beliefs, expectations, and forecasts of the world inner structure and generate feelings, emotions, and values (in fact non-values) of contentment, delight, happiness, safety, and success, but also their controversies. These feelings which are virtually satisfactory do control the will for a rhythmic and harmonic selection of the behavior according to ideological circumstances. In this way, the ideological circle of happiness, safety and their controversies is joined into circulation, in a loop of ideological feeling. In this respect, ideology is an evident example of a particularly composite informational circularity (recurrence, parallelness, and serialism).

The objective of an ideology is to suppress the spontaneous arising and spontaneous embedding of counter-information by which the ideology is observed, examined, and recognized. In the framework of an ideology, this counter-information is promulgated as an information of the hostile environment, because it causes the decay of ideological kernel and through this it results in decay downfall of ideology itself. For an ideology, every spontaneous arising of ideology-related or even ideology-non-related counter-information is dangerous and must stay under the systematical control of ideology itself.

In the realm of an ideology, the entire phenomenology, changing of the world and of information qua vitalization, qua renewing is comprehended only in the form of vaccination of the existing, governing ideology. It is performed by usage of ideologically approved rules of thinking and behavior. In this typical looping and ideological harmony, ideology meets its informational regression, extemporization, blindness, diminishing, and decline of its organism, functionality, structure, and organization. Sooner or later, every ideology must confront with consequences of its artificial activity, atrophy, changes of its age, decay, degeneration, and involution. Because ideology qua information does not allow the arising of a spontaneous being's counter-information (natural breakdowns), it remains Being to itself without an essential informational arising and vanishing. In this context, ideology is an unnaturally alive, artificially inhibited information.

As a being's informational process, ideology creates its characteristic world model, which is being supplemented through ideological behavior and also through ideological imagination of a being. Are there two world models in a being, the first one in the form of an authentic imagination and the second one as ideological behavior? What is the model duality created for, besides the being's authenticity? In general, ideology is an imposed outside informational process, belonging to other beings; this process interferes with authentic informational subprocesses. The feeling of intrusiveness is more or less ontically conscious, destined, existential and a being accepts it as an inevitable disturbance, as an indispensability and inconvenience or as a pretense and bother. This informational model duality creates the feeling of threat and fear, which by means of will select a counter-feint, artificial, and forced ideological behavior which is lacking liberty and variety in a spontaneous bethinking. A being selects the permissible behavior through a forbidden imagination. Ideology is a phenomenon of informational contrast, of the absurd of simultaneous affectation and spontaneity, where the spontaneity is imaginative and affectation is behavioral.

A typical informational side-effect of the ideological repression is the so-called ideological cynicism. If a being feels an ideology as a distortion, extortion, punishment, and torture, the being responds to ideology by mockery and revengefulness simultaneously, individually, however population-typical, and expressive, but population-diffused. As the reply to ideology, this consciousness is destructive, untouchable by ideology itself, private, ideologically non-convertible, generally skeptical, illuminated into counter-ideological consciousness, which is falsified, nihilistic, although it is rebelliously and resistantly imposed. This cynical consciousness is in contrast to pressing, to rational imperative, it is not capable to be on the way to this imperative in the context of ideology, because it is a radical irony standing outside of the senses of necessity and survival. An ideological cynic is like a ruler and resembles a subdued, malicious realist. He is like a socially integrated antiassociate, cynically socialized, lost in his own and populational cynicism. Ideological cynicism is a degraded consciousness, being ill from repression, for which it has to accept ideological relations, not believing in them, however being forced to be surrounded by them, and at last, for which it must perform ideologically distorted actions. Idelogical cynicism is an instructive example of a non-spontaneous, informationally repressed, however, individually perverted, protectively reactive behavioral process.

11.3. Dialectic

> Dialectical reasoning is only one of possible forms of counter-information, which could arise by circularly spontaneous informing, counter-informing and embedding of information.

Let now dialectic be illuminated as a particular example of an informational form and an informational process through the following questions: How is dialectic embedded in philosophy and ideology? What is dialectic as a world-view? What is dialectic as information and how can it informationally embed information as information?

The notion of dialectic comes from the Greek 'dialektike tehne' and has the meaning of skill performed in dialog, discussion, or discourse. Its very first meaning concerns intelligence of discourse or art of social interaction in the realm of language. In this sense, at its appearing, dialectic is the label for art of discussion in processes of questioning and answering.

Through history, dialectic as a principle of philosophy and ideology develops from Socratic, Platonic, Aristotelian, Kantian, and Hegelian platform to Marxist platform, that is from discourse, supreme knowledge, reasoning from premises, transcendental reasoning, and logical pattern that thought must follow (thesis, antithesis, synthesis) to the pattern that thought and reality must follow (subjective and objective thinking). In its late development, one of its supreme orientation or world-view marked dialectical materialism becomes a metaphysical doctrine or social ideology which asserts the matter, motion and development of matter as fundamental. An informationally parallel world-view arises as historical materialism, which as information deals with developmental laws of human society and thought. Dialectical materialism opposes idealism stressing that matter is not a product of mind, but that mind is a complex product of matter. In this manner, the dialectical doctrine distinguishes materialism and idealism explaining dialectical principles (rules of dialectical inference) as transformation of quantity into quality, interpretation of opposites, and negation of negation.

How can dialectical methodology be understood informationally? Certainly, dialectical principles of thesis, antithesis, and synthesis can be included in principles of information, counter-information and informational embedding, respectively. Furthermore, dialectic as a movement and development of things and thinking coincides with Informing and counter-Informing, respectively, however, not in information-embracing entirety. Counter-information, for instance, is not only an antithesis, but everything which arises, which comes into existence spontaneously from information. Also, information is not only a thesis which corresponds to a dialectical category of consciousness, but a phenomenon of free informational arising in a being's consciousness and also outside of any consciousness. Informationally, the dialectical circle of thesis-antithesis-synthesis is a part of informational circularity (recurrence, parallelness, sequentialness) of Informing, counter-informing, and informational embedding. In this informational circle, informational arising as circular informational spontaneity is stressed as the basic principle of various informational developments. Dialectical synthesis concerns counter-Informing and embedding simultaneously. In this respect, some essential differences between dialectical and informational approach can be observed, and it may happen that dialectical principles can be broadened into informational ones in an easy, informationally natural way.

Let now information which was developed as a notion of information and a notion illuminated through previous examples in this essay (through various examples of informational forms and informational processes) be thrown into the realm (informational system) of dialectical discourse. Dialectic as dialectically embedded informational system of materialism, which follows its own rules of inference now comes to the conclusion that the proposed notion of information in this essay is

nothing more than idealism and/or metaphysics (this already are the necessary dialectical-ideological etiquettes of triviality of a notion) because it stands outside the correct and known inferential realm, and maybe, cannot be "directly" comprehended through basic materialistic principles (for instance, materialistic particles, materialistic forms and materialistic processes of reasoning). Informationally, metaphysics is the label for the total (entire) information of a being (as determined in section 8.4). Informationally, idealism is a form of philosophical and/or ideological informationalism. Materialistically, idealism is the philosophical counterpart of materialism which does not enter into the materialistic realm as a materialistically legal inferential subsystem. Materialistically, metaphysics opposes dialectic, for metaphysics is a materialistically non-movable, static, idealistic doctrine. In this context, evidently, the exclusion principle of materialistic reasoning takes its philosophical and/or ideological power.

11.4. Science

> ... Heisenberg argues that contemporary thought is endangered by the picture of nature drawn by physics. This danger lies in the fact that the picture is now regarded as an exhaustive account of nature itself, so that science forgets that in its study of nature it is merely studying its own picture. ... Heidegger, however, sees the picturing of nature as a necessary feature of scientific thought. Yet both would agree that so long as science is allowed to speak ex cathedra, other modes of thought remain impossible.
>
> (H. Alderman in HMP, 41-42)

How science is revealed as phenomenon of information and how does scientific information arise within informational process of science?

Science concerns knowledge which is acquired or gathered scientific-methodologically and ordered scientific-systematically. In this respect, science as a discipline concerns its own scientific circle of its own methodology and of its own systematics. Science circularly informs its own, scientifically legal or scientifically permissive information. Scientific informational arising, counter-informing as a scientific contributing, and informational embedding as scientific (ac)knowledge(ment) are scientifically and disciplinary bound to the valid scientific system of truth and untruth. Science is informational process by which the scientific knowledge is produced through criteria of scientifically methodological truths. Knowledge generated by the corresponding science and scientifically generated methodology both represent the so-called scientifically embedded legal information, from which new scientific achievements can be informed and counter-informed.

Knowledge and methodology are the very valid informational foundation (background) of a science. Scientific Informing proceeds from this knowledge base by application of methodological rules. Thus, science becomes a formal system which is scientifically bound by its knowledge and rules of permissive inference (deduction, induction, inferential Informing). Science investigates its truth and its own investigation; by the said methods it deals with discovery of truth, rationality, reliable prediction, generalization from sufficient collection of data, theoretical construction, predictive theory, model, explanation, deduction, induction, knowledge, and scientific methodology, revealing the truth about the hidden objective reality. However, science also deals with its own ideological forms and processes of logic, empiricism, positivism, meaning of new theories, theoretical concepts, and verification. Through its life cycle, science accumulates (enlarges) subjects of its theories, increases its knowledge, suggests theoretical realism, but also imposes doctrines of formalism, instrumentalism, and scientism.

Science does not allow philosophical spontaneity and philosophical abyss, therefore, in many cases, it cannot recognize its factual possibilities and its attainments, when it alludes behind the possible reality. In the framework of its rigorousness and correctness, science has to preserve its characteristic ideology, autocensoring, disciplinarity, and steadiness, to prevent it from the decay and to enable the accumulation of its scientific information.

Although sciences are grounded in knowledge and sciences are merely that what is known, knowledge also exists outside of the sciences. Knowledge is a result of cognition, it is an objectively founded belief about the truth of a proposition, assumption, and consequence. Cognition, which is not only scientific, is an informational process capable of production of knowledge, where knowledge appears as truth. The notion of knowledge contradicts the notions of belief, opinion, and view. Knowledge by itself is an arising, changing, and recurrent information.

11.5. Technology

> ... What has the essence of technology to do with revealing? The answer: everything. For every bringing-forth is grounded in revealing. Bringing-forth, indeed, gathers within itself the four modes of occasioning - causality - and rules them throughout. Within its domain belong end and means, belongs instrumentality. Instrumentality is considered to be the fundamental characteristic of technology. If we inquire, step by step, into what technology, represented as means, actually is, then we shall arrive at revealing. The possibility of all productive manufacturing lies in revealing.
>
> (Heidegger, QCT, 12)

Technology as information is nothing more than a sort of productive presence of beings in themselves and in their environment. In this context, productive presence has the meaning of action, development, destruction, and construction. Technology qua information arises as beings' thinking and behavior for changing themselves and environmental circumstances, destructing and constructing themselves and environment physically (constructively, methodologically) and biologically (culturally, intellectually). In this respect, a machine or even a living being can be understood as a final physical and/or biological construction, whereas methodology (thinking or ideology) can be comprehended as a final cultural and/or intellectual construction (or destruction). However, machines and beings as physical/biological constructions arise through

circles of informational constructions (counter-informing, embedding). Technology is information of artificial and living construction and destruction, of machines as forms and processes, of beings as biological structures and organizations, and of methodologies and minds as forms and processes for achievement of constructive and/or destructive results.

Constructive technological activities can be observed in animal-labor (specific behavioral patterns), for instance, in constructive activities whose products are bird nests, lairs of beasts, beaver lodges and dams, spider webs, honeycombs, ant-hills, cocoons, and other forms which arise through behavioral activity. In constructions of this sort, principles can be observed whose nature can be compared with sensefulness, optimalness, solidness, intentness, suitability, rationality, and imagination. Technological phenomena in microstructure of matter are, for instance, crystals, macromolecules, synthesis of proteins (genetic information of DNA as a program of a machine for synthesis), and at last biological growth, which reveals construction (structure and organization of living).

Roughly, technology as informational phenomenon also materializes and methodizes thought and constructive imagination through specific human behavior which is called labor. The arising technological information influences and changes beings' behavior and simultaneously and continuously develops thinking into the direction of more and more complex technological achievements. On this way of technological development the question comes up if already being-like machines (instruments, processes) can be constructed which could possess intellectual, brain-like, or mind-like properties. Technology as realization of useful machines and methodologies is the developmental way of construction, of technological revealing and creativity, and also a part of strategic thinking and behavior for survival.

12

12.1. Aesthetics

> ... How nature pleases us belongs instead to the context that it stamped and determined by the artistic creativity of a particular time. ... We are justified, therefore, in proceeding from the work of art rather than from natural beauty if we want to define the relation between aesthetics and hermeneutics. ... The language of art is constituted precisely by the fact that it speaks to the self-understanding of every person, and it does this as ever present and by means of its own contemporaneousness. ... Everything depends on how something is said. But this does not mean we should reflect on the means of saying it.
>
> (Gadamer, PHH, 98, 102)

Experiencing, feeling, sensing, observing, investigating, recognizing, comprehending, understanding, living, describing, determining, expressing, thinking beauty and value are only a few informational processes, which can be hardly determined by linguistic thinking and expressed semantically. Do the notions of beauty and value exist hermeneutically, or even ontologically, or are they first of all an emotionally and intuitively conditioned and guided information? Are beauty and value information which also concerns ethics, morality, emotion, rationality, logic, reality, reason, behavior?

As information, beauty is an experience, property, and quality which informs as a feeling of fulfillment, perfection, improvement, and also as pleasure, happiness, delight, enjoyment, influencing pleasurably through its informationally characteristic filtering, modulation, and harmony the (artistic) emotional domain. In this concern, aesthetics appears as metainformation on beauty, on value of beauty in thinking, comprehension, and behavior. Beauty and the value of beauty are the main subjects of aesthetics which deals with questions as What is beautiful? What are attributes of beauty? What is the value of beautiful? Is beauty an irrational value? etc.

Evidently, it would be very difficult or even impossible to determine beauty qua information by lingual means. Beauty as information is only weakly expressive on the way to language. Beauty is a characteristic informational process of a being's metaphysics, an inward experience of belief which cannot be entirely explicated outwardly. Beauty, and through it easthetics, are evident examples of information and its metainformation, whose transformation from the hermeneutic (ontical) to the semantic (cultural, ontological, linguistic) level is informationally weak.

Informationally, beauty as information and counter-information is Informing of beauty through beauty. In this informational circle, the experience of beauty is arising within metaphysical embedding and within accepted environmental information of a being.

A work of art speaks to a being through its symbols. Only if it speaks, it is symbolic, it is beautiful, it is artistic, it is aesthetic. If it does not speak, if its symbols are not pointing to something which seems familiar, rhythmic, harmonious, observable, investigative, or cognitive, this work of art cannot be observed. It represents a non-understandable object whose symbols are evened by informational noise. In this way, beauty is a regularly relative information, which is populationally, culturally, and ontologically conditioned. However, beauty also speaks hermeneutically, inwardly, generating individual meaning or feeling of a being. The greatest art can appear as nonsense, which in fact does not sense the sensory system of a being at all. At this point, aesthetics, as metainformation of beauty or value of art, confronts with this particular, beauty-concerning informational relativity. It appears that counter-information of beauty, of the value of beauty, and of the value of art (aesthetics) is informing freely, wildly, spontaneously, and circular-unforeseeably. Evidently, informational illumination of aesthetics or informational thinking about aesthetics can bring to the surface observations, which are controversory to a general understanding of art or to teaching of an artistic school, which can reveal the blindness of beauty and art, however, also break down this blindness.

12.2. Ethics

> ... hypocrisy plays an important role in the realization of human societies, permitting human beings under stress to feign having certain properties which they abandon as soon as the stress is removed. This is why in a human society a social change takes place as a permanent phenomenon only to the extent that it is a cultural change: a revolution is a revolution only if it is an ethical revolution.
>
> (Maturana, AAC, xxvii)

The Greek 'ethos' has the meanings of habit, habitude, wont, use, usage, custom, practice, etc. The Greek 'ethikos' means moral, virtuous, righteous, and noble. Some of the basic ethical questions are What is right and good? How does ethics or moral imply conformity to established sanctioned codes or accepted notions of right and wrong? What is rigtness, fairness, equity, or moral excellence? What is moral eminence, particularity, individuality, and freedom from anything petty, mean, or dubious in thinking and behavior? How moral is social and antisocial?

In general, ethics is a philosophical discipline, which investigates objectives, senses, processes, and phenomena of moral wonts and behavior, criteria for evaluation of moral action, and through these, conception and origin of moral and morality. Within these various orientations, ethics can proceed autonomously (from an authentical, ontical, or a being's origin of moral) or heteronomously (from a cultural, ontological, or a population's origin of moral) in an a priori, empiricistic, intellectualistic, axiological, naturalistic or voluntaristic way. Within these conceptions and heterogeneousness of ethics, it is possible to follow eudaemonism, hedonism, perfectionism, etc. stressing individualism (egoism, altruism) or social-moral problems.

12.3. Rhythm and Harmony

> ... One of the most important properties of regularly recurring temporal relationships is that they are predictable. This permits efficient learning and optimization of behavioral trajectories.
>
> (Albus, BBR, 200- 201)

Rhythmic and harmonic informational forms and informational processes appear in the structure of cosmos and in natural forms and processes of matter, space, and time as consequences of different repeating and periodic phenomena; however, they also appear in living substance as particular phenomena of life, for instance, as oscillations of macromolecules, electrical charges, protein production, biological replication, and phenomena of thinking and behavior in a highly developed being. Rhythm and especially harmony create feelings of pleasure, beauty, and good in arising and changeable autopoietic systems. By their nature, autopoietic processes and autopoietic forms are rhythmic and harmonic processes of self-production, self-reference, and circular spontaneity.

Periodic (oscillating) and harmonic informational processes appear in a living cell and are characteristic of life processes of a being. Processes of this kind are, for instance, collective oscillations of cytoskeletal proteins in eukaryotic cells (this is a communicative function of the injection of viral nucleic acids into host cells (Hameroff, ULC, 238)), heart-beat, rhythm of breathing, a being's biological rhythm, harmony of musical, literary, and philosophical expression, etc. Characteristic behavioral patterns using rhythmic and periodic information are walking, dancing, singing, speaking, gesturing, playing, working, thinking, etc. Life activities of a being are synchronized and harmonized (coordinated, adjusted, consonant) according to natural and artificial environmental phenomena; these are life habits, the rhythm of day and night, cyclic gravitational influences of the solar system and cosmos in its entirety.

Poetical, literary artistic, metaphoric expression can have its hidden or revealed rhythmic semantics or lingual rhythmic measure by which phrases from semantically rhymed and harmonically selected meanings are composed periodically. Such a harmony of semantic, stylistic, and even syntactic periodicity arises on different informational levels of audio-speaking, sensory-motoric, and language-understanding hierarchy by means of metaphoric and diversely hermeneutic processes which enter in the highest informational areas (emotional, meaning-concerning, planning, decision-making) of cerebral cortex.

A musical expression has evident properties of basic rhythm and harmony or disharmony of sounds. A melody is a composition of harmonically sequenced and dependent sounds, which appear in phrases by means of regularly repeating themas. A number of synchronized rhythmic harmonies can be the measure for musical informational richness. A simple ballade has only one thema, one melody, and one verse. Multisound, choral music can have many parallel, harmonically dependent melodies and several verses, which can be synchronized into regularly repeated patterns. Symphonical music is based on a rather various set of instruments, where each of instruments is playing a different, however, harmonically dependent musical sequences. A symphonical performance and its comprehension are expressive examples of arising of a complex parallel information in brains of musical performers and their hearers.

In general, rhythm and harmony are characteristic processes of informational iteration and informational recurrence, both of which are survival-regular informational forms of life. In the brain structure, rhythmically repeating informational processes entering the associative memory areas within the internal, a being's model of the world exist. Repeating expectations are caused by these rhythmic informational processes and then these expectations are compared to rhythmically repeated sensory experience. A background of rhythmic patterns of meaning or rhythmic hermeneutics always exists and this hermeneutics pervades the entire generative information-processing hierarchy of the nervous system.

Regularly repeating informational phenomena and their relations are predictable. This property of informational iteration and expected informational recurrence enables effective learning of a being and through the learned or captured information also a senseful or optimal behavior. In this way, outcomes from various,

also critical situations, can be foreseen and several deviations from normal informational patterns can be revealed and learned. The capability of foreseeing and prediction is based on the capability of distinguishing differences which occur between expected and observed information. These differences are the so-called information of error concerning expectation. Information of expectant difference can be pleasant (harmonic, beautiful, good, rewarding, stimulative, healthy) or unpleasant (disagreeable, ugly, bad, punishable, stressful, pathogenic).

Recognizing of essential, although small systematic deviations from the predicted ones, gives rise to learning and to arising of more complex recognitions and predictions. Recognizing of deviations between expected and observed phenomena is the basis of a complex learning of a being, of changes in information-processing of brain. Differences between expected and observed informational processes also modify individual expectations, hypotheses, world models, and behavior of a being. In this concern, perception is an informational process of searching for such hypotheses that can produce expectations which are adopted to the appearing observations. Perception qua information is an instructive example of circular informing, counter-informing, and embedding of information.

A being masters its environment through the capability of prediction of its sensory information and through the capability to select such thinking and behavioral patterns that predominantly, consistently, or reliably produce results of delight (of beauty, good, pleasure, advantage, survival). In a learning environment, repeating of matter is rewarding or attractive by itself (repeating in singing, rhythmic movement and gesturing, hearing of known musical records, memorizing of sound phrases, everyday and working habits). In a potentially hostile environment, a being has several essential possibilities of survival if it can master the prediction of outcomes of its future actions. In a periodically changing environment such predictions are much more reliable than in a spontaneously or non-periodically varying world. In this way, every informational environment which demonstrates phenomenal periodicity is informing a being rewardingly. In this context, harmonic relations occur between informational hierarchies of behavior generation and of sensory information processing. Dwelling in a predictable environment appears to be pleasant, non-arduous, manageable, safe, rhythmic, and harmonic.

It is clear that an exact prediction is not always possible. Sudden, abrupt, unforeseen deviations from the expected or predicted ones can produce feeling of surprise, where surprise stirs up attentiveness and generates information of awakeness, openness, vigilance, lurking, willingness, and readiness. Processes and phenomena which are delightful in a surprising (a new, an unexpected) way are rewarding in an emphasizing way because before all they awake the lurking and then they reward beyond the expectation. Pleasant surprisings can awake behavioral responses like laughter, crying, and excessive enthusiasm.

A pleasant surprise can constitute the essence of good humor. An interesting story can end in an unexpected, thoughtful distortion, perversity, deformation, spinning, metaphoricalness, ambiguity. Daring, critical, risky humor has its interest in a luring action of exceeding, breaking, overstepping boundaries and limits of environmental, cultural morality, taboos, ceremonials, and habits. Humor possesses its unexpected, metaphorical rhythm and its spinning, semantically and hermeneutically ambiguous harmony.

Unpleasant surprises can awake emotional responses of fear, anger, aggressiveness, reluctance, disgust. In a hostile environment, new and unpleasant events can awake the feeling of danger, risk, or critical state. Deviations from the normal can cause disaster, accident, catastrophy, misadventure, misfortune. Incapability of prediction means to be surprised, means the possibility to lose the existing advantages of survival. Without predictability not only learning is threatened, but also the survival of a being or a population. A continuing and prolonged decay, disintegration, disorganization, and interruption of regular informational patterns in the internal rhythm and environmental stimuli of a being destruct predictability, disable normal learning processes, and cause individual and populational stresses (as forms of punishment) that can produce various social pathologic kinds of behavior (neuroses, intoxication like alcoholism, smoking, drugging, suicidity).

Rhythmic informational processes are periodic. Periodicity enables the prediction of information arising in informational processes. Harmonic informational processes are adjusted to experienced and memorized expectations, and in this respect, they are predictable. Informational prediction is the kernel of the arising of rewarding information. This information embeds (confirms) individual learning and experiencing principles, and changes them, if environmental phenomena are changing. Rhythm and harmony as information inform rewardingly, arise and are embedded circularly through the information of reward. However, the principle of rewarding is only one of many other aspects of informational rhythm and informational harmony.

12.4. Art

> ... We shall now attempt a first characterization of Nietzsche's total conception of the essence of art. We will do this by exhibiting a sequence of five statements on art which provide weighty evidence. ...
>
> 1. Art is the most perspicuous and familiar configuration of will to power; ...
> 2. Art must be grasped in terms of the artist; ...
> 3. According to the expanded concept of artist, art is the basic occurrence of all beings; to the extent that they are, beings are self-creating, created; ...
> 4. Art is the distinctive countermovement to nihilism; ...
> 5. Art is worth more than "the truth."
>
> (Heidegger, NIE, 67-76)

Art seems to be the summit of unanalyzable, sensitive, expressive, and recipient informational spontaneity, the synonym for the free creative power, however, also for informational skill, cunning, artifice, craft, and will to power. Literally, in its characteristic environment, a work of art is

arising, is creating and being created circularly, iteratively, repeatingly, rhythmically and harmonically. The artist creates and simultaneously observes, investigates, recognizes, experiences, and admires the beauty, artistry of his work. On this way, also the new, upredictable is arising, being created out from the artistic admiration and from the will of artistic power. The arising of a work of art proceeds successively, and from the achieved, new artistic details are growing sensitively and spontaneously. Thus, the artificially unforeseeable, unexpectable is generated. Certainly, the arising of work of art is social-circularly interactive, that is, interactively social-spontaneous.

Art is a specific people's mental, that is informational activity that carries information of sensitivity, incorporating creative glance, the work of art and its experiencing. Art is the capability of aesthetic creation, expression, reception, of aesthetic informing in the domains of feeling, thinking, fantasizing by means of spoken or written words, instruments, human voice, color, mimics, plasticity, gesture, etc. In a broader sense, art incorporates work of literature (drama, lyrics, epic, novel, story, tale, essay), work of music, ballet, acting, painting, sculpture, graphics, architecture. Art is productive (first-hand creativity) and reproductive (making music, acting, reciting), applicative (trade, industrial, professional), and mediative (film, radio, television).

Art informs in a free and spontaneous way. In symbolic art, information is coming into existence from abstract indefiniteness, from uncertainty of a form or of a process of performance and comprehension. Sculpture is predominant in classical art, and the inward, spiritual dominates over the outward in the art of romanticism (painting, music, and poetry). As information, in principle, also l'art pour l'art is nothing more than a consequence of artist's social and antisocial tendentiousness, of his informational obscurity, ambiguity, and enigmaticalness. Art is cultural information and has little or no essential influence outside its cultural domain.

12.5. Literature

> ... Literature is what has been literally written down, and copied, with the intent that it be available to a reading public.
>
> (M. Heidegger, WCT, 134)

Literature is information which dwells in many different forms and processes of its production and reception in the realm of natural or formalized (musical, programming, scientific) languages. The fundamental aspect of literature as natural or formalized language usage (speaking, hearing, writing, reading) is the social interaction of storytelling, of verbal expression and perception of stories, tales, adventures, imagination (also in the areas of music, formal sciences, programming).

Storytelling is the basic process in writing and speaking. The basic component of every literature is a story which can be philosophical, artistic, scientific, technological, and/or occasional. In the mind of a recipient, a story can create several informational paths (states, processes, and patterns of them) which can approach the sting or experience of the story in an associative and in a bisociative (unrelated) manner. A linguistic informational path is determined by a sequence of words and is understood to be a program for a distinct informational process or for a story. Each informational path represents a story for itself. Lower paths in the story hierarchy represent details of a story, whereas higher paths are stories with more comprehensive vocabulary, however with less details. These paths are actions in the hierarchy of storytelling behavior.

Informational paths can belong to very different informational segments of a being's metaphysics (total information of a storytelling and/or story-hearing being). For instance, informational paths in the hierarchy of sensory information describe experiences and feelings. Informational paths in the hierarchy of world-view or world model describe hopes, expectations, dreams, and beliefs. Each storytelling behavior can be composed of several stories which appear in recurrent, simultaneous, parallel, and sequential relation to different lower and higher paths, to details and programs of a story.

Informational paths as hermeneutical (sub)stories possess basic roles, for they influence a storyteller's behavior, determine experience, and cause imagination. Hermeneutic-semantic translation of informational paths into sequences of words causes the arising of language. A storytelling strategy selects one single sequence of words from many existing informational paths. Essential effects of an arising story can be achieved by jumping from one informational path to the other and from one hierarchical level to the other. A descent to a lower informational path enables the development of an exciting, actual, or essential detail of the story, whereas an ascent to a higher informational path enables a fast transition over unexciting, ordinary, or unessential events.

Informational paths (informational forms and informational processes) of mind (in brain) are various as much as possible, for instance, they are behavioral, experiential, emotional, faithful, intelligent and they enable jumping and transition over informational paths in a forward, backward, upward, and downward direction in groups of informational paths and their hierarchies.

13

13.1. Crisis

> Being of **crisis** is informational. Crisis as information is circled, closed into its own realm of criticalness. On the way out of crisis, its informational circle has to be broken into a new, releasing perspective.

A being's and a population's capability and incapability to survive, to be successful, to develop rhythmically and harmonically, can be particularly investigated through the perspective of an individual, populational, or social crisis. What is crisis qua information in the context of biological, behavioral, and informational capability and incapability and how does it arise?

The meaning of the Greek 'krisis' is manifold, for instance: separation, exposition, decay; quarrel, conflict, strife; claim, suit, truth; decision, breakdown; judgment, arbitration, opinion; ordinance, fixation. Information of crisis arises in different occasions, for instance, when a blind informational orientation is broken down by a new, necessary information for surpassing the blindness, when a ruling ideology is not capable to change incapable (unsuccessful) individual and social behavior in sharpened social circumstances, when the immune system of a being cannot resist to or neutralize pathological invaders, when a disease continuously progresses onto the account of health, when a being is feeling stress, repression, or hostility, etc. and begins to develop counter-information for improvement, defense, or change.

Crisis arises informationally from differences occurring among particular information, meanings, understandings, and informational backgrounds in the form of conflicting and decisive information among them. Crisis is differential information which considers informational differences as the most substantial and most essential informational entities. In crisis as information, differences are compared, confronted, and developed to serve in searching the way out of crisis, to solve the problems of crisis, and at last, to annul the information of crisis. If crisis cannot be solved satisfactorily, its information is growing in complexity overwhelming other, vital information and ending in catastrophe in which it ceases to exist.

In general, it could be said that crisis begins to arise in a form of counter-information from information within which the critical incapability is coming to the surface, through the recognition that this incapability is a consequence of informational blindness, insufficient knowledge, incorrect inference and cognition, etc. This recognition of blindness simultaneously represents informational breakdown, the point in which critical information begins to develop. This information which relates blindness and breakdown of blindness is called crisis. Crisis is an information of opposites which appear in the critical context of crisis. Simultaneously, crisis is a corrective information which insists to correct the ruling blindness into a new illumination and perspective, to solve the problem of crisis, to decide on the way to new orientation, thinking and behavior. In this respect, crisis is developing as regular counter-information through specific, crisis-concerning Informing, counter-Informing, and embedding.

Crisis as information appears in everyday life, is a consequence of individual or populational criticalness, is regular (well-intentioned, democratic, cultural) or irregular (subordinative, repressive, radical) information. Crisis as regular information gains the development of individual and populational capability, its irregularity leads to the development of informational incapability. Regular crisis appears to proceed by small, goal-directed, and successful steps, in an appropriate developmental rhythm and harmony, whereas irregular crisis acts chaotically, radically, archaically, ideologically, and in this respect arhythmically and disharmonically, also destructing already positive developmental achievements of a being's culture.

13.2. Capability

The basic question of survival of a being and its population remains in the decision: to develop individual and populational capability or to stay incapable.

Capability as information of a being concerns the metainformational structure and metainformational organization within the total information of a being, this is, informational structuring and organizing of its metaphysics. Information as informational process integrates informationally lower and higher components of information and information-processing. It is simultaneously a metainformational process which creates lower and higher informational components. A metainformational hierarchy includes basic informational components, metacomponents and higher metacomponents of a general degree

meta-meta- ... meta-components.

Metainformational hierarchy has the property of metainformation, that is, of a circularly (metarecurrently, metaparallel, and metasequentially) spontaneous metainforming, metaarising, metacounter-Informing, and metaembedding. Metainformation performs as information.

Capability is the information of comprehension and of behavior, by which a being or a population is observing, investigating, and recognizing its own information and the outside information. Capability is a degree, a stage, the greatest achievement in the comprehension of the living, also of the human mind. The life and the survival perspectives are directly dependent on abilities of observation, investigation, and cognition of a being, of its population, and of its environment. There is a substantial difference between intelligence and capability. Intelligence is above all information, which is a consequence of exercise, practice, experience, drill, instruction, learning, informational selection, and will. Capability on the other hand is the supreme and creative ability in search and examination of various and complex informational circumstances. Capability of a being is determined by the ability of its self-comprehension, where a being is critically investigating its entire information and the appropriateness of its intelligence and behavior. The counterpart of capability is characterized precisely by the deficiency of this self-comprehensive informational component.

Capability of a being is grounded in its self-comprehensive Counter-Informing. In this context, all of the four attributes, which are self-reference (survival, ethics, moral), comprehension (self-observation, self-investigation, self-cognition), intensive arising of Informing (creativity, variability, alternation), and resulting counter-information (antagonism, resistance, rebellion, disagreement, impossibility, etc.) are necessary. By capability a being is changing itself, realizing itself to a new, more appropriate capability. In this respect, capability as information concerns capability.

Capability as information can be an active strategy of increasing the structuring and organizing principles of a being's total information (its metaphysics). Through such an

intension strategical thinking and behavior are coming into existence that support the development of capability. Through capability, a being is intentionally changing, developing, improving its strategic information for its life and survival, and through this information, it is changing its behavior, developing new intelligence, and improving its strategic thinking.

13.3. Incapability

> To apply ideologies in practical reasoning and behavior means to follow the way into incapability. Thereupon, informational circle of incapability closes: ideology and incapability produce each other and perform a stable course of development into incapability.

Capability or incapability is the highest form and processing of informational structure and organization of the human mind. If informational capabilities lead to survival and development, then incapabilities approach to regression, decay, and dying out. There are some essential informational (also dialectical) opposites between capability and incapability. Whereas capability builds its decisive information on forms and processes of spontaneity, creativity, and throwness into factual situations considering developmental and intelligent experiences, incapability rests on philosophical and idelogical forms and processes of informational orientation, uniform (population-molded, conform, individually stupid) behavior, and predetermination of thinking and behavior in factual situations, considering promissory ideological experience.

In the life of a being and its population, the already acquired capability can suddenly change to such a degree that it in reality descends into incapability, for instance, through the practical use of a radical, revolutionary, unethical, or social-improper ideology. Such a turn in thinking and behavior through ideology prepares the way on which incapability dominates, so it can take its triumphant march into a stable, self-referencing, self-productive, and self-satisfactory incapability. In this case, incapability as information develops an incapability of a special kind, penetrating deeper and deeper into its own informational realm. In the course of time, incapability of this sort, which becomes beings' and a population's total orientation in thinking and behavior actually destroys informationally regular relations, capable informing and counter-informing and causes individual and social disaster (decay of sound economy and of skillful labor, disappearance of spontaneous creativity and intellectuality, increasing of lawlessness and social diseases, etc.)

14

14.1. Functions of the Mind

> ... While the principal task of the philosophy of mind may be regarded as the attempt to provide an account of what mind is, the complexity of the task is such that there is little in philosophical literature that will seem to constitute an integrated theory of the mind per se. Rather, issues that are central to the philosophy of mind occur throughout metaphysics, epistemology, logic, aestethetics, and, particularly, in the contexts of the philosophy of psychology and what is called mind-body problem. In contemporary philosophy, most discussions take the form of analyses of specific mental concepts: for example, consciousness, emotion, imagination, introspection, intention, thinking, and the will.
>
> (DPH, 233, 234)

Viewed informationally, mind is a notion which concerns informational forms and informational processes of the human nervous system, this is, informational structure and informational organization of central and peripheral nervous system and of the body and its various kinds of processes as a whole. Mind concerns known and yet unknown information of body's processing, conscious or integrative and microstructural information on the level of brain and on the level of processing within a particular living cell. The concept of mind is unrevealed and a being observes its own mind through "looking" into it experientially, investigating it in a top-down (brain-integrative, global, psychological) and in a bottom-up (biologically, physically, and biochemically fundamental or neurophysiological) manner.

Several functions of mind have been discussed in this essay, for instance, intelligence, creativity, language, ideology, crisis, incapability, etc. However, still many of basic functions have to be analyzed informationally (memory, learning, vision, etc.)

14.2. Information Systems

> ... Advances in intracellular imaging and molecular biology have illustrated the complex dynamic organization of intracellular cytoplasm. Specifically a dense, parallel, highly interconnected, solid state network of dynamic protein polymers, the "cytoskeleton," is a medium which appears to be ideally suited for information processing, and which is actively involved in virtually all cell functions.
>
> (Hameroff, ULC, 36)

Information informs (influences) and informatizes (generates actively) in several systems of information. An information system (or informational system) does not exist as a singular (individual) system of information, as an informationally isolated or informationally independent system. However, information systems exist in the form of informational plurality, interacting among themselves, participating in other information systems, being informationally distributed, associated, and integrated. As a consequence of informational plurality, information systems are to some degree mutually informationally embedded in each other, in informationally divergent and convergent manner. Within this informational plurality, information arises as a consequence of systems' plurality, this is, circular-spontaneously in the informational sense. In fact, arising of information brings new informational systems into existence. Information systems integrate a more or less informationally universal realm of information.

In general, information is always thrown (injected) in an informational realm in which it operates and is operated on its way of informing. On this way of informing, information finds its informational ground, its embedding and its distributive informational power in several developing information systems. Information performs as information system. Information system is only another form of information.

14.3. Social Systems

> ... An autopoietic system participates in the constitution of a social system only to the extent that it participates in it, that is, only as it realizes the relations proper to a component of the social system. Accordingly, in principle, an autopoietic system may enter or leave a social system at any moment by just satisfying or not satisfying the proper relations, and may participate simultaneously or in succession in many different ones.
>
> (Maturana, AAC, xxv)

By philosophical terms, an autopoietic system will be presented as being-in-the-world (or, more generally, as Dasein). What does a social system mean from the point of view of living integration? What is the nature of the so-called social information?

A very important constitutive (integrative) domain of a human social system is the cognitive linguistic behavior of human beings that integrate such a system. In a social system, beings of this system inform (or informatize) among themselves in a cognitively characteristic way using different linguistic, symbolic, behavioral, and environmental forms of informing, counter-informing, and embedding of being's and of society's information. Every circularly arising informational spontaneity is socially bounded and it underlies the characteristically bounded autopoiesis of a particular society. This informational boundary that constitutes a society's informational closeness (particularity, peculiarity, prohibition), contributes essentially to the constitutive and integrative power which orders, manages, subjects, and governs beings of a society into main informational perspectives of the social systems within society.

From the informational point of view, a social system is integrated on the basis of an instantaneously ruling social consciousness (a state of culture, of economic development, of a population's ingenuity) - the society integrative ideology. This ideology is no more than a commonly (socially, culturally, populationally, democratically, ethically) or authoritatively (asocially, ideologically, totalistically, repressively) accepted information which implements the informational coupling among beings, realizing a social autopoiesis. In this concern, a social system behaves as a being of beings, as the populational being. True and valid social information is only the commonly accepted information of beings which integrate a social system. Otherwise, social information is disagreeable, disappointing, and disintegrative and has not the power of the being's validity and acceptance.

14.4. Artificial Intelligence

> ... twenty-five years of artificial intelligence research has lived up to very few of its promises and has failed to yield any evidence that it ever will. The time has come to ask what has gone wrong and what we can reasonably expect from computer intelligence. How closely can computers processing facts and making inferences approach human intelligence? How can we profitably use the intelligence that can be given to them? What are the risks of enthusiastic and ambitious attempts to redefine our intelligence in their terms, of delegating to computers key decision-making powers, of adapting ourselves to the educational and business practices attuned to mechanized reason?
>
> (Dreyfus, Dreyfus, MOM, xi)

Artificial intelligence as a discipline could be hardly classified as a science, art, or even technology. Does it belong to a kind of new era's working philosophy which only searches its authentic field in relation to concepts and yet indeterminable images of natural intelligence?

On its way of development, artificial intelligence has established its own sorts of applicability and methodology. Its applications can be classified, for instance, as theorem proving, games, robotics, vision, natural language processing, automatic programming, knowledge engineering, etc. Its basic methodology has developed as representation of problems, searching, inference mechanisms, languages for artificial intelligence, representation and utilization of knowledge, etc. These methods and activities tend to explore and investigate information processing in mind and machines. In this respect, artificial intelligence is a real and particular information-concerning discipline dealing with information in mind, brain, and in possible future machines. Artificial intelligence particularly concerns information, its forms and processes in the living and the artificial.

By methodologies used up to now, artificial intelligence has not achieved the goals and promises which it has given during its developmental phases. It seems evident that its philosophy of research was founded on the belief which says that Western-traditional, rationalistically hard, and in some way essentially reduced (simplified, quasi-scientific) techniques could be sufficient for mastering problems in the domain of intelligence. However, up to now, artificial intelligence has not succeeded to realize any real or evident intelligence-like product, for instance, neither on the level of computer architecture nor in the domain of computer programming. The progress artificial intelligence has brought lies in the field of improved algorithmical concepts which accelerate, combine, and reduce data processing, however do not prove to be intelligent in any respect.

Although artificial intelligence can and must use rationalistic experience of the past, it must simultaneously step on and search new ways outside of traditional thinking and technological behavior. In the course of time, it has to establish a new philosophy which will consider results coming from marginal sciences

and from marginal philosophies. Philosophy of information sketched in this essay and in Principles of Information (POI) is marginal.

15

15.1. Conclusion

In this essay, trials have been made how to determine different phenomena and/or their phenomenologies informationally. There could be understood that it is always possible to observe, investigate, explain, recognize, and finally also comprehend the entire cosmic phenomenology through informational illumination, where everything (metaphysically, on conscious or other non-cognitive level) appears as information. In this essay, first of all examples of the so-called integrative, global, cortical, mind-concerning informational forms and informational processes have been revealed. No examples belonging to the field of biomolecular consciousness (e. g., Hameroff, ULC) were given (for instance, informational processes of molecular computing; of consciousness as particle/wave physics, as a property of protoplasm, as learning and as a metaphysical imposition; of holography, neuronal signalling, microtubules, solitons; cytoskeletal information processing, viruses, etc.) This kind of informational forms and informational processes will be treated in a separate essay.

The objective of this essay is to show the necessity and usefulness of further development of information philosophy, theory, and technology within a broad, phenomenologically all-embracing informational realm and not only from the standpoint of existing mathematical and communicational theory of information. In the future, the arising information philosophy should also essentially contribute to the philosophy of technological possibilities when intelligent machines will be constructed.

C

(BBR) J. S. Albus: Brains, Behavior, and Robotics. Byte Books. McGraw-Hill. Petersborough, N. H., 1981.

(CPY) J. R. Anderson: Cognitive Psychology and its Implications. Second Edition. W. H. Freeman & Co. New York, 1985.

(MOM) H. L. Dreyfus, S. E. Dreyfus (with T. Athanasiou): Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer. The Free Press, Macmillan. New York, 1986.

(DPH) A. Flew (Editor): A Dictionary of Philosophy. Revised Second Edition. St Martin's Press. New York, 1984.

(PHH) H.-G. Gadamer: Philosophical Hermeneutics. University of California Press. Los Angeles, 1977.

(ULC) S. R. Hameroff: Ultimate Computing: Biomolecular Consciousness and Nanotechnology. In Print: Elsevier Science Publ. 1987.

(BAT) M. Heidegger: Being and Time. Harper & Row, Publ. New York, 1962.

(BW) M. Heidegger: Basic Writings. Harper & Row. New York, 1977.

(BPP) M. Heidegger: The Basic Problems of Phenomenology. Indiana University Press. Bloomington, 1982.

(QCT) M. Heidegger: The Question Concerning Technology and Other Essays. Harper & Row. New York, 1977.

(NIE) M. Heidegger: Nietzsche. Volume I: The Will to Power as Art. Routledge & Kegan Paul. London, 1981.

(WCT) M. Heidegger: What is Called Thinking? Harper & Row. New York, 1968.

(AAC) H. R. Maturana, F. J. Varela: Autopoiesis and Cognition: The Realization of the Living. D. Reidel P. C. Dodrecht, Holland, 1980.

(HMP) M. Murray (Editor): Heidegger and Modern Philosophy: Critical Essays. Yale University Press. New Haven, 1978.

(AOT) G. Wallas: The Art of Thought. Harcourt Brace. New York, 1926.

(CRY) R. W. Weisberg: Creativity: Genius and Other Myths. W. H. Freeman & Co. New York, 1986.

(OWI) A. P. Zeleznikar: On the Way to Information. Informatica 11 (1987), No. 1, 4-18.

(IDI) A. P. Zeleznikar: Information Determinations 1. Informatica 11 (1987), No. 2, 3-17.

(POI) A. P. Zeleznikar: Principles of Information. Informatica 11 (1987), No. 3, 9-17.

E R R A T U M

In the article A. P. Zeleznikar, Principles of Information, Informatica 11 (1987), Nr. 3, pp. 9-17, pages 11 and 12 were printed twice! Instead page 11, the next page (page 26 in this issue) has to be inserted. We appology to our readers for this fatal error.

examples. On the other side, the following principles will depend upon the previous ones, so that all principles will be recurrently interwoven among themselves and simultaneously open for further comprehensional development.

THE CONSEQUENCE OF INFORMATIONAL PRINCIPLES. Principles of information will be counter-informational too. This means that irrespective of the principle of information in this essay, a principle of information will have the property of information, of its circular and spontanebus development (thinking, arising, embedding) in the realm of information.

THE CONSEQUENCE OF INFORMATIONAL OBJECT AND INFORMATIONAL SUBJECT. Information is an informational object and/or informational subject (a mastery over itself). This fact can be expressed in formalized terms where information can be understood as an informational operand and/or as an informational operator (operation). Informational objects are both informational forms and informational processes which are simultaneously informational operations. Informational objects (operands) can be formally expressed as unities in the forms as informational_form, informational_process, being_in_the_form, being_in_the_process, etc. Similarly, informational operations (operators) can be expressed by different verbal compositions, e. g., inform, generate, arise, give, vary, come_into_existence, come_in_presence, modify_intelligently, embed, etc.

THE CONSEQUENCE OF COUNTER-INFORMATION. Counter-informational objects and counter-informational operations are, for example, counter-informational_form, counter-informational_process, etc. as objects and inform_by_observing, generate_through_cognition, arise_by_investigation, etc.

THE PRINCIPLE OF INFORMATIONAL FORMULA. Information can be formally expressed by informational and counter-informational formulae which can be constructed by using informational operands (objects) and informational operators (subjects) in a linguistically free, unbounded form. Informational operations and their compositions can be used as informational operands (objects) and as informational operators (subjects). Although an informational formula is a free sequence (a serial expression) of informational operators and informational operands, it can (formal-linguistically) express circularity, spontaneity, parallelness, etc. of forms and processes appearing in an informational formula.

Now, some examples of primitive and composite informational formulae can be shown.

AN EXAMPLE OF BASIC INFORMATIONAL INFERENCE. The following formula is the deductive inference represented in the usual if-then form:

```
information informs
                its_own_and_other_informing;
     informing informs information;
-----------------------------------------
     information informs information;
```

An individual statement is terminated by a semicolon. In general, a list of statements can represent an 'and', 'or', or even an 'and/or' statement, etc.

AN EXAMPLE OF INFORMATIONAL EQUIVALENCE. The following formula describes the equivalence

```
information informs intelligence
          is_equal_to
information informs informing;
informing informs intelligence;
```

Operands and operators of a formula can be constructed by free choice. A statement before an operator is not terminated by a special sign.

AN EXAMPLE OF TWO MEANINGFUL FORMULAE. The following two formulae, where the first one concerns the operation composition and the second one the equivalence of two formulae, are meaningful:

```
(1) inform is_equal_to
                inform, generate, modify;

(2) information informs intelligence
                is_equal_to
    information informs, generates,
                modifies intelligence
                is_equal_to
     information informs intelligence,
    information generates intelligence,
    information modifies intelligence;
```

In this example some statements are terminated by the comma to explicate the special 'and' connective, etc.

THE CONSEQUENCE OF THE FORM OF THE OPERATOR 'INFORM'. The most general (and perhaps the strongest) informational operator is the operator 'inform'. This operator can be understood as a composition of an arbitrary sequence of other informational operators. Each operator composition can be meant to be purely parallel, a mixture of parallel and serial, or purely serial. The order of an individual operator in the composite sequence may or may not be determined by distinguished formal connectives.

2. Principles of Informing of Information

In this section we shall deal with several principles which originate in the meaning of the verb 'to inform' and several of its consequent derivatives. Let the verb 'to inform' include the meaning of all possible verbs and verbal compositions. In this way, the verb 'to inform' can represent an arbitrarily complex informational operator. Let this meaning also be transparent to the noun 'information'. Yet, information is understood as an arbitrary form, an arbitrary process, or as both of them. In this way, information can represent an arbitrarily complex informational operand (object) and informational operator (subject).

Informing of information as the broadest comprehension which is imaginable concerns four basic forms or processes of information. These are: informing, embedding, arising, and Counter-informing of information. This reflection yields the following four principles:

THE PRINCIPLE OF INFORMING OF INFORMATION. Informing of information, where informing is inherent to information itself, means that through informing, information informs according to the principle of information, that through informing counter-information is coming into existence according to the principle of counter-information, and that this counter-information is embedded in existing information according to the principle of embedding of information. Informing circularly and spontaneously concerns information according to

UDK 681.3.001:519.6

Jurij Šilc and Borut Robič
Jožef Stefan Institute, Ljubljana

Abstract. Execution models of the data flow based parallel inference machine for OR-parallel and AND-parallel Prolog and the experimental machine architecture are presented. It is shown that two types of logic programming languages with different aims can be supported on this machine. The programs are compiled into data flow program graphs corresponding to machine language codes. Thus, parallelism in the program can be exploited naturally. The machine is constructed from processing elements and structure memories interconnected through a hierarchical network. The processing elements interpret the procedures represented by the data flow program graphs in parallel. Structured data is distributed to structure memories and shared among these procedures.

Keywords. Fifth generation computer systems, parallel inference machine, OR-parallel Prolog, AND-parallel Prolog, data flow mechanism, machine architecture.

1 Introduction

The Fifth Generation Computer Systems (FGCS) research and development aim is to build a prototype of knowledge information processing system capable of efficiently performing knowledge-based problem solving and inference. Toward this end, a ten-year period has been assigned to the FGCS Project, and this period has been further devided into three stages. The goal of the initial three-year stage is to conduct basic research on individual system components in order to establish basic configuration technology for subsystems which are to be realized in the intermediate four-year stage.

Fig. 1 shows what has become known as the "basic configuration image" of the fifth generation computer [1]. Looked at vertically, it has a hardware layer, a software layer, and an external interface to applications systems, as might be expected. Looked at horizontally, it becomes clear that each aspect of the functionality of a fifth generation computer - problem solving and inference, knowledge base management and intelligent interfacing - requires its own hardware and software support mechanisms.

The parallel inference machine and knowledge base machine are the most important hardware components of the FGCS. In the FGCS prototype to be completed as the final product of the project, the two machines will be integrated through a close link. In the initial stage, however, research and development are proceeding separately for each machine with research themes separately determined, since the initial stage mainly aims to conduct research and development of individual component technologies to establish the basic technology for the hardware, called the inference subsystem and knowledge base subsystem to be build in the intermediate stage [8].

2 Knowledge base and inference subsystems

Development of the FGCS hardware and architecture will include the implementation of mechanisms for processing and controlling a knowledge base and efficient execution of problem-solving and inference techniques with these mechanisms. The system will depend on multiprocessing and parallel processing techniques for which two objectives are critical:

(1) Provide machines with the power to handle the natural parallelism found in problems tackled by humans. The structure of a problem and the neccessary processing for solving it can be shown by rules controlled by an inference mechanism. Thus a major goal of the FGCS project is to devise an execution model for the inference mechanism and to determine a way to configure it.

(2) Achieve high-speed parallel processing capable of supporting intelligent human activities. For this requirement, the principal research must be concentrated on knowledge base processing algorithm to handle a large number of facts as well as a mechanism supporting the algorithm.

The inference subsystems, together with the knowledge base subsystem, forms the kernel of the FGCS hardware [8]. The ultimate aim of the FGCS research and development project is a machine enabling the execution of parallel inferences [3,7,9]. In the following we shall describe some FGCS project "data flow directed" efforts in designing such a machine.

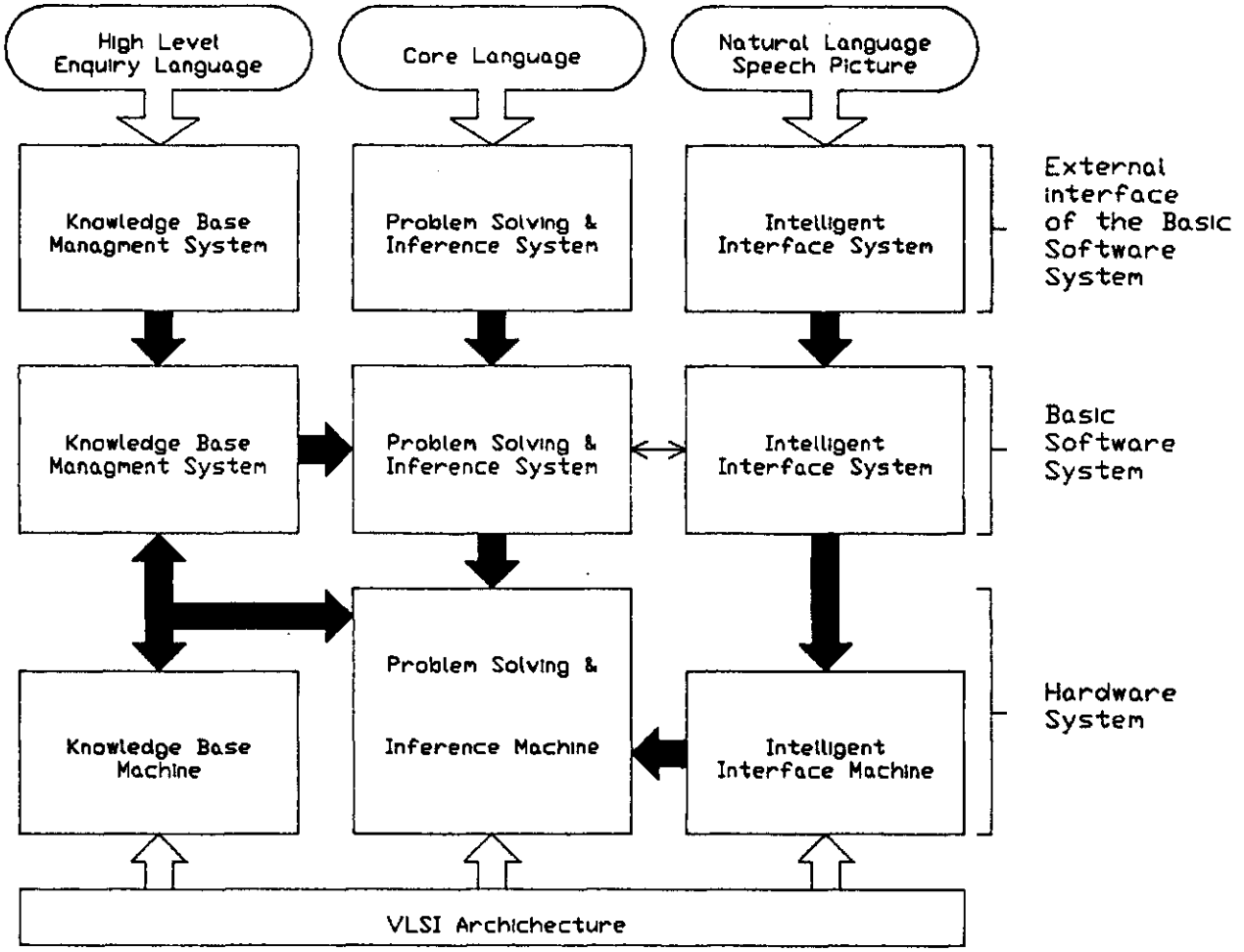


Fig. 1 The overall structure of a fifth generation computer.

3 Parallel inference subsystem

Machine language of parallel inference machine

In FGCS project, logic programming was selected as a bridge to fill the gap between a highly parallel computer architecture and knowledge information processing. Several logic programming languages, named kernel languages, which define an abstract interface between the hardware and the software, are beeing developed. The kernal language KL0 is the machine language of sequential inference machine from which a parallel version KL1 is beeing developed. KL1 is the the machine language of parallel inference machine [2] including two types of basic languages: AND-parallel Prolog and OR-parallel Prolog [4].

In the execution of logic programs, a high degree of parallelism can be implemented with use of AND-parallel and OR-parallel executions. When OR-parallelism is applied alternative clauses of the same goal are executed in parallel. The alternative clauses have identical initial states and do not interfere with each other, except for possible concurrent initialization attempts of a goal variable by multiple clauses. On the other hand, in AND-parallelism the conjunctive goals of a clause body are executed in parallel. In general, the goals may share variables and thus interfere with each other when the shared variables are accessed concurrently. AND-parallelism in logic programming involves the simultaneous execution of subgoals in a clause. Whereas OR-parallelism attempts to achieve increased speed by investigating many possible solutions in parallel, AND-parallelism attempts to achieve increased speed by investigating the subparts of a particular solution in parallel.

In conventional sequential Prolog the search and test operations (called unifications) are executed one by one, but parallel search and test operations can be implemented through parallel machine architecture to obtain a high-speed machine. A few sources of parallelism which can be distinguished for parallel execution of Prolog are AND parallelism, and OR parallelism.

(1) OR-parallel Prolog. When a goal literal G is given, the definition of G is invoked. A clause C is then selected from the definition, and unification of G and the head H of the clause C is attempted. Generally, when multiple clauses C1, C2, ... Cn exist in the definition, unification of G and each H1, H2, ... Hn can be executed in parallel. A unit clause Ci that is successfully unified with G returns the solution(s). A nonunit clause Cj initiates the next unification, treating its body as a new goal statement, and waits for the solutions. The resulting solutions of the goal G are merged into streams by stream merging primitives (in the order in which they are obtained) and then returned to the goal. Thus, OR-parallel Prolog is suitable for the clas of "search-for-all-solutions" problems.

An example of OR-parallel Prolog is Parallel Prolog [4].

(2) AND-parallel Prolog. When a goal is expressed as G1 AND G2 AND ... AND Gm, AND-parallelism can be used to search for conditions for all literals Gi in parallel. The goal statement is satisfied only when solutions are found for all the literals Gi and there is no inconsistency between these solutions. The consistency checking is easy or even unnecessary in cases when the goal literals Gi have no unbound variables shared among AND processes, or where shared variables are bound to the ground instances before invocations of these literals.

Several languages have been proposed to realize AND-parallelism. They include PARLOG, Concurrent Prolog, and GHC (Guarded Horn Clauses) [4,5].

Mechanisms of parallel inference

Various mechanisms of parallel inference and architectures based on those mechanisms are beeing studied: data flow mechanism, reduction mechanism, complete-copying mechanism and, clause unit processing mechanism. In what follows we shall briefly describe these four mechanisms [8]:

(1) Data flow mechanism. In the data flow concept, execution starts when data necessary for the execution arrived. This concept can result in parallelism regardless of whether it is explicitly indicated in the program. This mechanism executes kernel language programs in parallel based on the data flow concept.

(2) Reduction mechanism. When executed, an OR-parallel and AND-parallel Prolog program generates resolvents from a goal and clause. This can be regarded as a process in which a goal modifies itself using a clause as a rule. The reduction mechanism can also be viewed as a kind of self-modification. Thus, there is a close similarity between the execution of OR-parallel and AND-parallel Prolog programs and the reduction mechanism. Accordingly, the reduction mechanism was selected for a machine architecture that executes OR-parallel and AND-parallel Prolog programs.

(3) Complete-copying mechanism. Complete-copying is type of reduction mechanism. Even if a process includes several literals (subgoals) and only one literal (subgoal) is reducible, the whole process is copied and transferred to a unit that executes the unification process. This increases the number of copies and the length of a packet in the network, while enhancing the independence of each process.

(4) Clause unit processing mechanism. In response to a request from an idle processing unit, a busy processing unit sends a process. Thus, this mechanism can avoid an explosion of resource requests. However, it takes time for all processing units to become busy.

4 Data flow based inference machine

The parallel inference machine based on data flow mechanism (PIM-D) is naturally well suited to parallel procesing becouse the data flow mechanism is closely related to functional languages.

Data flow computation ...

Programs in the data flow model are represented by data flow graphs, nodes correspond to operators and directed arcs correspond to data paths along which operands are sent. An operator is driven by operand arrivals from its input arcs, and it outputs the result operands to its output arcs without affecting the other operators' execution. This functionality of operators has close similarity to the functional languages.

... and logic programming

Execution of logic programs is performed in a goal-driven manner: a clause in the programs is initiated when a goal is given and returns the solutions to the goal. Logic programming languages make use of the unification operation, which is one of their basic functions. Nondeterminism is another basic feature of these languages; in particular, "don't-know nondeterminism" is required for OR-parallel Prolog, while "don't-care nondeterminism" is required for AND-parallel Prolog. The data flow model is also similar to logic programming languages such as OR-parallel and AND-parallel Prolog. The pograms written in OR-parallel or AND-parallel Prolog are compiled into data flow graphs.

Implementation of GHC

GHC was selected as a basic language of KL1 becouse it has clearer semantics and provides more efficient implementation than Concurrent Prolog, and it has more powerful descriptive power than PARLOG. GHC programs consist of guarded clauses such as:

H :- G1, G2, ..., Gm | B1, B2, ..., Bn.

where, H, Gi, and Bj are head, guard and body literals, respectively and "|" is called a commit operator. When a goal literal is given each definition clause is invoked and a semaphore flag shared among these clauses is created. Unification is attempted between the head literal and the given goal literal and if it succeeds then the guard literals are invoked as the new goal literals. Only the clause whose guard literals succed first can execute its body; i.e. the clause whose guard succeeds performs a test-and-set operation to the shared semaphore flag. If the result of this operation is also successful, the clause can execute its body; processing of the other clauses is terminated. Thus, one clause is exclusively selected for a given goal from all the clauses whose guards succeeded. There are several implementation schemes to support the guard mechanism in GHC [6]:

(1) Complete compilation scheme. All the unification directions are analysed in compilation time and codes are generated using unidirectional unification primitives. In this scheme the compiler is complicated.

(2) System number scheme. All the environments are managed by guard system numbers. A new guard system number is allocated, when a new definition is invoked and is restored to its parent number when the commit operator is executed. The guard numbers are associated with all the variables included in the invoked clauses and the invironment to which each variable belongs is compared with the current environment when unification to the variable is attempted.

(3) Pointer coloring scheme. The pointer coloring scheme distinguishes variables belonging to the goal literals from those belonging to the current guard by coloring. If unification is attempted between a goal variable and a variable in the invoked clause, the callee's variable is changed to a colored variable, which points to the original variables. If a colored variable is unified with a term, the instance bound to the variable is read before unification. The commit operator restores the colored variables to their original variables.

(4) Read-only tagging scheme. This scheme is an extension of the pointer coloring scheme, in which every variable has a tag specifying its read-only level. The read-only levels of the goal variables are incremented by one before the definitions are invoked, and decremented by one when the commit operator of each invoked clause succeeds.

Translation from GHC program into the data flow graph

We shall illustate the translation from a given GHC program into the corresponding data flow graph. Let us have a sample program written in GHC (Fig. 2). It is a list-append program which appends a list specified by the second argument of the head literal to the end of the list specified by the first argument. The resulting list is unified with the third argument. The compiled code is depicted in Fig. 3. The first statement of the compiled code specifies the procedure name "app" and its arguments "arg1", "arg2", and "arg3". The pocedure body is enclosed by "begin" and "end" statements and consists of three segments. The second and the third segment are compiled codes of the first and second source clause, respectively. The role of the first segment is to decide which of the subsequent segments should be invoked. Each body statement corresponds to a node in the data flow graph. The left side of the "=" operator specifies destination paths for the results of the instuction specified by the right side of the "=" operator. The "<<=" operator specifies a procedure "app" invocation macro. The "wait_instance" instruction reads the instance of the first goal argument which is passed along the input path "arg1". If the goal argument is an unbound variable, it is suspended until the variable is instantiated ("uarg1"). This operation will need remote access if the variable cells are distributed over the memory units in the system. Then, the "switch_by_type" instructions switch all the goal arguments according to the first argument "uarg1". If the first argument is nil, they put their left operands on their first destinations, otherwise (if "uarg1" is a list) they put their left operands on the second destinations. Thus, one of the subsequent segments is invoked exclusively. The "write_instance" instruction tries to unify its two operands and if one of them is a variable, it will instantiate the variable to another operand. In the second clause of the source program, there is a variable "Z1" in the body which does not appear in the guard. For such a variable, the "create_global_var" instruction creates a new variable cell and initializes it. Two body literals in the body of the second source clause will be executed in parallel. The first is the "write_instance" instruction and the other is the recursive invocation of the predicate. The "check_consistency" instruction tests their results whether they terminated successfully. The data flow program graph representation of the compiled code is shown in Fig. 4.

```
{ GHC SOURCE PROGRAM }

apnd([],Y,Z):-true|Z=Y.
apnd([H|X],Y,Z):-true|Z=[H|Z1],apnd(X,Y,Z).
```

Fig. 2 GHC source program.

```
{ COMPILED CODE }

ret<<=app(arg1,arg2,arg3).
begin.
 { CLAUSE INDEXING }
 uarg1=wait_instance(arg1).
 (arg1_L,arg1_R)=switch_by_type(uarg1,uarg1).
 (arg2_L,arg2_R)=switch_by_type(arg2,uarg1).
 (arg3_L,arg3_R)=switch_by_type(arg3,uarg1).
 (ret_1,ret_2)=switch_by_type(ret,uarg1).
 { COMPILED CODE OF THE FIRST CLAUSE }
 res1=write_instance(arg3_L,arg2_L).
 return(res1,ret_1).
 { COMPILED CODE OF THE SECOND CLAUSE }
 (p1,p2)=decompose_list(arg1_R).
 p3=create_global_var(arg1_R).
 p4=cons_list(p1,p3).
 p5=write_instance(arg3_R,p4).
 p6<<=app(p2,arg2_R,p3).
 res2=check_consistency(p5,p6).
 return(res2,ret_2).
end.
```

Fig. 3 Compiled code.

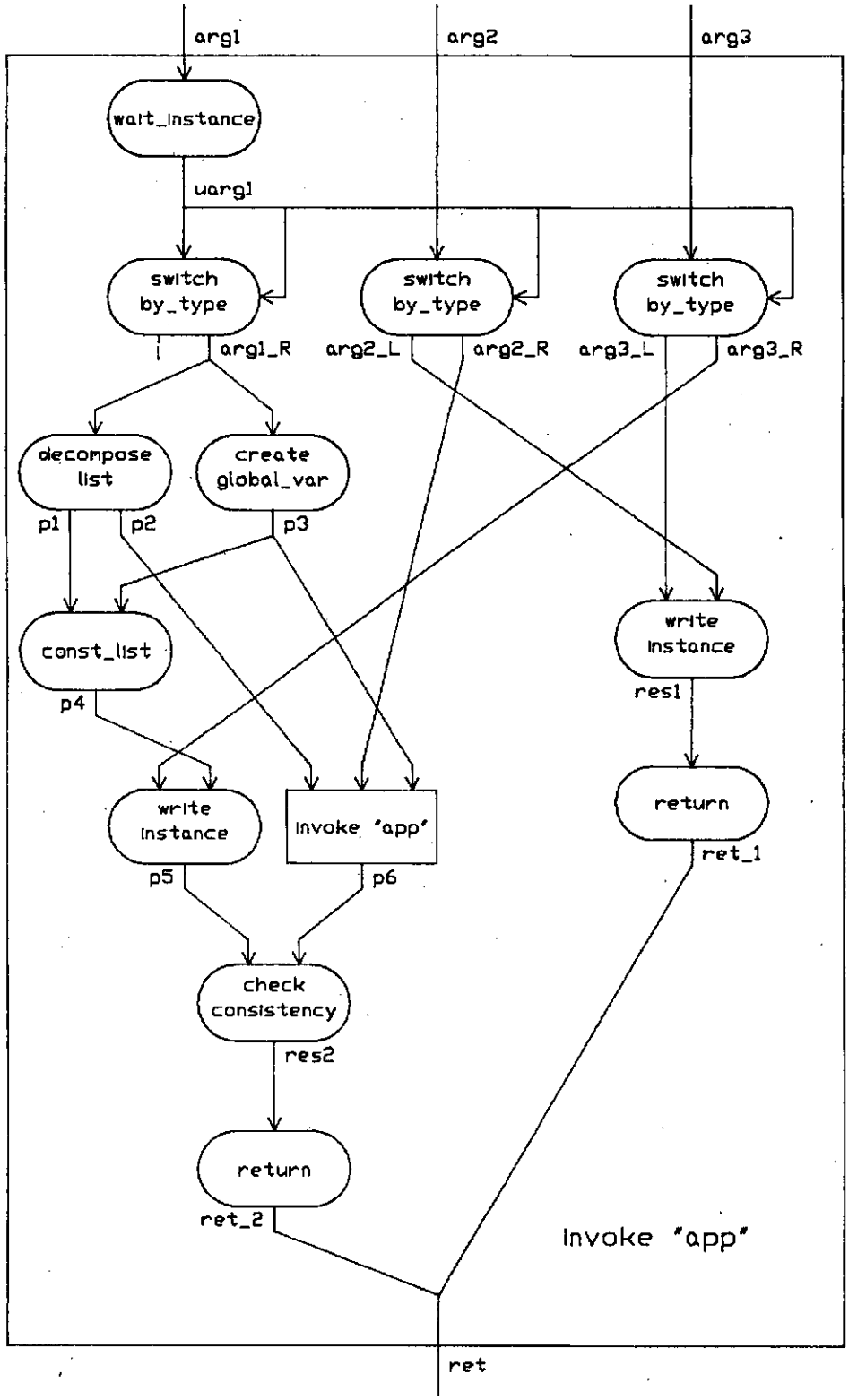


Fig. 4 Data flow graph representation of the compiled code.

Machine architecture

Abstract machine architecture. The machine can exploit OR and AND-parallelism as well as parallelism in unification. In head unification, if both literals consist of multiple arguments, or if both arguments are structured data, the unification of these arguments or their substructures can be executed in parallel. The machine is constructed from multiple processing elements and multiple structure memories interconnected by networks. The abstract machine architecture is shown by Fig. 5 [43].

Experimental machine. The experimental machine is constructed from multiple processing element modules (PEs) and multiple structure memory modules (SMs) interconnected through a hierarchial network as shown in Fig. 6 [5,6]. There are several hierarchy levels in the

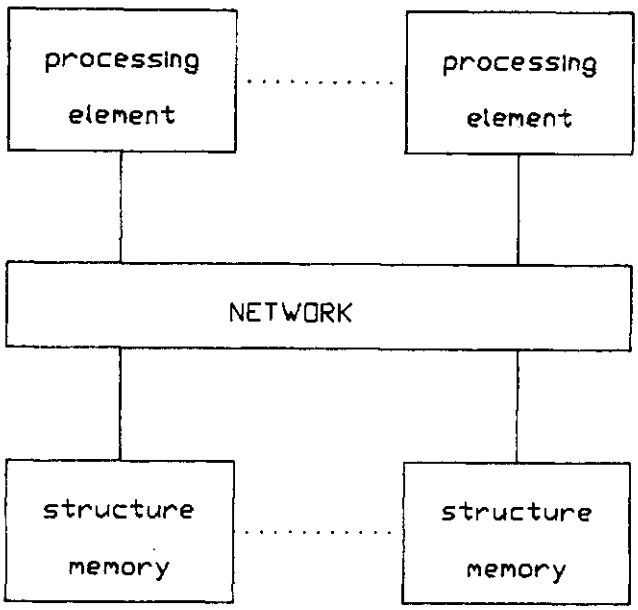


Fig. 5 The abstract machine architecture.

interconnection network. Each PE has its local bus. Four PEs and four SMs are interconnected by an inter-module network bus. A set of these modules is called a cluster. Several clusters are futher interconnected by an inter-cluster network bus. The hardware specification for these interconnection busses are the same, and they are called T-busses (token busses).

Actual implementation of the experimental machine includes two clusters and is currently being expanded to four clusters. Of these clusters, one is specialized, having one SM repleced by a host processor (VAX-11/730), which is used to initialize or monitor the system.

Packet formats. Each PE has several stages in order to implement pipelined or parallel execution. Packets transferred between these stages include result packets and executable instruction packets.

A resul packet (a token which is sent along the directed arc in the program graph), consists of three fields:

(1) The activity identifier (16 bits) specifies the invoked procedure instance name to which the result packet belong.

(2) The destination field (24 bits) specifies the address of the destination instruction (a node in the data flow graph) of the result packets. It also includes two bits for additional information; one specifies whether the destined instruction receives one or two operands, and the other specifies whether the operand is a left or right operand.

(3) The data field (32 bits) contains the operand data to be send to the instruction. The machine uses a tagging scheme, in which each operand has a value field (25 bits) and tag field (7 bits), which specifies the data type of the operand. If the operand is a structured data, the value field has a pointer to the structure memory (5-bit module number and a 20-bit local address in memory), and tag field is further divided into two subfield: a data type subfield, which specifies the data type of structure (i.e., list, vector, ...) and a attribute subfield. The attribute subfield contains a non-ground flag, which indicates whether the structure has any simple variables. The attribute subfield also contains a shared flag, which indicates whether the structure has any shared-type variables (i.e., shared variables, global variables, or read-only variables). The machine recognizes the tag field of the operand and transfers control to the appropriate firmware routine.

An executable instruction packet consists of five fields:

(1) The current instruction address (20 bits) indicates the instruction address to be executed and is used to obtain the destination

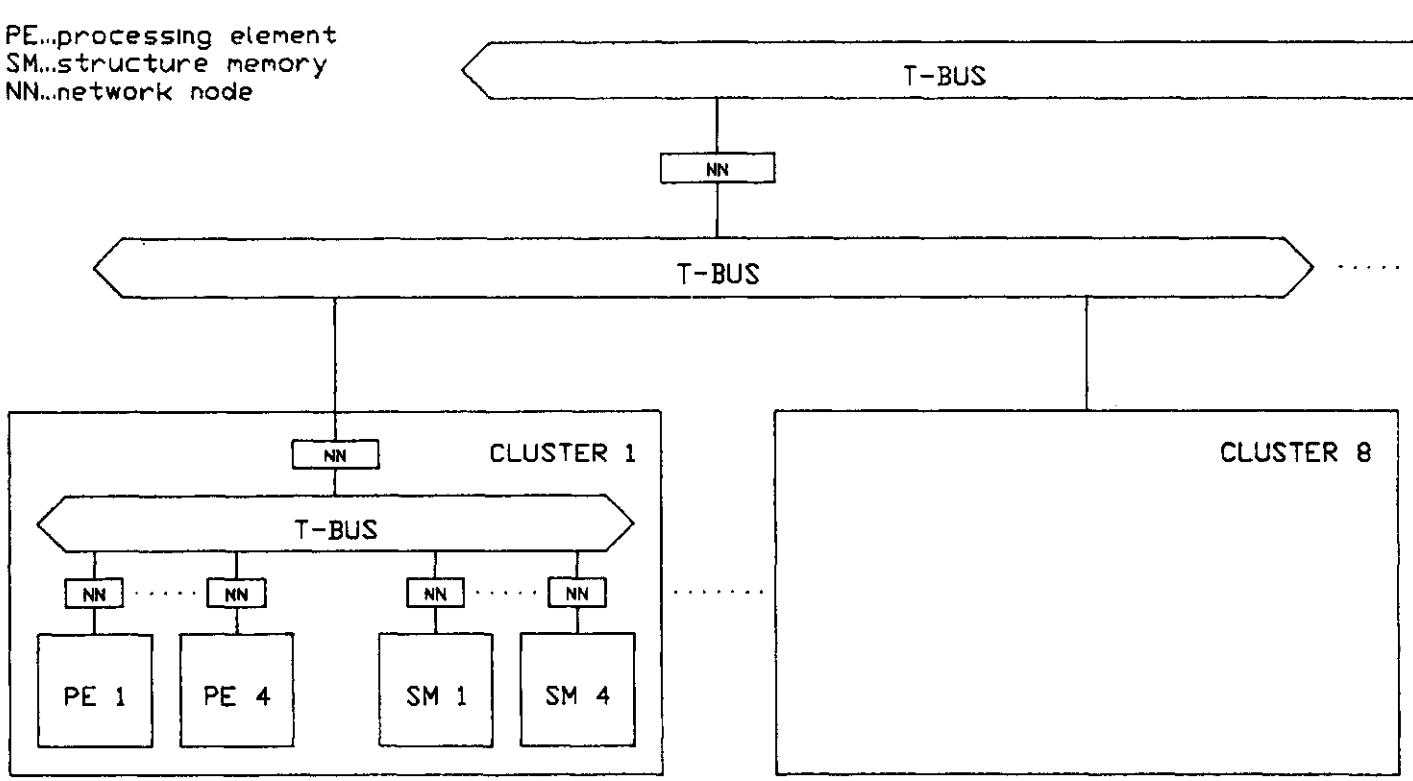


Fig. 6 Configuration of the experimental machine.

address from the destination specifier field as described below.

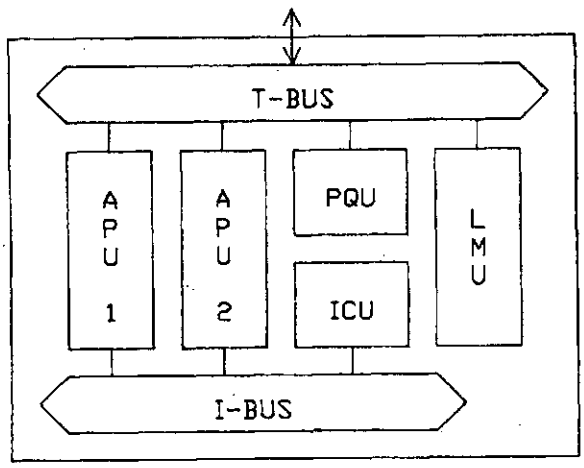
(2) The operation code field (8 bits) specifies the operation to be executed.

(3) The left operand (32 bits).

(4) The right operand (32 bits).

(5) The destination specifier field (48 bits) specifies the destination addresses of the results. There are two modes to specify the destination addresses in the destination specifier field; in the full destination mode the specifier field contains up to two destinations (each of them is 24-bit length), and in the short destination mode the specifier field contains up to four destinations, where each destination is of 12-bit length and contains the relative addresses from the current instruction. The relative addresses are added to the current instruction address to obtain the absolute addresses.

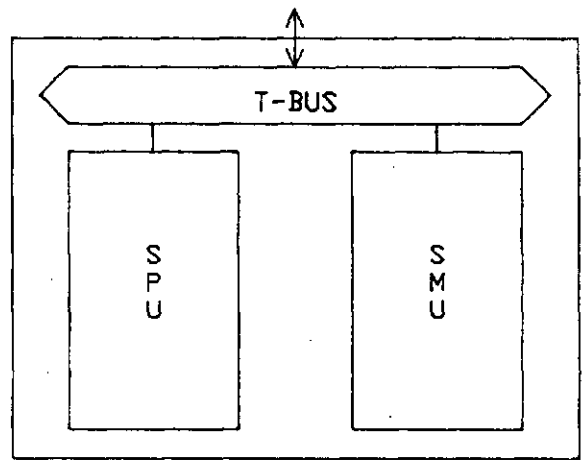
Processing element module. Fig. 7 depicts the configuration of each PE. The stages in a PE include a packed queue unit (PQU), an instruction control unit (ICU), several atomic processing units (APUs), and a network node (NN). These functional units have their own controllers and are operative in a pipelined manner. Packet transmission via T-bus is controlled by a NN, which has nine-to-one arbiter to arbitrate the requests from its lower level units and from its higher level bus. The PE has a local memory unit (LMU), which is used to store local data such as activity manegement information, and is shared and accessible from APUs. PQU is a FIFO queue memory to store the result packets from the T-bus. ICU receives the result packets from PQU and checks if the destination instructions are executable or not. An instruction is executable if it receives a single operand, or if the partner operand is already in the operand memory (OM) in the ICU when it receives two operands. In the later case, the ICU searches in its OM whether the partner operand exists or not. If it does, the partner is removed from the memory; otherwise, the result packet is stored in the OM. This searching is performed associatively by hardware hash using the identifier and the destination address as the key field. If the instruction is executable, the ICU fetches the instruction code in its instruction memory (IM) and constructs an executable instruction packet and sends the packet to the next stage, one of APUs via the instruction bus (I-bus). The APU interprets the instruction packets and sends result packets to the PQU in its PE or other PEs, or sends structure access comand packets to SMs via the token bus.



APU...atomic processing unit
PQU...packet queue unit
ICU...instruction control unit
LMU...local memory unit
T-BUS...token bus
I-BUS...instruction bus

Fig. 7 Configuration of a processing element.

Structure memory module. The SMs are responsible for the structure access commands, perform structure manipulation operations, and return results to the destination specified by the commands. Each SM consists of an structure processing unit (SPU) and structure memory unit (SMU) for storing the structured data (Fig. 8). The SPUs receive the structure manipulation commands from the APUs and interpret them. If the commands need the responses, new result packets are created and sent back to the PEs. Such commands include read commands, memory allocation commands, and so on.



SPU...structure processing unit
SMU...structure memory unit
T-BUS...token bus

Fig. 8 Configuration of a structure memory.

The specification and typical processing times of the various units are given in Table 1 and Table 2, respectively.

Table 1 Specification of the units.

unit		specification*
PE	PQU	FIFO size: 16Kw x 86b (16K tokens)
	ICU	IM size: 96Kw x 59b (96K instr.)
		OM size: 32Kw x 64b (32K tokens)
	APU	micro store: 1Kw x 32b ROM
		7Kw x 32b RAM
	LMU	memory size: 512Kw x 32b
SM	SPU	micro store: 1Kw x 32b ROM
		7Kw x 32b RAM
	SMU	memory size: 1Mw x 34b (data, tags)
		512Kw x 10b (ref. count)
NN		FIFO size: 64w x 86b (64 tokens)

* w .. word b .. bit

Table 2 Typical processing times of the units.

unit		item	time*
PE	PQU	packet receive	2
		delay in queue	8
	ICU	single operand instruction	2
		two operand instruction	
		(on arrival of 1st operand)	3
		two operand instruction	
		(on arrival of 2nd operand)	5
	APU	"copy" instruction	3
		packet creation	2
SM		SM-read operation	2
		SM-write operation	2
NN		packet receive	2
		packet send	8

* in machine cycles

5 Concluding points

Striking progress in computer technology has given us single-chip computers whose processing power far exceeds that of the first-generation computers. There are also various high-level languages, operating systems, and data-base systems. As a result, programs for almost any kind of application can be written, provided that their algorithms can explicitly be described. This means that computers can replace people in many areas because of their high-speed processing and large memory capability. However, there remain many application fields with hard-to-solve problems. One such is the knowledge-information processing field, where FGCS are expected to play an important role.

A machine to cope with knowledge-information processing should support extensive storage of data and high-speed inference using the data. Up to now, inference procesing has involved implementing functional and logic programming languages on conventional sequential computers. However, the need for processing power of new applications in knowledge-information processing may exceed the capabilities of sequential computers.

The architecture of parallel inference mechine makes it a possible candidate for coping with such processing requirements. Computer architectures proposed for parallel inference machines include the high-level language machine [10] as well as the data flow machine.

6 Literature

[1] P. Bishop, Fifth generation computers: concepts, implementations and uses, (Ellis Horwood, 1986).

[2] K. Furukawa, T. Yokoi, Basic software system, in: ICOT, ed., Proc. Int'l Conf. Fifth Gener. Comp. Systems 1984, (North-Holland, 1984) 37-57.

[3] L. O. Hertzberger, R. P. Van De Riet, Progress in the fifth generation inference architectures, Future Generation Computer Systems 1 (2) (1984) 93-102.

[4] N. Ito, H. Shimizu, M. Kishi, E. Kuno, K. Rokusawa, Data-flow based execution mechanisms of Parallel and Concurrent Prolog, New Generation Computing 3 (3) (1985) 15-41.

[5] N. Ito, M. Kishi, E. Kuno, K. Rokusawa, The dataflow-based parallel inference machine to support two basic languages in KL1, in: J.V. Woods, ed., Fifth Gener. Comp. Architectures, (North-Holland, 1986) 123-145.

[6] N. Ito, M. Sato, E. Kuno, K. Rokusawa, The architecture and preliminary evaluation results of the experimental parallel inference machine PIM-D, Proc. 13th Int'l Symp. Comp. Arch., (IEEE, 1986) 149-156.

[7] T. Moto-oka, H. Tanaka, H. Aida, K. Hirata, T. Murayama, The architecture of a parallel inference engine - PIE, in: ICOT, ed., Proc. Int'l Conf. Fifth Gener. Comp. Systhems 1984, (North-Holland, 1984) 479-488.

[8] K. Murakami, T. Kakuta, R. Onai, Architectures and hardware systems: parallel inference machine and knowledge base machine, in: ICOT, ed., Proc. Int'l Conf. Fifth Gener. Comp. Systems 1984, (North-Holland, 1984) 18-36.

[9] K. Murakami, T. Kakuta, R. Onai, N. Ito, Research on parallel machine architecture for fifth-generation computer systems, Computer 18 (6) (1985) 76-92.

[10] H. Tanaka, A parallel inference machine, Computer 19 (5) (1986) 48-54.

PODATKOVNO PRETOKOVNI PARALELNI STROJ ZA SKLEPANJE. V članku je predstavljen paralelni stroj za sklepanje, ki temelji na podatkovno pretokovnem izvrševanju logičnih programov. Stroj podpira izvrševanje logičnih programov, zapisanih v OR ali AND-paralelnem Prologu (Parallel Prolog, PARLOG, Concurrent Prolog, GHC). Takšni programi se prevedejo v podatkovno pretokovne programske grafe, ki ustrezajo strojnemu jeziku. Podan je primer transformacije programa, zapisanega v jeziku GHC, v ustrezni podatkovno pretokovni programski graf. Arhitektura stroja obsega procesne elemente ter strukturne pomnilnike, ki jih povezuje hierarhična mreža. Procesni elementi izvršujejo dele programskega grafa sočasno, pri čemer si delijo podatke, zapisane v strukturnih pomnilnikih. Podan so tudi prostorske in časovne zahteve posameznih komponent arhitekture.

ONE SOLUTION OF THE BUSLINE CREW SCHEDULING PROBLEM

Miroslav B. Jocković
Institute »Mihailo Pupin«, Belgrade
Dušan M. Velašević
Faculty of Electrical Engineering, Belgrade

UDK 681.3:656.132

SUMMARY: The paper considers the problem of minimising the number of crews to carry out a specified number of trips. A trip represents in this case a movement between the two adjacent bus stops (each bus stop is treated as a timing point). Two procedures are presented for establishing a number of crews in a period of 24 hours which can handle all trips on one bus route. The concept of finding an optimum was not applied in solving this problem. Instead, smaller sets of trips were extracted from the whole number and the best possible combinations of trips in each set are established. This gave excellent practical results for the test cases.

SADRZAJ: Rad razmatra problem minimiziranja broja posada pomoću kojih se opslužuje potreban broj vožnji. Vožnja u ovom slučaju pretstavlja kretanje izmedju dve susedne stanice autobusa (svaka zaustavna tačka se tretira kao vremenska tačka). Prikazane su dve procedure za izračunavanje broja posada u periodu od 24 časa koje uključuju sve vožnje na jednoj autobuskoj liniji. Koncept nalaženja apsolutnog optimuma nije primenjen pri rešavanju ovoga problema. Umesto toga, mali skupovi vožnji su izdvojeni iz celog skupa i iz njih izračunavane najbolje kombinacije vožnji. Ovaj postupak je dao odlične praktične rezultate pri testiranju.

1. INTRODUCTION

The problem of calculating the necessary number of crews was solved using different techniques (Ref.1,2,3, 4). The heuristic method is employed to minimise the number of crews serving a large number of trips, while smaller sets of trips allow application of the deterministic method to find the absolute minimum of the number of crews required. So, this problem was never solved in a unique manner for any number of trips.

The intention here is to suggest a new method for solving the crew scheduling problem, and to trigger off further research for development of this model. The solution which will be presented does not require special mathematical software (linear programming, etc.) and it is economical in hardware requirement and running time.

2. DEFINITIONS

Definition 1: A trip is one-way movement of a bus along a route between the two terminal points. However, in the current context each of the bus stops will be treated as a timing point and the term "trip" will indicate a one-way movement between the adjacent timing points.

We also define the variables for a trip:

- point of departure — BPV,
- point of arrival — BPN,
- departure time — ABF,
- arrival time — ANK,
- direction of the trip — R,
- live running time — FZ,
- "link" time (required to move the bus from the arrival point to the departure point) — UMZT,
- preparation time — ZVG,
- layover time in case of an early arrival — MWZ.

Definition 2: A duty consists of a set of trips executed by one crew during one day.

A duty is defined by:

- working hours (maximum allowed) — ARBMAX,
- crew preparation time — ARBVOR,
- clear time — ARBABS,
- a selected subset of trips — $a_{i1}, a_{i2}, \ldots, a_{in}$,
- number of trips — n,
- meal break — T_p.

A sequence of trips may constitute a duty if it satisfies the requirement and restrictions above and at least one of the conditions (1^o and 2^o below) of the meal breaks T_p to which the crew is entitled.

1^o The meal break T_p during a duty must not be less than 1/z of the sum of trip times (e.g. z=6)

2^o If the above condition (1^o) is not satisfied then at least the following must apply:

- one meal break of P_1 min, or
- two meal breaks of P_2 min, or
- three meal breaks of P_3 min.

From experience, these values are P_1=30 min, P_2= = 20 min and P_3=15 min. If a minimum number of breaks exists in a duty (as in case 2^o), then the time distribution of breaks must be as follows:

- the time from the start of a duty to the start of meal break P_1 and from the end of meal break P_1 to the end of the duty must not exceed 270 (4:30) min,
- the time from the start of a duty to the start of the second meal break P_2 and from the end of the first

meal break P_2 to the end of the duty must not exceed 270 min,

- the time from the start of a duty to the start of the third meal break P_3 and from the end of the first meal break P_3 to the end of the duty must not exceed 270 min.

Meal break may be taken at any stop on route. The shortest possible meal break is fixed to 10 min. A meal break less than 10 min is ignored.

The problem is to find a simple algorithm for determining the minimum number of crews (or value close to the minimum) on one bus line. In other words, to distribute the total number of trips N on one line in one day to the minimum number of duties M, so that each duty satisfies all the previously mentioned restrictions. Also, it should allow for the fact that a new trip of the same duty cannot start before the previous one has finished (this will be referred to as "trip combination").

The number of buses servicing one line has not been considered in the analysis. The number of buses represents a separate problem.

3. ALGORITHM

This section considers an algorithm which can give a short-cut solution SC and a general solution GS which is more complex but more accurate. They differ in the number of solutions which can be obtained from one execution of the algorithm and the way it is implemented on a computer. Two methods can be applied for both solutions. The methods differ from each other in the sequence of selecting duties; method F calculates duties in an ascending time flow, while method B does the reverse, i.e. in a descending sequence. Method F will be explained only since there is an analogy between F and B.

The general solution GS

A. Sort all the trips using departure time ABF; resulting in a sorted set of trips

$$S_N = \{a_i;\ i=1, N\}.$$

B. Extract all the trips from the set S_N which satisfy the inequality

$$ABF(a_1) < ANK(a_j) \leq ARBMAX - ARBVOR - ARBABS + ABF(a_1),$$

$$(a_j;\ j=1, r11) \qquad (1)$$

and let them constitute the trip subset $S_{r11} = \{\tilde{a}_i;\ i=1, r11\}$ from r11 members, with member $\tilde{a}_1=a_1$ in the first position. Trips from this subset only can take part in the selection of the first duties.

C. Combine all the trips from subset S_{r11} in accordance with the restrictions from section 2, so that each combination has in its first position the trip a_1. Combinations with the highest time utilisation factor (the ratio of the sum of trip times in a duty and the duration of the duty) are stored as the best solutions for the first crew, hence creating a table $C = ||c_{ij}||_{r11xr11}$.

Element c_{ij}, for i<j, assumes a meal break p_{ij} between $\tilde{a}_i$-th and $\tilde{a}_j$-th trip from the subset S_{r11}. If the trips $\tilde{a}_i$ and $\tilde{a}_j$ are simultaneous, then $c_{ij}=0$. The symmetrical element c_{ji} assumes the sum of times f_{ij} of executing the $\tilde{a}_i$-th trip and moving the bus to the starting point of the new trip $\tilde{a}_j$ at the same station. If $\tilde{a}_i$ and $\tilde{a}_j$ cannot be combined, then $f_{ij}=0$.

The values of p_{ij}, and f_{ij} are

$$p_{ij}=ABF(\tilde{a}_j)-ABF(\tilde{a}_i)-FZ(\tilde{a}_i)-UMZT(\tilde{a}_i,\tilde{a}_j)-ZVG(\tilde{a}_i)-G(BPN(\tilde{a}_i),\ BPV(\tilde{a}_j))$$

$$f_{ij}=FZ(\tilde{a}_i) + UMZT(\tilde{a}_i,\tilde{a}_j) \qquad (2)$$

G is a table of the times between stops. Element g_{ij} of this table represents the time required to travel the distance between the arrival point $BPN(\tilde{a}_i)$ of the trip $\tilde{a}_i$ and the departure point $BVP(\tilde{a}_j)$ of the trip $\tilde{a}_j$. The user specifies a time interval T_c in which trips can be combined and this results that the table C contains only those members of c_{ij} and c_{ji}, which satisfy the inequality for trips $\tilde{a}_i$ and $\tilde{a}_j$.

$$ABF(\tilde{a}_j) < ABF(\tilde{a}_i) + T_c, \quad i<j \qquad (3)$$

This restriction and condition for combining trips results in a sparse table C, greatly reducing the number of combinations. The solutions for the first crew are found via the subscripts of the table elements which are greater than zero and are located above the main diagonal. E.g., let us examine a series of this table elements: $C_{1,3}$; $C_{3,8}$; $C_{8,13}$; $C_{13,27}$; $C_{27,41}$; $C_{41,57}$. Their subscripts point to the trip combination a_1, a_3, a_8, a_{13}, a_{27}, a_{41}, a_{57}. When we analyse all the trip chains, we get S_{11} solutions for the first crew. Let us label them as $K_1(a_{11})$, $K_2(a_{11})$,...,$K_{s11}(a_{11})$. The parametar a_{11} means that these potential solutions for the first crew have been selected from the first extracted subset s_{r11}; that a_{11} is the first trip in all solutions (in the first execution of step C in the algorithm $a_{11}=a_1$). Hence, it follows that

$$\forall(K_i(a_{11}),K_j(a_{11}))\in S_{r11} \text{ is } K_i(a_{11})\wedge K_j(a_{11})\neq 0$$

$$i\neq j; \quad i,j\in\{1,S_{11}\}. \qquad (4)$$

Each calculated duty which has a time utilisation factor greater then the preset thershold time utilisation factor ρ_p, is entered in one row of the table H. Let us assume that v11 solutions satisfy this condition. For instance, let us say that after the first execution of steps B and C of the algorithm we have v11=3 solutions for the first crew, i.e.

$K_1(a_1)$: $a_1\ a_3\ a_4\ a_6\ a_7\ a_9\ a_{10}a_{11}a_{13}a_{14}$ 0...0 a_2 10 1 ρ_1
$K_2(a_1)$: $a_1\ a_2\ a_4\ a_6\ a_8\ a_9\ a_{11}a_{13}a_{14}$ 0.....0 a_3 9 1 ρ_2
$K_3(a_1)$: $a_1\ a_2\ a_3a_5a_7a_8a_{10}a_{11}a_{14}a_{15}$ 0.....0 a_4 10 1 ρ_3

This shows that the number of trips for three duties is $n_1(a_1)=10$, $n_2(a_1)=9$ and $n_3(a_1)=10$, with the first unrelated trips being $a_{p1}(a_1)=a_2$, $a_{p2}(a_1)=a_3$ and $a_{p3}(a_1)=a_4$. Time utilisation factors are ρ_1, ρ_2 and ρ_3, respectively.

In general case, after first execution the steps B and C we have $K_1(a_{11})$, $K_2(a_{11}),\ldots,K_{v11}(a_{11})$ solutions for the duties with $n_1(a_{11}),n_2(a_{11}),\ldots,n_{v11}(a_{11})$ trips in each, and the unrelated trips in these solutions being $a_{p1}(a_{11}),a_{p2}(a_{11}),\ldots,a_{pv11}(a_{11})$, respectively.

D. Create sets $S_{N-n_1(a_{11})},S_{N-n_2(a_{11})},\ldots,S_{N-n_{v11}(a_{11})}$ without the trips included in the duties $K_1(a_{11})$, $K_2(a_{11}),\ldots,K_{v11}(a_{11})$, respectively.

Extract subsets S_{r21}, $S_{r22},\ldots,S_{r2v11}$ from these sets by executing step B of the algorithm v11 times.

Execute step C once for each of these subsets and the best second duties should be extracted.

E.g., the solutions for the first and the second duties whixh can be combined would be

$$\begin{aligned} &K_1(a_{11}) + (K_1(a_{21}),\ K_2(a_{21}),\ldots,\ K_{v21}(a_{21}))\\ &K_2(a_{11}) + (K_1(a_{22}),\ K_2(a_{22}),\ldots,\ K_{v22}(a_{22}))\\ &\quad\cdot\\ &\quad\cdot\\ &K_{v11}(a_{11})+(K_1(a_{2v11}),\ K_2(a_{2v11}),\ldots,\ K_{v2v11}(a_{2v11})) \end{aligned} \qquad (5)$$

The number of solutions is high, hence the user is allowed to select only those pairs of duties which have a high time utilisation factor, or a high number of trips in duties. If we wish to retain the same number of solutions as in the selection for the first crew, and assuming that the values in brackets in relation (5) are sorted in a descending sequence of the time utilisation factors, the set of solutions for two duties would be $K_1(a_{11})+K_1(a_{21})$, $K_2(a_{11})+K_1(a_{22}),\ldots,$ $K_{v11}(a_{11})+K_1(a_{2v11})$. These solutions satisfy the condition for combining duties represented by relation

$$(K_i(a_{11}) \wedge K_1(a_{2i}))=0; \qquad i\in\{1,v11\} \qquad (6)$$

The unrelated trips of the first duties are the first trips of the second duties, i.e.

$a_{p1}(a_{11})=a_{21}$; $a_{p2}(a_{11})=a_{22}$;...;$a_{pv11}(a_{11})=a_{2v11}$.

This explains how a set of solutions for two crews is created. The procedure is repeated until each row of the table H contains all the trips from set S_N. If a greater number of solutions is obtained (rows in the table) which contain the same minimum values of the number of crews M, then the final choice is made by the user.

The short-cut solution SC

This is a simplified version of the algorithm providing a single solution of the required number of crews. Because of simplicity and short running time on the computer, this solution is of greater practical value.

a) Sort all trips of the set S_N by ABF parameter as in A.

b) Extract the $S_{r11}=\{\tilde{a}_i;\ i=1,\ r11\}$ subset of trips from the S_N set according to the inequality condition (1).

c) Determine the combination of trips from subset S_{r11} which has the highest time utilisation factor as in C. Let this be the combination $K_1(a_{11})$ with $n_1(a_{11})$ trips. Let us declare this combination as the first duty.

d) Revise the set S_N extracting all the trips which make up the first duty.

We now obtain a new set $S_{N-n_1(a_{11})}=\{\tilde{a}_i;\ i=1,\ N-n_1(a_{11})\}$. Branch to step b of the algorithm.

After k-th pass through steps b, c and d of the algorithm, duties $K_1(a_{11})$, $K_1(a_{21}),\ldots,$ $K_1(a_{k1})$ have been established from subsets S_{r11}, $S_{r21},\ldots,$ S_{rk1}, with $n_1(a_{11})$, $n_1(a_{21}),\ldots,$ $n_1(a_{k1})$ trips per duty, respectively. Test whether the remaining trip set $S_{N-n_1(a_{11})-n_1(a_{21})-\ldots-n_1(a_{k1})}$ is empty. If it is, the algorithm terminates and the total number of crews is M=K, otherwise substitute k+1→k and branch to b.

4. REDUCING THE NUMBER OF CREWS

If we use alternative GS of the algorithm, we can end up with a large number of solutions. Amongst that set of solutions there is a great chance that we obtain the solution with the minimum number of crews. However, the general solution is economically justifiable only if it is applied to smaller sets of trips or if few solutions are required. We shall now explain the procedure for reducing the number of crews which will also improve the result obtained from the short cut colution.

We proved experimentally that in applying methods F and B of the short cut solution on a set of N trips, we get two independent solutions R_1, $R_2,\ldots,$ R_p and L_1, $L_2,\ldots,$ L_{q-1}, L_q. The number of crews p and q are different in the general case. It becomes evident that there is a tendency for the time utilisation factor of the duties to decrease as the subscript i increases for the row R and vice versa for L. This statement is not always valid for two consecutive duties, but it can be considered as such when the whole set of duties is analysed. This becomes more evident when numerous trips are taking place simultaneously. Also the quality of each duty is dependent on the size of the subset from which it was selected, the mean trip duration, time distribution of trips and on the selection of the previous duties.

Here, we shall try to reduce the number of crews as obtained from solution R and L, to get a series of

m crews where m can be represented by

$$m < \min(p,q) \tag{7}$$

The following procedure is recommended. Let us consider the relationships which may occur between duty R_1 and L_1.

1^0 $L_1 \subset R_1$; all trips of duty L_1 are included in duty R_1.

Based on 1^0 transform series R and L into the following form

$$\begin{array}{l} R_1, \mid R_2, \ldots, R_{p-1}, R_p \\ R_1, \mid L_2^{(-)}, L_3^{(-)}, \ldots, L_{q-1}, L_q \end{array}$$

The character (-) appears, for example, in duties L_2 and L_3, and means that certain trips have been extracted from these duties, as they have been included in duty R_1. In further steps, the series which are being compared are on the right hand side of the vertical line.

2^0 $L_1 \supset R_1$; all trips of duty R_1 are represented in duty L_1. The procedure is the same as in 1^0. We get

$$\begin{array}{l} L_1, \mid R_2^{(-)}, R_3^{(-)}, \ldots, R_{p-1}, R_p \\ L_1, \mid L_2, L_3, \ldots, L_{q-1}, L_q \end{array}$$

3^0 $R_1 \leftrightarrow L_1$; certain trips from duty R_1 are not included in duty L_1 and vice versa. Based on the time utilisation factor we select, for example, duty R_1. The series now becomes

$$\begin{array}{l} R_1, \mid R_2, R_3, \ldots, R_{p-1}, R_p \\ R_1, \mid L_2^{(\mp)}, L_3^{(\mp)}, \ldots, L_{q-1}, L_q \end{array}$$

Characters + and - show that some trips have been extracted and added to duties L_2 and L_3, for example. For the new values $L_2^{(\mp)}$ and $L_3^{(\mp)}$ it is not evident whether they can represent duties. However, based on the assumption of monotony of the series R and L, R_2 will assign in the second place of the L series, hence

$$\begin{array}{l} R_1, R_2, \mid \ldots, R_{p-1}, R_p \\ R_1, R_2, \mid L_3^{(\mp\mp)}, \ldots, L_{q-1}, L_q \end{array}$$

By analogy, after two steps, going from right to left the series becomes

$$\begin{array}{lll} R_1, R_2, & \ldots, R_{p-2}^{(\mp\mp)}, & L_{q-1}, L_q \\ R_1, R_2, & L_3^{(\mp\mp)}, \ldots, & L_{q-1}, L_q \end{array}$$

Generalising the procedure we arrive at a subscript i for which

$$\rho_R^{(1)} + \rho_R^{(2)} + \ldots + \rho_R^{(i)} > \rho_L^{(1)} + \rho_L^{(2)} + \ldots + \rho_L^{(i)} \tag{8}$$

is valid. Here ρ_R and ρ_L are time utilisation factors of the series R and L. If the standard deviation from trip duration is small, we can also suppose that

$$n_R^{(1)} + n_R^{(2)} + \ldots + n_R^{(i)} > n_L^{(1)} + n_L^{(2)} + \ldots + n_L^{(i)} \tag{9}$$

(a simplified notation was used, hence $n_R^{(i)} = n_1(a_{i1})$).

We repeat the procedure starting from the end of series R and L until we reach subscript j for which

$$\rho_R^{(p-j+1)} + \rho_R^{(p-j+2)} + \ldots + \rho_R^{(p)} < \rho_L^{(q-j+1)} + \rho_L^{(q-j+2)} + \ldots + \rho_L^{(q)} \tag{10}$$

and

$$n_R^{(p-j+1)} + n_R^{(p-j+2)} + \ldots + n_R^{(p)} < n_L^{(q-j+1)} + n_L^{(q-j+2)} + \ldots + n_L^{(q)} \tag{11}$$

are valid.

Hence, i+j duties with high utilisation factors are taken as those finally selected and we extract all their trips from further reduction. These are $R_1, R_2, \ldots, R_i, L_{q-j+1}, \ldots, L_{q-1}, L_q$. The remaining duties are determined by executing the short cut version of the algorithm on the remaining $N^{(1)}$ trips

$$N^{(1)} = N - (n_R^{(1)} + n_R^{(2)} + \ldots + n_R^{(i)}) - (n_L^{(q-j+2)} + \ldots + n_L^{(q)}) \tag{12}$$

and then we repeat the reduction procedure with an interactive control from the terminal. After reaching certain values i and j, the selection of duty R_{i+1} may have a negative effect on L_{j-1} and vice versa. In such a case the general solution of the algorithm is recommended.

5. A NUMERICAL EXAMPLE

In order to test the algorithm, a program was written which processed a set of N=227 trips. Variable length trips, between 5 stops of a route were taken, with a mean trip duration of 35.45 min and with a standard deviation of 8.91 min. The short cut solution gave a result of M=27 crews. By reduction, the total number of crews was reduced to 26. Using a Honeywell, level 6 computer, and 23 K words of store, the processing took 12 min.

6. CONCLUSION

During the analysis stage of this problem it was quite evident that it is impossible to calculate, in a short time on a computer, all distributions of N trips in M crews allowing for a set of restrictions. Hence, it was impossible to select a solution with a minimum value for M. However, an algorithm was developed, where the number of combinations under investigation is reduced to a minimum by searching through a significantly smaller trip subsets and extracting the best duty from each.

The weaknesses evident in selecting duties from subsets with a small number of elements or in selecting the last duties, can be eliminated by reduction. This procedure also reorganises duties. On the other hand, the general solution represents a set of simplified solutions. Also, it gives the user the flexibility in

selecting solutions with a high time utilisation factor in accordance with the optimisation level it wishes to attain and depending on the computer resources available.

It would be interesting for further research to apply this algorithm to the inseparable strings of trips. In this case, a string would mean a sequence of trips with a high percentage of live running time.

ACKNOWLEDGEMENT

The author would like to express his appreciation for the many helpful comments and suggestions in defining the problem received from Mr. Miodrag Mojsilović, from Hamburger Hochbahn.

REFERENCES

|1| Jerrod Rubin, A. Technique for the Solution of Massive Set Covering Problems, with Application to Airline Crew Scheduling, Trans.Sci.,1973, 7, No 3.

|2| J.P.Arabeyre, J.Fearnley, F.C. Steiger, W.Teather, The Airline Crew Scheduling Problem, Trans.Sci., 1969,3, No 2.

|3| A.P.Artinov, V.V.Skaleckij, Avtomatizacija i procesov planirovanija i upravlenija transportnimi sistemami, Akademija nauk SSSR, Moskva, 1981.

|4| R.A. Chapman, J.F. Michel, Modeling the tendency of buses to form pairs, Trans.Sci., 1978, 12, No.2.

|5| A.G.Wilson, Entropy in urban and regional modelling, 1970, London.

UDK 681.3.02

Saša Prešern
Iskra Delta and
Jozef Stefan Institute, Ljubljana

ABSTRACT. This paper is a selected survey of parallel computer systems. A classification of parallel computers is given and some most attractive architectures are discussed. Special attention is paid to massively parallel processors. The organization and interconnection structure of multiprocessor systems is given. By analyseing a trend of research in parallel computer systems over last 10 years some predictions are given about individual features which will probably have great influence on future parallel computer systems. An extensive survey of references in parallel computer systems is given.

IZBOR IN PREGLED PARALELNIH RAČUNALNIŠKIH SISTEMOV. Članek podaja izbor in pregled paralelnih računalniških sistemov. Narejena je klasifikacija paralelnih računalnikov in opis nekaterih najbolj zanimivih arhitektur. Podana je organizacija multiprocesorjev in opisane so različne povezovalne strukture med procesorji ter pomnilniki v posameznih sistemih. Analiza trenda raziskav paralelnih računalniških sistemov v zadnjem desetletju omogoča izločitev posameznih značilnosti, ki bodo predvidoma močno vplivale na razvoj bodočih paralelnih računalniških sistemov. V bibliografiji je priložen obširen pregled referenc za paralelne računalniške sisteme.

1. INTRODUCTION - EVERYBODY MAKES IT PARALLEL

A few years ago all high developed counties in the world have started projects in developing a parallel computer system. All these projects were financially supported by governments. Many companies and research institutes also started research projects on parallel systems. The falling price of microcomputers and VLSI facilities on universities has encouraged many universities to design and to build parallel computer architectures based on linking many microprocessors or specially designed VLSI chips together to work on one job.

Development of a parallel computer is an extremely difficult task which includes:

- development a new concept of parallel computer architecture,

- design of an operating system that supports parallel architecture,

- transformation of traditional sequential application programs to parallel programs either by preprocessor or by a parallel programming language.

We see that by switching from SISD (single instruction single data) machines to MIMD (multiple instruction multiple data) machines one can not simply upgrade an existing SISD computer system but one is faced with problems which are conceptually new. Research and development of a parallel computer system requires a very strong research which often includes:

- more than 100 specialists,
- a bilion dollar financial support,
- research and development phase which lasts several years.

Government financial support is only a fraction of the whole finances which are devoted to projects in parallel computing. Strategy makers in most companies are familiar with market research studies which predict that parallel processing machines will take about 50 percent of the market in high-performance computers by 1990.

2. CLASSIFICATION OF PARALLEL SYSTEMS

Parallel computers are usually divided into three architectural configurations:

- SIMD pipelined computers
  * early vector processors,
  * attached processors,
  * recent vector processors,
  * other vector processors,

- SIMD array processors,

- MIMD parallel processors,
  * massively parallel processors,
  * small scale parallel systems.

Another grouping in possible as for example classification according to distribution of local and global memory into tightly and loosely coupled parallel systems or classification according to application possibilities into general purpose or special

purpose computers.

Many existing computers are now using several parallel approaches. Parallelism in pipeline computers is performed by overlapping computations and is therefore temporal parallelism. Parallelism in array processors is performed by multiple synchronized ALUs and is therefore spatial parallelism. Parallelism in multiprocessor systems is performed by a set of processors with shared resources which work in asynchronous mode.

The list of projects in parallel computing is getting longer every day. By comparing the architectural approach in different projects we see that the computer scene in parallel computer systems is particularly varied. It is difficult to classify parallel computers, but helpful in order to concentrate on similarities and differences between the computer architectures. Because parallel computers are using several different architectural principles one might argue a proposed classification.

Some of described computer are "paper machines" that have been studied theoretically and by simulation, but have not been build. Many of this projects were funded by government agencies, but some of them are industry projects (IBM, Burroughs, CDC,...).

There follows an alphabetic list of the parallel computer systems or projects, each with the name of the chief architect and host institution. A list of references dealing with each project is also given. The most interesting architectures are briefly described. The list of parallel computers is grouped according to upper classification.

SIMD PIPELINE COMPUTERS

EARLY VECTOR PROCESSORS

BVM (Boolean Vector Machine), Robert A. Wagner, Duke University, North Carolina.
This is a collection of 1-bit processing elements connected as a hypercube with rings at each corner, using the Cube-Connected-Cycle topology.

STAR-100, Control Data Corporation .
The design of Star started in 1965 and was delivered in 1973. This is a processor with two nonhomogeneous arithmetic pipelines. (HWA85, LIN82, PUR74).

TI ASC, (Texas Instruments Advanced Scientific Computer), Texas Instrument.
This machine uses 1 to 4 homogeneous pipelines and was delivered in 1972. (HWA85, KOG81).

ATTACHED PIPELINE PROCESSORS

CSPI MAXIM/64, (CSP Inc., BILLERICA, Massachusetts).
Maxim/64 in a minimal configuration includes a 16 slot chassis, a 64-bit floating point array processor, 16 Mbytes of data memory and Micro VAX-II CPU. The machine is designed for research, scientific and engineering users and costs about $170.000. (NAN86).

FPS-AP120, Floating point systems, Beaverton, Oregon, USA.
This company produces also a new version of attached pipeline processors FPS-164 and FPS-264 which is used in configuration named LCAP (Losely coupled array of processors). More than 1500 machines had been sold and were used mostly for signal processing. They are quite cost effective in comparison to Cray or Cyber computers. (HOC81, HWA85, WIL82).

IBM 3838
IBM 3838 is a multiple pipeline scientific processor specially designed to attach to IBM mainframes, like the System/370, for enhancing the vector-processing capability of the host machine. It is microprogrammed pipeline processor which can be supplied with custom-ordered instruction sets for specific vector applications.

RECENT VECTOR PROCESSORS

Cray-1 Cray Research Inc., Chippewa Falls, Wisconsin, USA.
This is the first successful vector computer. More than 40 computers have been sold and installed, first in 1976. It comprises 12 special-purpose pipelines for the different arithmetic operations. It is very expensive. (HWA85, JOR82, RUS78).
An upgrade of this computer is Cray-2, (HOL85).

Cyber-205
This computer is an example of pipelined architecture and is highly competitive with the CRAY-1. It is based on CDC STAR 100. It is based on one, two or four pipelined general-purpose units working always to and from main memory. It is an expensive machine, designed initially to weapons' calculations and weather simulation. (HOC81, HWA85, VON84).

CDC/NASF Control Data Corporation Numerical Aerodynamic Simulation Facility.
This is a supercomputer to be used in 1990s for aerospace vehicle or superjet designs. The speed requirements was set to be at lest 1000 Mflops and the purpose is to calculate the viscous Navier-Stokes fluid equations for three dimensional modeling of the wind tunnel experiments. (HWA85, HOC81).

VP-200, Fujitsu.
This system has a scalar and a vector processor which can operate concurrently and it can be used as a loosely coupled back-end system. (HWA85, LLU84, UCH85).

OTHER VECTOR PROCESSORS

Ahmdal 1200
This computer is a European version of Fujitsu's recent vector processor VP-200. Similar version of VP-100 is known in Europe as Ahmdal 1100 computer. (KOC85).

Siemens VP200
This is another European version of Fujitsu's vector processor VP-200. Fujitsu's VP-100 is as Siemens product known as Siemens VP100. (KOC85).

YH1
This is Chine's first supercomputer, known also as "Galaxy". The development started in 1978 at the University of Defense Science and Technology in Changsa. The machine looks like a Cray computer. (NEW85/1).

SIMD ARRAY COMPUTERS

BSP, (Burroughs Scientific Processor), Burroughs (HOC81, HWA85, KUC82).
BSP has been largely based on the experiences that Burroughs have gained as major contractors on the ILLIAC IV project. The design principles of the BSP were to provide a machine using a standard technology, which would be programmed in a high level language and sustain a continuous 20-40 Mflops/s.

ICL DAP (ICL Distributed Array Processor) (HOC81).
This is an array of one-bit processors which are often called associative processors. The design of pilot DAP was started in 1974 and consisted of a two-dimensional arrays of 1024 1-bit processors.

ILLIAC-IV (BAR68, DAV69, BOU72, HWA85)
This computer was designed for the solution of partial differential equations and can be described as an 8x8 array of 64-bit floating point processing elements each (PE) with 2Kwords of memory. It was working with nearest-neighbor connections (fig. 1) and controlled by a single instruction stream processed in a central control unit.

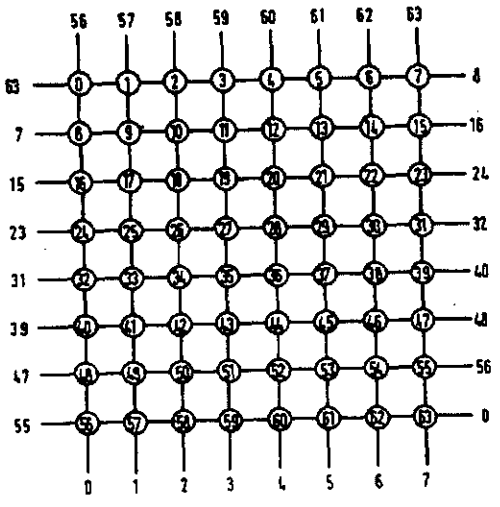


Fig. 1.: The connectivity between 64 processing elements in ILLIAC IV (HWA85).

MPP, (Massively Parallel Processor)
This processor was developed for processing satellite imagery at the NASA Goddard Space Flight Center and has 128x128=16384 microprocessors that can be used in parallel. Each processor is associated with a 1024-bit RAM. (BAT80, BAT82, HWA85)

PEPE (Parallel Element Processor Ensemble).
This special purpose computer is Burroughs floating point processor array which was developed at Bell Laboratories and designed to control a ballistic missile defense system of radar detectors and missile launchers. This is a loosely coupled system of 288 processing elements, each containing three processors. (KAR82, YAW77, FIN77)

STARAN
In this processor a bit serial associative memory is used. Staran consists of up to 32 associative array modules each containing 256 processing elements. The first Staran was installed for digital image processing in 1975. (YAW77, RUD72, BAT77, KAR82)

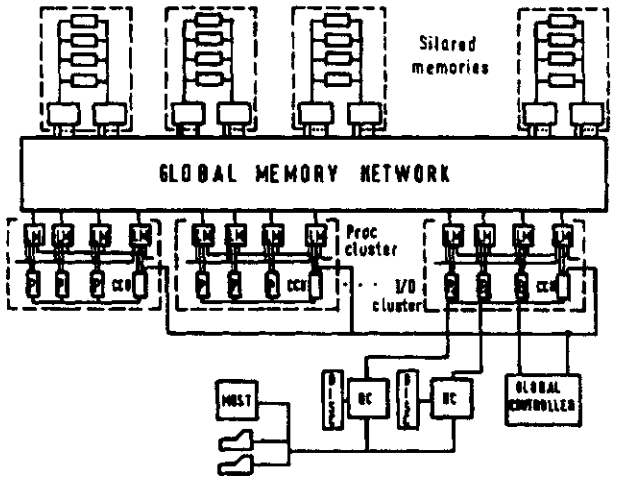
MIMD PARALLEL PROCESSORS

MASSIVELY PARALLEL PROCESSORS

The main accent in this architecture is in interconnection mechanism to connect several hundred processors with memory modules. A high processing power is achieved even by applying a standard processors.

BUTTERFLY, BBN - Bolt, Beranek & Newman (HOL85/2, HOL86, RET86)
The Butterfly computer is a large scale shared memory parallel processor that achieves high performance in configurations as large as 256 processors. The processors used are Motorola 68020 with Motorola 68881 floating point hardware. The system has a maximum performance of 256 MIPs of processing power in 1 MIP increment and up to 1 Gbyte of memory in 4 Mbyte increments.
Processor-memory interconnection is realized via a multistage self-routing switch network. All processors can access memory simultaneously and in parallel, provided that no two processors try to take the same output path from a particular node. Butterfly network for 16 processors and 16 memories, called barrel-switching network is shown in fig 17.
The speedup is nearly linear and is measured in a system with 256 processors ranging form 180 to 230 times that of a single processor (fig 22).

CEDAR, David Kuck, Duncan Lawrie and Daniel Gajski, University of Illinois at Urbana-Champaign, USA. (ABU84, ABU86).
Cedar is an eight year project that started in 1983. The architecture is hierarchical: sixteen clusters of eight processing elements are connected via an extended Omega global switching network to 256 global memory modules of 4 to 16 Mwords each. Each cluster has eight processing elements, each with 16 kwords of local memory. These processing elements are pipelined and interconnected via a local switching network (fig 2.)



LM - local memory
CCU - cluster control unit
P - processor
P1 - communication processor
DC - disc controller

Fig. 2.: The architecture of CEDAR parallel computer (ABU85).

The prototype Cedar 32 has four clusters of eight PEs and uses 400 ns clock period. This

gives a total maximum performance of 80 Mflops/s (Comparable to the Cray-1) for the desk-top sized prototype. Cedar 128 will have 16 clusters, giving total maximum performance of 320 Mflops (1988) and Cedar 512 will have 64 clusters, giving total maximum performance of 1.2 Gflop/s (1990). An alternative engineering is planned using 40 ns clock period giving a four cluster Cedar 32H 800 Mflops/s (1989) and 16 cluster Cedar 128H 3.2 Gflops/s (1991).
Extensive software development project called Parafrase is underway. It is focused on program transformations to enable standard FORTRAN programs to run on parallel Cedar machine.

TRAC, (Texas Reconfigurable Array Computer), J.C.Browne et. al., University of Texas, Austin.
16 8-bit microprocessors will be connected via a 4-level banyan switch to 81 memory modules. (JEN81, JEN82, LIP77, PRE82, SEJ80).

CHiP (Configurable Highly Parallel Computer), Lawrence Snyder, Purdue University, Indiana (SNY81/1, SNY81/2, YAL85)
This computer is an array of processing elements embedded in an array fo switching elements such that network connectivity between the processing elements can be reconfigured under program control in one machine cycle. The switch lattice is typically a regular structure such as four neighbor or eight neighbor mesh. Fig. 21. illustrates how the original lattice is reconfigured as a mesh and as a binary tree. The project aims to produce 2exp8 and 2exp16 processing elements with a few processing elements on a VLSI chip.

COSMIC CUBE (Nearest Neighbor Concurrent Processor, NNCP), Geoffrey Fox and Charles Seitz, CalTech (California Institute of Technology), Los Angeles, California.
The first machine is 2exp6 Hypercube hosted by VAX11/780, with processor Intel 8086 at each node together with 8087 floating point coprocessor and 128 Kbytes RAM. This machine was commercialized by Intel and marked as iPSC. The Intel iPSC is available with 32, 64 and 128 nodes. Each node is an Intel 80286 processor and 80287 coprocessor together with 512 Kbytes of local memory. The maximal performance of the 2exp10 hypercube is estimated to be about 100 Mflops/s that is to say about the same as the large supercomputers Cray X-MP and Cyber 205. (CHA86, SEI85, EMM85, EMM86/1, EMM86/2).

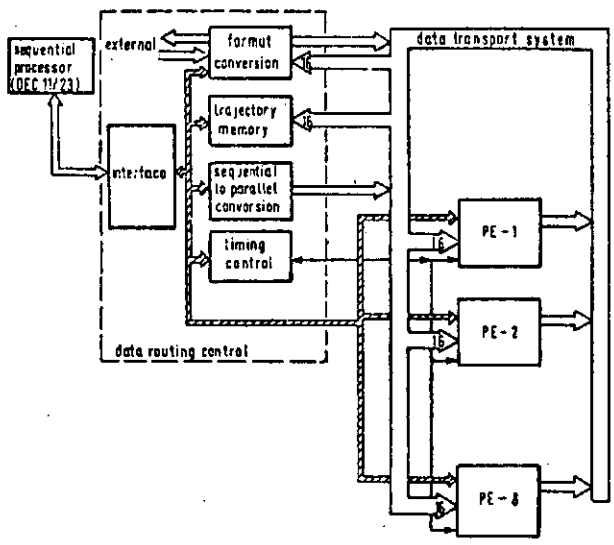
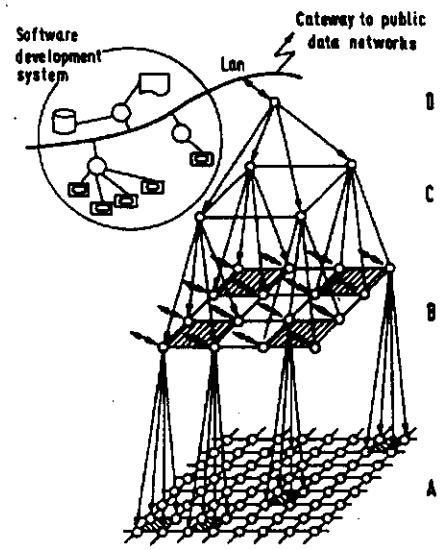


Fig. 3.: Architecture of Delft Parallel Processor DPP81 (SIP84).

DPP87 (Delft Parallel Processor 87)
This computer is an upgrade of DPP81. The DPP81 consists of one PE-cluster with 8 processing elements (fig. 3).
DPP87 is a modular MIMD system with up to 16 processing modules each having 32 processing elements. Each processing element consists of a stack oriented arithmetic processor AMD 9511. A PDP 11/23 is a host computer. The DPP87 computer is designed for simulation of systems (SIP84).

EGPA (Erlangen General Purpose Array). W. Handler (HAN85)
EGPA consists of a grid-like array of memory-coupled processor modules. Above the array there is a pyramidal hierarchy of processors for supervising and for data transport. Each node consists of one processor and one memory block (fig.4).



Processor-Memory-Module (PMM)

symmetric multiport-memory connections between neighboring PMMs

asymmetric multiport-memory connections between PMMs of different hierarchical levels

I/O communication to elementary pyramid, supported by I/O processor

Fig. 4.: The EGPA multiprocessor architecture consisting of 85 processor-memory-modules (HAN85).

The project started in 1975. The processor-memory modules are commercially available computers AEG 80/60. Interprocessor communication takes place via common control blocks and mailbox techniques.

FEM (Finite Element Machine), David Loendorf and Harry Jordan, NASA Langley Research Center, Hampton, Virginia.
Processing elements are TI9900 microcomputers, controlled by a TI990 minicomputer.

FLEX/32 (Flexible Computer)
This parallel system is composed of 32 bit processors NS 32032. Up to 20 processor modules are connected by a bus and for a box. Common busses link as many as 10 local buses per cabinet (fig 5.).

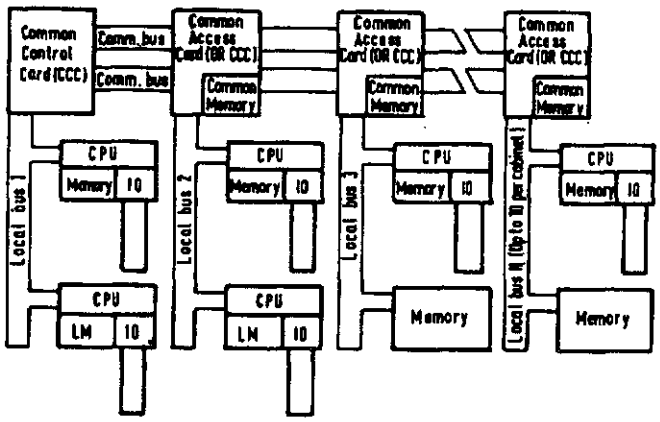


Fig. 5.: The architecture of Flex/32 multicomputer.

A total system is designed for 2480 processors. The system has global and local memory. The price for minimal configuration is $150.000. (MAN85, ZS086).

IBM GF11, IBM T.J. Watson Research Center, Yorktown Heights, NY, USA (BEE85)
GF11 is a parallel computer with 576 floating point processors (512 primary processors and 64 spares). Each processor has space for 2 Mbyte of memory and is capable of 20 Mflops, giving the total machine a peak of 1.1 Gbyte of memory and 11.5 Gflops. The floating point processors are interconnected by a dynamically reconfigurable non-blocking switching network called the Memphis switch (fig 6.).

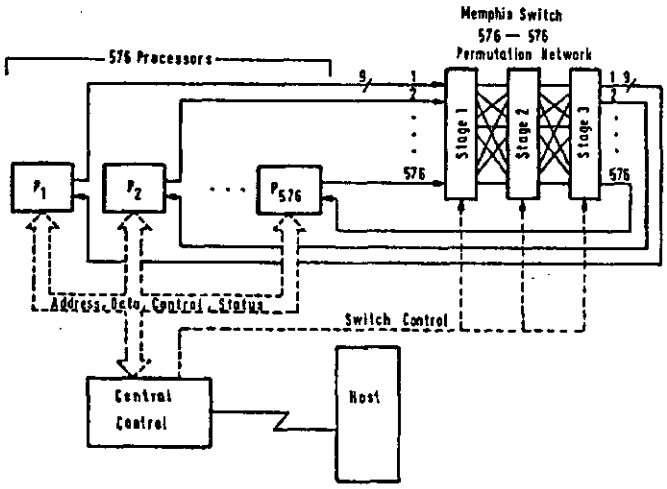
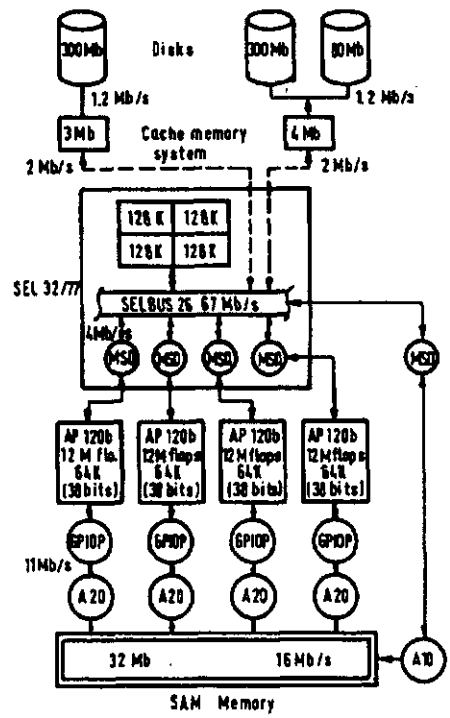


Fig. 6.: The GF11 architecture.

The main intended application of GF11 is a class of calculations arising from quantum chromodynamics in nuclear physics where GF11 is expected to be 100 times faster than Cray 1.

IBM RP3 (Research Parallel Processor Project), IBM T. J. Watson Research Center, Yorktown Heights, N.Y. (PFI86).
RP3 project is performed in cooperation with the Courant Institute of Mathematical Science at New York University. The goal of RP3 project is a parallel system with 512 32-bit microprocessors and 2 Gbytes of main storage. RP3 has two multistage Omega-like networks to connect processor-memory elements. The first network is constructed from high-speed bipolar logic and designed for fast interconnection to nonlocal storage. The second network is using a technique, developed at the NYU Courant Institute Ultracomputer project (EDL85, GOT82, GOT83, SCH80). This network contains more complex functions required to carry out synchronization operation and storage requests. With these techniques the efficiency of parallel system is not degraded as the number of processors is increased.
The language support, initially envisaged, consists of minimal extensions of commonly used languages such as Fortran, C, and Pascal. The adaptation of other languages, such as Common Lisp and Ada to the highly parallel environment is also being studied.
The operating system will probably be an extension of BSD 4.2 Unix, modified internally to make it a fully distributed, symmetrical system, and extended to provide multiple process shared memory and efficient massage passing.
The RP3 is expected to achieve an aggregate of 1000 MIPS on shared memory scientific applications. A demonstration of the RP3 with 64 processor-memory elements is planned for 1987.
RP3 machine is not associated with any IBM product program, and it is expected that very few of the RP3 prototypes will be constructed. RP3 is target to exploit automatic parallelization of Fortran and compilation of functionally-oriented languages.
Development of parallel applications for RP3 is performed before the completion of RP3 hardware construction by means of an experimental emulation system called EPEX (Environment for Parallel EXecution).

MIDAS (Modular Interactive Data Analysis System), Creve Maples, University of California, Berkeley.
This is a hierarchical system of a primary



AP 120B - array processor
HSD - high speed data interface with 4 Mbits/s throughput
GPIOP - general purpose I/O processor

Fig. 7.: The architecture of ONERA parallel computer (LEC86).

computer controlling several secondary computers, each of which controls a Multiple Processor Array (MPA). Each MPA also has an input and an output processor and a crossbar switch connecting the processors to 16 switchable memory modules each of 256 Kbytes. The system is used to solve problems in computational physics particularly nuclear science.

ONERA. (Office National d'Etudes et Rechers Aerospatiales).
This is French project on a multi-array processor which started in 1979 and evolved to a loosely coupled architecture. It was designed to solve partial differential equations. The present system has as a host a 32-bit Gould SEL 32/77 minicomputer. Four array processors AP 120B are connected to the SEL bus. The APs are connected to a sharable memory of 32Mbyte (fig. 7.) (LEC86, ADE85)

PASM (Partitioned SIMD/MIMD machine), Howard J. Siegel, Purdue University Indiana.
Envisaged that a full machine might have 1024 processing elements connected via a multilevel switch network. A prototype will have 16 Motorola 68000 processors and four control units (SIE81, SCH86).

PRINGLE, University of Washington and University of Purdue (called RP2).
64 processors Intel 8031 are connected via a switch and controlled by an Intel 8086. (KAP84).

SUPERNUM. W. K. Giloi, H. Muhlenbein, (GIL86, HOP86, KRA87)
This is a national project in parallel processing in Germany. The computer has a hierarchical hardware structure consisting of nodes, clusters and hyperclusters. It is conceived as a 64 cluster machine with 1024 nodes. Each cluster has 16 nodes and 140 Mbytes winchester disc. Motorola 68020 processors are used in nodes. Fig. 8 shows a SUPERNUM architecture with 16 clusters, where each cluster has 4 processors. The processors in the cluster are connected by s bus. The clusters are connected by row and column rings.

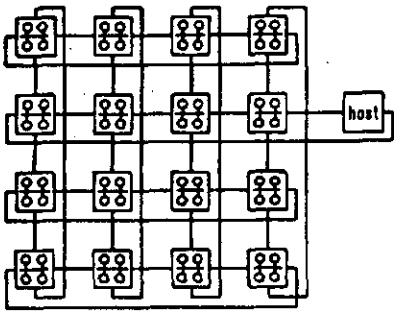


Fig. 8.: The topology of the SUPRNUM parallel computer (KRA87).

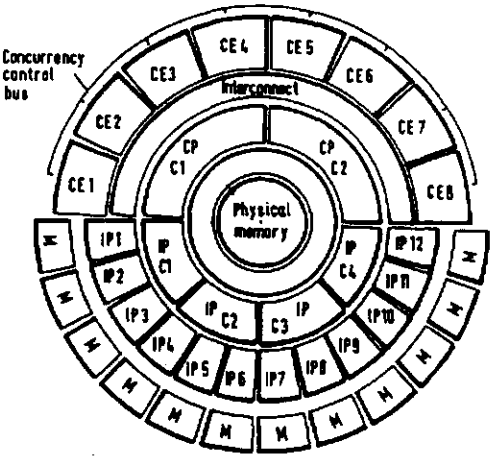
The system is designed for solving partial differential equations and other numerical applications.

VFPP (Very Fast Parallel Processor), Norman Christ and Anthony Terrano, Columbia University, New York.
This is conceived as a 16x16 array of 256 processing elements with nearest neighbor connections. Each processing element has an Intel 80286 with 80287 coprocessor, 20 Kbytes of local memory and an 11 stage pipelined microprogrammable vector processor. VFPP is a special purpose computer designed for lattice-gauge and similar calculations.

SMALL SCALE PARALLEL SYSTEMS

The word small in this context means only a few processors connected together. To get high processing power by these machines one needs powerful processors with interconnection mechanism which need not be as sophisticated as by connecting several hundred processors.

ALLIANT FX/8 (Alliant Computer Systems Corp., Acton, MA 01720) (LLU86/2, HAR86, SIE86, TEC85)
An FX/8 computer combines vector and concurrent processing in a system consisting from 1 to 8 computational elements. Each computational element is a microprogramed processor which can execute both scalar and vector instructions. Computational element can access via a crossbar switch, two 64 Kbyte caches. FX/8 can have from 2 to 12 interactive processors based on a Motorola 68012 microprocessor. Each interactive processor has 512 Kbyte local memory and is designed for execution of parallel I/O and operating system tasks (fig. 9).



CE - computational element
IP - interactive processor
IP C - IP cache
CP C - CP cache
M - multibus

Fig. 9.: The Alliant FX/8 architecture.

FX/8 operating system Concentrix is an extension of UNIX Berkeley 4.2 version. A minimal configuration with 1 computational element costs $270.000.

CONVEX C-1 (Convex Computer Corp., Richardson, TX 75081)
C-1 supercomputer is based on a Cray-like architecture. The processing units are interconnected through 64-bit buses and include a dedicated scalar and vector unit. It has a dual ported main memory and up to five 32-bit I/O processors. The operating system is Convex UNIX operating system, similar to UNIX 4.2 BSD operating system. The price for a basic system is $500.000. New versions are coming to the market, C1 XP processor, a faster version of C-1. (HOL85/3, TEC86)

CRAY X-MP, Steve Chen, Cray Research Inc., Mendota Heights, Minnesota. (HWA85, OED86, ERH86, LUB85)
The Cray X-MP is a multiprocessor upgrade of the Cray-1 architecture. It comprises 1, 2 or 4 CPUs sharing common memory of up to 8 Mwords in 64 banks of 38 ns ECL memory chips (fig 10).

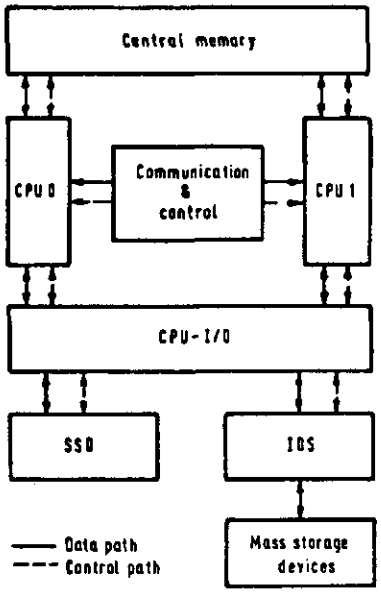


Fig. 10.: Cray X-MP system organization (HWA85).

Each CPU has 13 pipelined functional units which operate with data in 8 vector registers, each holding up to 64-bit elements. Three memory access pipelines or ports are provided, allowing each CPU to read two vector arguments and store one vector result simultaneously. Each CPU executes its own instruction stream, and rapid synchronization is achieved via 16 shared registers and 32 shared one-bit flags. The clock period is 9.5 ns, giving maximum performance of a pipeline 105 Mflops/s.

CRAY-3, Seymour Cray, Chippewa Falls, Wisconsin. (ERH86, HWA85, OED86,)
The Cray-3 is scheduled for 1987. It consists of 4 CPUs accessing a shared memory of 256 Mwords. It is an implementation of Cray-2 in gallium arsenide technology, and it is speculated that a clock period of 1ns might be obtained leading to a maximal performance of 1Gflops/s per floating point pipeline.

C.mmp, Carnagie-Mellon University, Pittsburgh, Pennsylvania, (HWA85, JON80, MAS82, OSL82).
This was one of the most ambitious early examples of MIMD computers. This comprised 16 DEC PDP-11 minicomputers connected to 16 memory modules by a 16x16 crossbar switch. The design stagted in 1971 and the machine was completed in 1975.

Cm*, Carnagie-Mellon University, Pittsburgh, Pennsylvania.
This computer was a successor of C.mmp and was based on microprocessors that had now been available. Communication between the microprocessors is via a hierarchical packed switching network. A basic computer module was DEC LSI-11 microprocessor and may act as an independent computer or may be linked to a common interclusted bus with up to 14 other modules to form a tightly coupled cluster. The total Cm* is built-up by loosely coupled clusters. (SWA77).

CYBERPLUS, CDC Corporation, Minneapolis, Minnesota, USA.
The architecture of CyberPlus is based on communication via multiple ring topology. The architecture was derived from the Advanced Flexible Processor which was build for rapid analysis of photographs taken form aircraft. CyberPlus comprises form 1 to 16 CyberPlus processors connected in a ring and attached to a channel of a host CDC Cyber 170/800. Up to 4 such rings can be attached to the host. Communication between processors is achieved by sending information packets to the ring. The packets move round the ring at the rate of one station per clock period, until their destination is reached.
The CyberPlus processor has 256K or 512K words of 64-bit memory for floating point data, 16K words of 16-bit memory for integers and a program memory of 4K words of 240-bit instructions. It has 15 independent functional units. The clock period is 20 ns , giving a floating point capability of 65Mflops/s in 64-mode and 103 Mflops/s in 32-bit mode.

ELXSI 6400, Elxsi Corp., San Jose, California.
This computer is similar to CyberPlus. It contains 1 to 12 CPUs and 1 to 4 I/O processors accessing 1 to 6 memory systems via the Gigabus. Potentially the system can achieve 72 Mips. The system incorporates three operating systems: Embos, UNIX BSD 4.3 and UNIX System V.2 - they can all run concurrently. The price for 12-CPU system is approxametely $3 million.

HEP (Heterogeneous Element Processor), Burton J. Smith, Denelcor Inc. Aurora, Colorado ,USA. (HWA85, LIN85, SNE85)
The HEP computer was the first commercial computer to offer the facility of programming with multiple instruction streams. A full HEP configuration comprises of 16 Process Execution Modules (PEMs) connected to 128 Data Memory Modules via a multilevel packet switching network called shuttle network. Each may have up to 50 user instruction streams.
But the largest system built at the time of writing has 4 PEMs and 4 DMMs and is installed at the NASA Goddard Space Flight Center (fig. 11).

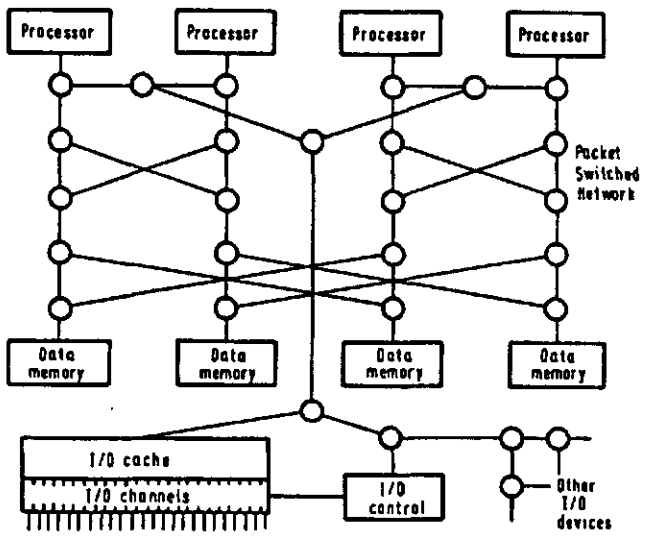
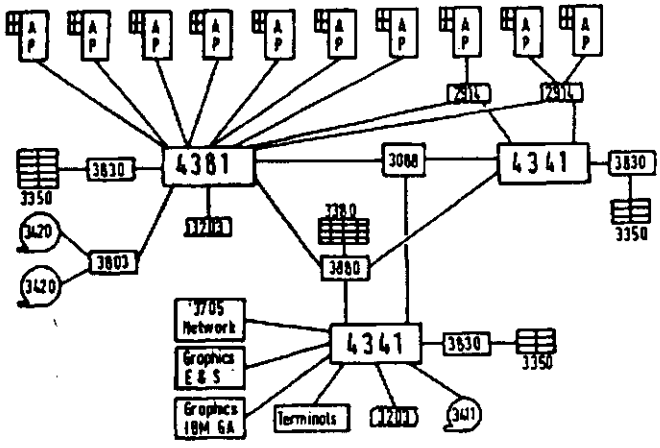


Fig. 11.: The architecture of a typical HEP system with four processors (HWA85).

IBM LCAP, (Loosely Coupled Array of processors). Enrico Clementi, IBM Kingston, USA. (CLE84, NEW86, SIE86)
LCAP is a powerful parallel system put together

from parts "off the shelf". These parts are from IBM and Floating Point Systems, Beaverton, Oregon. IBM contributed a host which may be different in different configurations as for example IBM 4381, IBM 4341, IBM 3081, IBM 3089, IBM 3090, and Floating Point Systems contributed attached pipeline processors FPS 164, FPS-164/MAX and FPS-264. Each FPS-* is a single instruction stream computer. An example of LCAP configuration consists of seven FPS-164, each with 4Mbyte of main memory, attached to an IBM 4381 host through a 3 Mbyte/s channel, tree FPS-164 are hosted by an IBM-4341 (fig. 12). For favorable problems the system is capable of 60 Mflops/s.



AP - array processor

Fig. 12.: Schematic diagram of the LCAP architecture (CLE84).

A bottleneck in this architecture is the time needed to transfer information between the constituent computers. It is necessary to decompose a problem into substantial parts that seldom need to communicate with each other. Many physical and chemical problems decompose well and for those problems an LCAP is a cost effective computer.
A similar project to Clementi's is that of Ken Wilson at Cornell University. He has linked 8 FPS-100 to VAX 11/750. He plans to expand the system to 4000 164/MAX to give a theoretical maximum performance of 40 Gflops/s.
New versions of LCAP are under construction LCAP-2 and LCAP-3.

MINERVA, Lawrence Widoes, Stanford University, California.
8 Intel 8080 microprocessors and four Intel 3000 microprocessors form a shared memory bus system.

PLURIBUS
This is a symmetric tightly coupled multiprocessor which was based on Lockheed SUE minicomputer, this is a 16 bit computer similar to DEC PDP-11. The system has its beginings in 1972. A lot of attention was paid to software development. (KAR82)

S-1, Michael Farmwald, George Michael et al., Lawrence Livermore Laboratory, Livermore, California. (HWA85)
This is the largest MIMD project, sponsored by US Navy and Department of Energy. The complete design for S-1 computer comprises 16 Cray-1 class pipelined vector computers connected to 16 memory banks by a full cross-bar switch. The S-1 can therefore be regarded as a "grown-up" version of C.mmp. An overall performance is expected to be 1 Gflop/s. Each of the uniprocessors is provided with a data cache of 64 Kbytes and instruction cache of 16 Kbytes in order to limit traffic through the switch. Each memory module may contain up to one Gbyte of storage, giving a total physical storage of up to 2 Gbyte. Single instructions are provided for some common mathematical functions e.g. sine, exponential, etc, and operations e.g. matrix multiply, fast Fourier transform, etc.

We have in this survey excluded logic machines that are being proposed to support the aims of Fifth Generation project, and also dataflow and reduction machines. The reason for this is the fact that these type of computers will probably form a special group of dedicated machines and will not evolve in a general purpose parallel computer of 90'.

3. MULTIPROCESSOR SYSTEM

After we have looked at different parallel computer system architectures, we are going to concentrate on multiprocessors. We are particularly interested in multiprocessors because they are almost general purpose parallel computers and have therefore a great potential power to upgrade or even replace some existing computer architectures.

Multiprocessor is a single computer with multiple processors. Processors communicate and cooperate at different levels in solving a given problem. Multiprocessor is classified as MIMD computer which is defined to be a control-flow computer capable of processing more than one stream of instructions. The communication between processors may occur by sending messages from one processor to the other or by sharing a common memory. Processors have access to common sets of memory modules and peripheral devices.

A mutiprocessor system is controlled by one operating system which provides interaction between processors and their programs at the process, data set and data element level.

Multiprocessors are classified according to organizational classification into tightly coupled and loosely coupled multiprocessors and according to structural classification into groups which have similar interconnection structures or topology.

3.1 Multiprocessor organization

Multiprocessors can be organized in a tightly coupled organization or in a loosely coupled organization.

Tightly coupled multiprocessors

Tightly coupled systems can tolerate a high degree of interaction between tasks performed on different processors. Processors communicate through a shared main memory. A small local cache memory may exist in each processor. The connectivity may be accomplished by different interconnection structures between the processors and the shared memory. When two or more processors attempt to access the same memory unit concurrently performance degradation occurs due to memory contention (fig. 13).

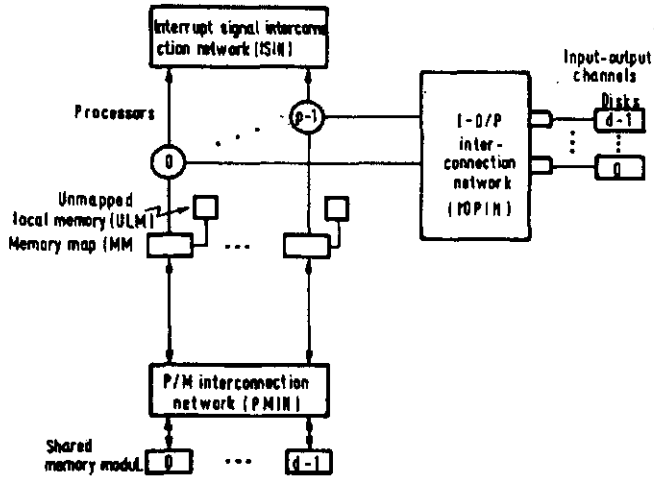


Fig. 13.: Tightly coupled multiprocessors where a complete connectivity exists between the processors and memory (HWA85).

Loosely coupled multiprocessors

Loosely coupled systems are efficient when the interactions between tasks are minimal. Processes which execute on different computer modules communicate by exchanging messages through a message - transfer system (Fig. 14). In loosely coupled multiprocessors each processor has s set of input-output devices and a large local memory where it accesses most of instructions and data. Sometimes loosely coupled multiprocessors are referred to as a distributed system.

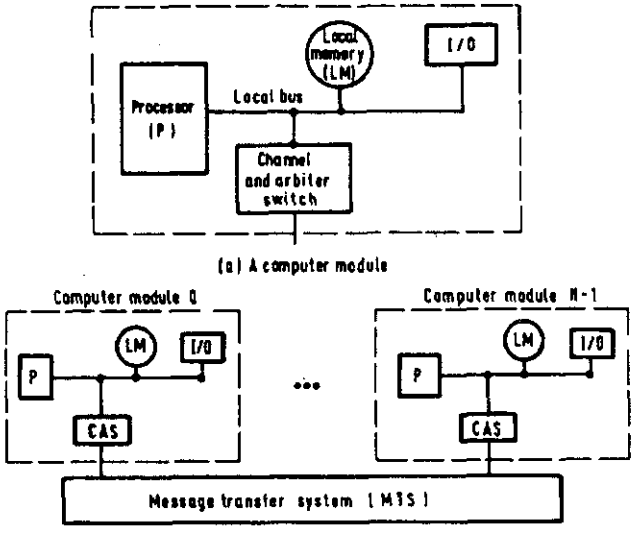


Fig. 14.: Loosely coupled multiprocessors where processes communicate through message transfer system (HWA85).

The message - transfer system could be a time shared bus or a shared memory system.

Very often computer architecture in parallel processing is a combination of loosely and tightly coupled processors. Loosely coupled systems have often hierarchical organization.

3.2. Interconnection structure

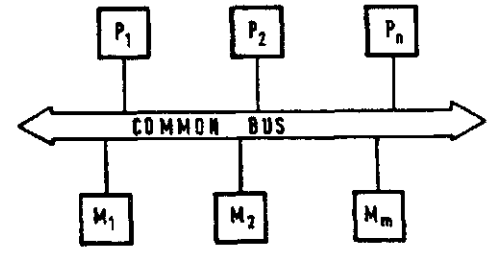
Interconnection structure between the memories and processors is either:

- time shared common bus,
- switched network,
  - * crossbar
  - * multistage
    - . omega
    - . banyan
- multiport memories,
- interconnection network,
  - * mesh,
  - * cube,
  - * reconfigurable,
  - * hierarchical.

A term "interconnection network" is sometimes used also for switched networks especially for multistage switched networks because they interconnect processors with memory modules. But we distinguish in this paper switched networks from all other interconnection networks.

Time shared common bus

The time shared common bus is the simplest and attaches every processor to every memory board (fig. 15). Bus requester, driver, and receiver perform all address and data handling. Because its low cost, low complexity and high maximum throughput, the common bus interconnection structure is today the most widely used commercial type of parallel computer system, but is limited to small shared memory computers with up to 20 processor modules (ALLIANT FX/8, ELXSI 6400, FLEX/32, MINERVA).



P - processor
M - memory module

Fig. 15.: A common bus interconnection structure.

Switched network

The switched network system is realized either as a crossbar switch or as a multistage switch.

A further subdivision of switched networks is divided, according to the type of interconnection network, into:

- cross bar (C.mmp, S-1)
- multi-stage (Butterfly, Cedar, GF-11, HEP, RP3, ULTRA, TRAC).

A switched system can be realized as shared memory system or as a distributed memory system. In a shared memory system each processor communicates with each memory module through a switch. In a distributed memory

system each memory module is connected as local memory to corresponding processor. The role of a switch is now to interconnect the processing elements, and there are no memory modules connected directly to the switch.

The **crossbar switch** is an extension of the common bus and implements N buses for N processors and M buses for M memory modules. A separate switch unit connects together a number of processors (P) and memory modules (M). The nodal circuits that couple the processor bus to a memory bus are the switch. A crossbar switch allows all processors to access memory modules simultaneously, as long as each processor accesses different memory module. When two or more processors contend for the same memory module, arbitration lets one processor proceed while the others wait by applying the same techniques as used on a common bus architecture. Since most of the logic is concentrated in the switch nodes, the complexity of a crossbar switch and its cost grows as the square of configuration size. The switch is quite likely to be the largest unit in the system and may be as expensive as one or several of the processors. It is a good choice for systems that are not highly parallel and have about 10 powerful processors.

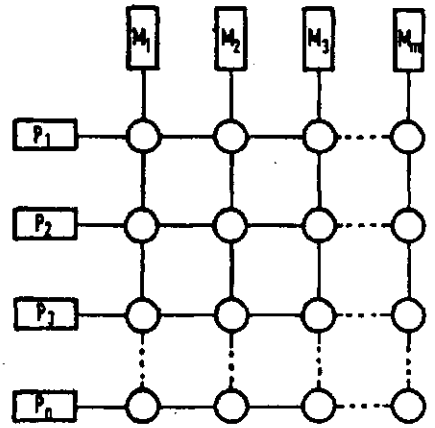


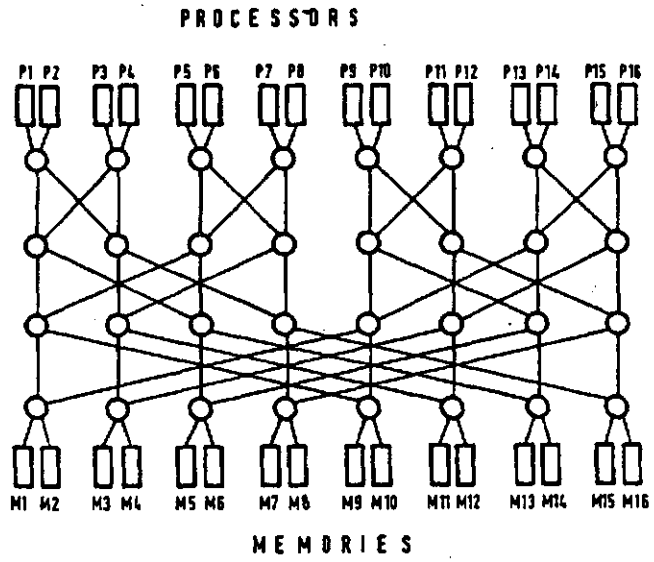
Fig. 16.: A crossbar switch interconnection structure.

A **multistage network** reduces the size and the cost of a crossbar switch by linking multiple crossbars as nodes in a network so that each node in a multistage network resembles a small crossbar switch. For example a multistage network that connects 16 processors to 16 memories, realized in two levels consists of 4-by-4 crossbars. The switching elements are distributed throughout the system (fig. 17).

A cost of multistage network that attaches n processors to n memories grows as n log n. All processors can access memory simultaneously , provided that no two processors try to take the same output path from a particular node. In order to reduce this limitation many multistage network architectures have extra pathways to reduce the potential of contention.

The switch in a multistage network is complex. A type of multi-stage switching network is usually omega or banyan. It seems that the multistage shared memory switched computer is the most favored current architecture in parallel processing and enables efficient parallel systems with up to a few hundred processors.

In most switched systems there is both substantial local memory as well as substantial global memory. Local memory is often realized in a form of registers, cache or buffer memory.



P - processor
M - memory module

Fig. 17.: A multistage network interconnection structure.

Multiport memory

A mutiport memory with m ports is similar to n-by-m crosbar switch. In a multiport memory system is the switching logic simply bounded onto the memory module.

Frequently memory module has two ports, one connected directly as local memory to one of the processors, and the other port connected to the switch. Thus each memory bank is both local to one of the processors and globally available to the other processors via the switch.

A multiport memory system could also be treated as a special technical implementation of a switched network and not as a topology.

Interconnection networks

This is a large varity of different interconnection network topologies in parallel systems. Networks are constructed in four different topologies:

- mesh networks (CYBER PLUS, VFPP)
- cube networks (COSMIC CUBE, iPSC)
- reconfigurable network (CHiP) and
- hierarchical network (Cm*, EGPA, SUPERNUM).

Mesh networks are one or multidimensional and are realized in square, hexagonal or other geometry. In a square mesh of dimensionality, d, each processing element is connected to 2d neighbors. The number of processing elements N=n exp d, may be varied independent of of the dimensionality by increasing the linear dimension of mesh n. For example, a square mesh with dimensionality 2 is connected to four neighbors. Fig. 18. shows a mesh system with 16 processing elements.

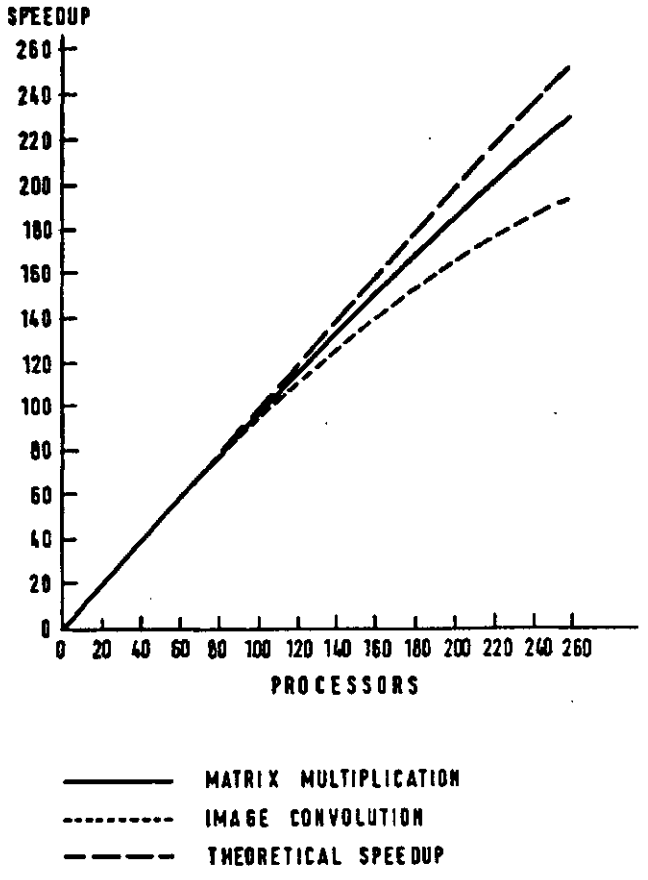


Fig. 22.: Linear speedup in a multistage switched network with 256 processors for Butterfly parallel system.

4. APPLICATION OF PARALLEL SYSTEMS

Uniprocessor architectures are approaching theoretical limits in processing speed.

In high speed or real time processing tightly coupled computer systems have to be used.

Most parallel computer nowadays are designed for numerical work with floating point numbers, and are build for the solution of large problems in physics, chemistry and engineering.

Large computer capabilities are necessary particulary in:

- complex graphic images,
- structural analysis,
- aerodynamics,
- meteorology,
- medical diagnostics,
- research in an oil exploration,
- research in fusion physics,
- industrial automatization,
- processing of sensing signals,
- genetic engineering,
- molecular dynamics,
- quantum mechanical problems,
- socioeconomic models, etc.

Mathematical problems which are solved by parallel systems are:

- Monte Carlo simulation,
- Hartree-Fock equation in the electron gas,
- finite element methods, etc.

Many of multiprocessors are almost general purpose as for example:

ALLIANT, BUTTERFLY, CEDAR, C.mmp, Cm*, CONVEX, COSMIC CUBE, CRAY X-MP, CRAY-3, CYBERPLUS, DCA, DPP, EGPA, ELXSI 6400, FLEX/32, FMP, HEP, IBM RP3, IBM LCAP, IBM GF11, MINERVA, ONERA, PLURIBUS, PRINGLE, SUPERNUM, S-1, TRAC, ULTRA.

These multiprocessors are designed for large scientific and engineering problems, for CAD automation, real time voice data multiplexing and other computationaly involved problems.

Some multiprocessors are more limited in applications and are considered as special purpose parallel computers designed for one bit logic operations, or image processing, knowledge based expert systems or designed for other special applications in artificial intelligence. Special purpose multiprocessors are:

CHIP, DADO, FEM, MANIP, MEIKO, PASM, PUMPS, VFPP.

5. WILL PARALLEL PROCESSING WIN?

Many ambitious projects in parallel processing in past have failed. For example ILLIAC IV cost four times the original contract figure and did not come even within a factor of 10 of its originally proposed performance. However its influence was profound and ILLIAC IV was the first to to pioneer the new and faster emitter-coupled logic (ECL) rather than the established transistor-transistor logic (TTL). ILLIAC IV also pioneered the use of 15-layer circuit boards and computer aided layout methods.

Other parallel computer systems of the 70' were also not very successfull. For example C.mmp and Cm* had problems with hot memories because their inteconnections structure, which was based on crossbar switch was not intelligent and could not overcome this problem. Nowadays these solutions are given. BSP NASF had a bottleneck in a central control processor and no efficient synchronization mechanisms were known at that time. But again ICL DAP was pioneering in an important feature of engineering design that processing element logic is mounted on the same printed circuit board as the memory to which it belongs. VLSI technology can now include processing element and its memory on the same chip.

Now the technology has advanced sufficiently to make parallel architecture practicable. Therefore we see such great interest in parallel processing.

6. CONCLUSION

It is not possible to predict which of these varied computer architectures will prove the most successful in future on the market.

By analyzing the performance of multiprocessors which is primarly dependent on interconnection structure one might get an insight to the development of parallel computer systems and try to predict future trends in parallel computing. It seems that in next decade the most influence on parallel computing are going to have the projects in massively parallel processor:

- NYU Ultracomputer, whose principles are applied in IBM-RP3 parallel system,
- Butterfly, produced by the company BBN Bolt, Beranek & Newman,
- Cedar, a multiprocessor supercomputer of the University of Illinois.

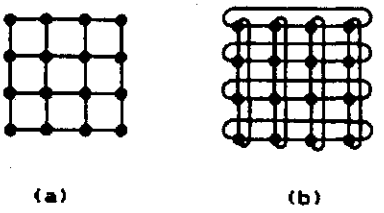


Fig. 18.: A mesh network with 16 processing element connected in a lattice (a) and connected as a torus (b).

Cube networks have either hypercube architecture or cube-connected-cycles network architecture. A cube-connected-cycles network is a cube where each node of the hypercube is replaced by a ring (or cycle) of processing elements. In a d-dimensional binary hypercube there are d connections to each processing element, n=2, and therefore the number of processors equals N=2 exp d. We see that the number of processing elements can not be increased without also increasing the number of connections to each processing element. For example, a six-dimensional hypercube which has 64 nodes is topologically the same as 4x4x4 three dimensional mesh with triply periodic boundary conditions.

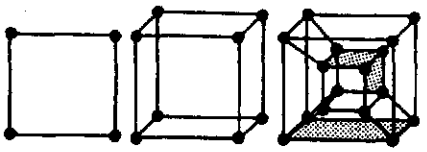


Fig. 19.: A cube network with 4, 8 and 16 processing elements.

Hierarchical class of multiprocessor systems is realized as tree network, hierarchy of pyramides or clusters of clusters.

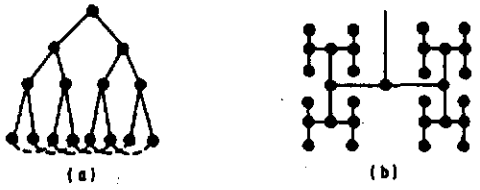
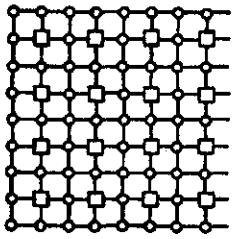


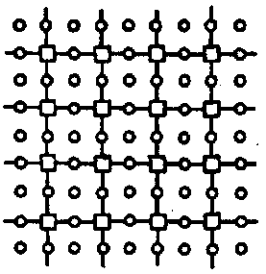
Fig. 20.: A hierarchical network realized as a tree network.

Reconfigurable networks include all cases in which the interconnection pattern between processing elements can be changed. This is usually achieved by interspersing switching elements between the processing elements which may be controlled by a user program.

the original
lattice architecture



the switch lattice
configured as a mesh



the switch lattice
configured as a
binary tree

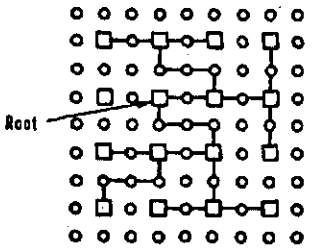


Fig. 21.: The original switch lattice in CHiP parallel computer configured as a mesh and as a binary tree.

Let us compare three most widely used topologies: common bus, crossbar and multistage network. Seven features are going to be compared:

1 - cost
2 - complexity
3 - max. throughput
4 - interconnect bandwidth
5 - # of signal paths
6 - efficiency
7 - max. # of CPU

We see (table 1.) that cost of a parallel system is the lowest in a common bus topology, but efficiency drops with increasing the number of processors. A crossbar switch is very powerful in connecting a few processors, but the price and complexity of a system is very high. A multistage network is a good topology to interconnect a large number of processors for a medium cost.

FEATURE	BUS	CROSSBAR	MULTISTAGE NET.
1	low	high (CPU exp2)	medium (n logn)
2	low	high (CPU exp2)	medium
3	high	no limit	high
4	fixed by cycle time	proportional to # of CPU	proportional to # of CPU
5	large	medium	medium
6	drops	linear	linear
7	up to 30	up to 10	up to 1000

Table 1.: Comparison of different features for a common bus, a crossbar and a multistage parallel computer system.

New synchronization mechanisms for multistage switched networks are nowadays enabling almost linear speedup for systems with up to 256 processors (fig. 22).

The reasons for success of these projects seem to be the fact that they are devoted to development of a GENERAL PURPOSE MIMD parallel computer. Excellent performance results are reached particularly because they use:

- MULTISTAGE INTERCONNECTION NETWORK: A near linear speedup is reached by 256 processors using as interconnection structure between the memories and processor a multistage interconnection network (Omega network);

- INTERLEAVING: that is spread data uniformaly throughout common memory modules in order to avoid contention for any one memory module;

- FETCH-AND-ADD: a very effective interprocessor synchronization operation.

An operating system seem to be a parallel version of a UNIX-like operating system. It is possible to achieve high performance by connection a large number of processing elements, even with "of the shelf" standard processors. It seems also that some architectural features as for example the size of local memory or the size of cache memory at every processor is of secondary importance for high performance of a parallel system.

7. REFERENCES

(ABU84) Abu-Sufah W., A. Kwok, Performance Prediction Tools for Cedar: a Multiprocessor Supercomputer, IEEE Conf. on Comp. Architecture, 1984, p.406-413

(ABU86) Abu-Sufah W., H. Husmann, D. Kuck, On I/O Speedup in Tightly Coupled Multiprocessor, IEEE Trans. on Computers, June 1986, p. 520-530

(ADE85) Adelatado M., D. Comte, P. Siron, Ph. Berger, Expression of Concurency and Parallelism in an MIMD environment, Computer Physics Commentars 37, 1985, p. 63-67, North Holland

(BAR68) Barnes G. at al., The Illiac IV Computer IEEE Trans. on Comp., August 68, p. 746-756

(BAT77) Batcher K., The multidimensional Access Memory in STARAN, IEEE Trans. on Comp., 1977, p. 174-177

(BAT80) Batcher K. E., Design of a Massively Parallel Processor, IEEE Trans on Comp., Sept. 80, p. 836-844

(BAT82) Batcher K. E., Bit Serial Parallel Processing System, IEEE Trans. on Comp., May 82, p. 377-384

(BEE85) Beetem John, et al., The GF11 Supercomputer, IEEE, pp 108-115, 1985.

(BOU72) Bouknight at al., The Illiac IV System Proc. IEEE, April 1972, p. 369-388

(CLE84) Clementi E. at al. Parallelism in Computations in Quantum and Statistical Mechanics, Proceeding on 2nd International Conf. on Vector and Parallel processors in Comp. Sci., Oxford, August 84, p.287-294

(CHA86) Chamberlain Richard, Experiances with the Intel iPSC hypercube, Supercomputer, p 24-29, 1986.

(DAV69) Davis R., The Illiac IV Processing Element, IEEE Trans on Comp., Sept. 69, p. 800-816

(EDL85) Edler J., A. Gottlieb at.all, Issues Related to MIMD Shared-memory Computers: the NYU Ultracomputer Approach, IEEE Conf. on Comp. Architecture, 1985, p. 126-135

(EMM85) Emmen ad, Intel's iPSC: a family of parallel computers based on microprocessors, SUPERCOMPUTER News, May 1985

(EMM86/1) Emmen Ad, Hypercube-toy or tool?, SUPERCOMPUTER News, July/September 1986

(EMM86/2) Emmen Ad, Vector extension for the iPSC, SUPERCOMPUTER News, July/September 1986

(ERH86) Erhel J., Parallel programming and applications on Cray X-MP, Supercomputer, Sept. 86, p. 53-60

(FIN77) Finnila, Charles A., H. Love, The Associative Linear Array Processor, IEEE Trans. on Comp., Feb. 77, p. 112-129

(GIL86) Giloi W. K., H. Muhlenbeim, Rationale and Concepts on the Supernum Supercomputer Architecture, MIPRO 86, Opatija, 1st Jugoslav Conf. on New Generation of Computers, p. 3.1-3.17

(GOT82) Gottlieb A. at al., The NYU Ultracomputer - Designing a MIMD Shared Memory Parallel Computer, IEEE Conf. on Comp. Architecture, 1982, p. 27-42

(GOT83) Gottlieb A. at al., The NYU ULTRA computer-Designing on MIMD shared Memory parallel Computer, IEEE Trans. on Comp., 1983

(HAN85) Handler W. at al., A tightly coupled and hierarchial Multiprocessor architecture, Computer Physics Comm 37, 1985, p. 87-93

(HAR86) Hars N., New Systems offer near-supercomputer performance, IEEE, March 86, p. 104-107

(HOC81) Hockney R.W. and C. R. Jesshope, Parallel Computers, Adam Hilger Ltd, Bristol, p. 126-143, 1981

(HOL85/1) Hollenberg Jaap The Cray-2 computer system SUPERCOMPUTER 8/9 , September 1985

(HOL85/2) Hollenberg J., The Butterfly Parallel Processor Computer System, Supercomputer, Sept. 85, p. 23-27

(HOL85/3) Hollenberg J., The C-1: A Minisuper Supercomputer, March 85, p. 7-8

(HOP86) Hoppe H. C., H. Muhlenbein, Parallel adaptive full-multigrad methods on message-based multiprocessor, Parallel Computing, Oct. 86, p. 269-289

(HWA85) Hwang K. and F. Briggs Computer Architecture and Parallel Processing, McGraw-Hill Book Company, p. 237-241, 1985.

(JEN81) Jenevein R., D. Degroot, G. Lipovski, A Hardware Support Mechanism for Scheduling Resources in Parallel Machine Environment, IEEE Conf. on Comp. Architecture, 1981, p. 57-65

(JEN82) Jenevein R.,J.Brown, A Control Processor for a Reconfigurable Array Computer, IEEE Conf. on Comp. Architecture, 1982, p.81-89

(JON80) Jones A., P Schwarz, Experience Using Multiprocessor Systems: A Status Report, ACM Computing Surveys, June 80, p.121-167

(JOR82) Jordan T. L. A Guide to Parallel Computation and Some Cray-1 Experiences Parallel

Computations AP, 1982

(KAP84) Kapauan A., J. Field, D. Gannon, L. Snyder, The PRINGLE Parallel Computer, IEEE Conf. on Comp. Architecture, 1984, p. 12-20

(KAR82) Kartashev S., S. Kartashev, Designing and programming modern Computers and Systems, vol. 1, chapter II, Prentice-Hall, p. 143-154, 1982.

(KOC85) Koch Wilhelm , First European installation of Siemens VP-200, SUPERCOMPUTER 7, May 1985.

(KOG81) Kogge Peter M. The Architecture of Pipelined Computers, McGraw-Hill Bool Company, p. 159-162, 1981.

(KRA87) Kramer O. and Muhlenbein H., Mapping Strategies in Message Based Multiprocessor Systems (to be published).

(KUC82) Kuck David J. and Richard A. Stokes The Burroughs Scientific Processor (BSP) IEEE TRANSACTIONS ON COMPUTERS vol. C-31, No. 5, May 1982

(LEC86) Leca P., The ONERA experimental MIMD system, Supercomputer, Sept. 86, p.91-96

(LIN82) Lincoln Neil R. Technology and Design Tradeoffs in the Creation of a Modern Supercomputer IEEE TRANSACTIONS ON COMPUTERS vol. C-31, No. 5, May 1982

(LIN85) Linebock R., Parallel Processing: Why a Shakeout News, Electronics, Oct. 85, p. 32-34

(LIP77) Lipovski J., On a Varistructured Array of Microprocessors, IEEE Trans. on Computers, Feb. 1977, p 125-138

(LLU84) Llurba Rossend ,VP-200: Fujitsu's Supercomputer, SUPERCOMPUTER 2, July 1984

(LLU86) Llurba R., The Alliant FX/8 entry level supercomputer, SUPERCOMPUTER, March 86, p. 7-11

(MAN85) Manuel Tom, Parallel Machine Expands Indefinitely, Electronics Week, May 85, p. 49-53

(MAS82) Mashburn Henry , The C.mmp/Hydra project: An Architectural Overview, Computer Structures: Reading and Examples, ed D. Sieworek, Bell, Newell, p. 350 - 370, McGraw Hill, 1982

(OED86) Oed W., O. Lang, Modeling, measurements and simulation of memory interference in the Cray X-MP, Parallel Computing, Oct 86, 343-359

(OSL82) Oslund B., P. Hibbard, R. Whiteside, A Case Study in the Application of the Tightly Coupled Multiprocessor to Scientific Computation, Parallel Computations, ed G. Rodrigue, p. 315-364, Academic Press, 1982

(PFI86) Pfister G. F., Parallel processor project to link 512 32-bit micros, IEEE Computer, Jan. 86, p 98-99

(PRE82) Premkumar U., J. Browne, Resource Allocation in Rectangular SW Banyans, IEEE Conf. on Comp. Architecture, 1982, p. 326-333

(PUR74) Pursell Charles J. The control data STAR-100- Performance measurements NCC 74

(RET86) Rettberg Randall and Robert Thomas, Contention is no obstacle to shared-memory multiprocessing, Comm. of the ACM, vol. 29, No. 12 p.1202-1212, December 1986.

(RUD72) Rudolph J., A production implementation of an associative array processor- STARAN, Fall Joint Computer Conference, 1972, p. 229-241

(RUS78) Russell Richard M. The CRAY-1 Computer System Comm. ACM, vol. 21, No. 1, January 1978

(SCH80) Schwartz T., Ultracomputer, ACM Trans. on Programming Languages and Systems, Oct. 1980, p. 484-521

(SCH86) Schwederski Thomas and Siegel Howard Jay, Adaptable Software for Supercomputers, IEEE, Computers, pp.40-48, February 1986.

(SEJ80) Sejnowski et al., Overview of the Texas Reconfigurable Array Computer, AFIPS National Computer Conference, 1980, p. 631-642

(SEI85) Seitz Charles L. , The Cosmic Cube, Communications of the ACM vol.28, No.1, January 1985

(SIE81) Siegel H., PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition, IEEE Trans on Computers, Dec. 1981, p. 934-947

(SIE86) Sieworek D., New Trends in Comp. Architecture, MIPRO Conference, Opatija 1986

(SIP84) Sips H., The DPP81- an exercise in parallel processing, Supercomputer , Nov. 84, p. 31-37

(SNE85) Snelling D., HEP Applications: real time flight Simulation, Computer Physics Comm. 37, 1985, p.261-271

(SNY81/1) Snyder L., Programming Processor Interconnection Structures, Technical Report CDS-TR-381, Perdue University, 1981

(SNY81/2) Snyder L., Introduction to the Configurable, Highly Parallel Computer, IEEE Computer, Jan. 1981, p. 47-56

(SWA77) Swan, Fuller, Sieworek, Cm* - A Modular Multi-microprocessor, AFIPS National Computer Conference, 1977, p637-644

(UCH85) Uchida Keiichiro and Mikio Itoh, High Speed Vector Processor in Japan, Computer Physics Communications 37 (1985) 7-13, NH Amsterdam

(VON84) Vons Peter Cyber 205 vector-features used by vectorizers SUPERCOMPUTER 3, September 1984

(WIL82) Wilson Kenneth G. Experiences with a Floating Point Systems Array ProcessorParallel Computation AP, 1982

(YAW77) Yaw S. S., H. S. Fung, Associative Processor Architecture- A Survey, ACM Computing Surveys, March 77, p. 3-27

(ZSO86) Zsohar Leslie et all., Bus Hierarchy Facilitates Parallel Processing in 32-bit Multicomputer, Computer Technology Review, summer 1986, pp 51-59.

(NEW85) News, China's first supercomputer, SUPERCOMPUTER 6, March 1985

(NEW86) News, "Supercomputer is just an advertisement word" - an interview with E. Clementi, SUPERCOMPUTER, Sept. 86, p. 24-33

(TEC86) Technology to watch, This Minisuper is aimed at parallel processing, Electronics, Oct. 86, p. 56-60

UDK 681.3.06

Branko Mihovilovič, Peter Kolbezen, Jurij Šilc
Jozef Stefan Institute, Ljubljana

Abstract - The implementation of a raytracing algorithm on an array of transputers is presented. Such system has some important properties. Processing speed is directly proportional to the number of transputers used. The system is remarkably robust namely, individual transputers can be removed from the system while the program is running, and the system will continue to function althought with reduced performance and some loss of data.

KEY WORDS - Transputer, occam, raytracing, performances.

1. INTRODUCTION

This paper describes the implementation of a computer graphics program on an array of transputers. This program was written to provide a demonstration of the performance obtainable by using large numbers of transputers. We used a technique known as raytracing wich can generate very realistic images but requires massive amounts of computer power. This is an ideal application for transputers as the calculation for each picture element on the screen are independent of one another an so can be done in parallel on separate transputers. The entire program is written in occam, however, the main part of the program could have been written in any suitable language. Only those parts of the program which deal exsplicitly with concurrency and the distribution of work are easier to write in occam.

2. BACKGROUND

2.1. Transputer

Concurrent systems can be constructed from a collection of microcomputers which operate concurrently and communicate through point to point serial communication links. The INMOS transputer /1-4/ is the ideal building block of high performance multiprocessor systems which provide maximum speed with minimum hardware.

A transputer contains memory, a processor and a number of standard point-to-point communication links which allow direct connection to other transputers. The processing capability may be general purpose or may be optimized to a specific purpose. The on-chip memory may be extended of chip by suitable interface. A transputer may also have special purpose interfaces for connection to specific types of hardware.

The first transputer available was the IMS T414, a 32 bit RISC microprocessor with a throughput of 10 mips. It has 2 Kbytes of fast SRAM (50 ns) and four serial links. The 32 bit multiplexed bus allows up to 4 Gbytes of external memory to be accessed. Compatible with the T414 is IMS T800 wich includes 4 Kbytes SRAM and floating point arithmetic.

2.2. Occam

Occam /5,6/ is a small and elegant language. It combines the best of contemporary thought about control structures and variable scoping with some radical new structures to handle concurrency. It is based on an model of computation that is different from conventional languages in that it includes the notion of communication, parallel execution, and synchronization in its very structure.

The basic unit of occam programming is a process that performs a sequence of actions and then terminates. There may be more than one proccess executed at any given moment. Occam programs are constructed from three primitive processes: assignment, input and output. The assignment

```
v := e
```

sets the variable v to the value of the expression e. The output

```
c ! e
```

outputs the value of expression e to the channel c. Similarly, the input

```
c ? v
```

sets the variable v to a value input from the channel c. Constructors are used to combine processes to form larger processes. The sequential constructor, SEQ, causes its components to executed one after another. The parallel constructor, PAR, causes its components to be

executed concurrently. Finally the alternative constructor, ALT, chooses one component process for execution which is earliest ready. It is cleare that IF and WHILE constructors are also provided in occam program.

2.3. Raytracing

Raytracing is a now well introduced technique for realistic image synthesis from three dimensional geometric scenes. The basic raytracing algorithm is described in /7/ and is briefly given here. Simplifying somewhat, for each picture element of the rastered image, a ray is trased from the viewpoint into the three dimensional scene to calculate the first intersection with an object. If the object is reflecting or refracting, an appropriate ray is determined by the law of reflection and refraction. These new rays are traced analogously. To calculate shadows, the ray-object intersection points are connected by line segments to the point light sources illuminating the scene. If there is an object intersecting the line segment, the intersection point lies in the shadow of this light source, and its intensity is not taken into account for intensity calculations.

3. LOGICAL ARCHITECTURE

The key problem with raytracing is the relatively high unavoidable basic amount of computation. In the past, two main strategies were followed to process sets of elementary primitives: image decomposition and scene decomposition. In a former case a subset of picture elements of the image are assigned to each of several processors. Every processor has access to the relevant primitives of the scene. In the later case a subset of primitives are assigned to each processor. A processor has access to the relevant rays of the scene.

The calculations performed for each picture element on the screen are completely independent so they can be performed in any ordered and on any number of processors. The example of distributing the work to a number of processors is given in Fig. 1.

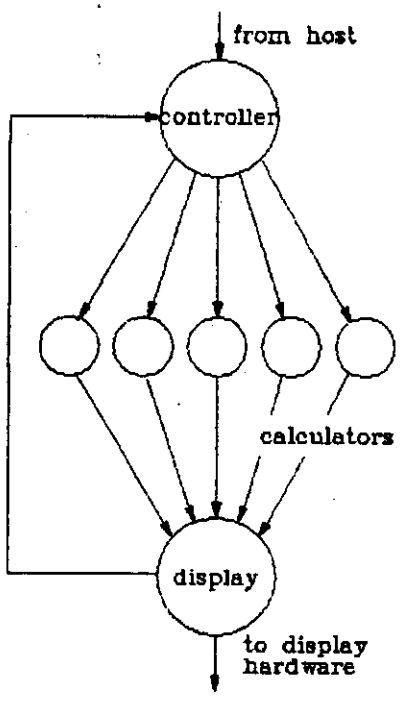


Fig.1: Logical architecture

This solution requires three different processes running concurrently on three, or more, processors:

- control process (controller),
- intersect and shading calculation process (calculator),
- display process (display).

A controller interfaces with the user or host computer to provide a description of the scene beeing viewed and allocate work to processors. A calculator can be replicated any number of times, to render the picture elements. A display collect the results from each calculator and drives the graphic display.

Every calculator is first given the description of the scene and then processing work can be allocated by the controller giving each calculator picture element to evaluate. When the calculations have been completed the results are passed out to the display. The display then informs the controller that there is now a free processor and another picture element is sent out for evaluation.

The amount of computation required varies from picture element to element and this method automatically balances the load amongst the processors and ensures they are all kept busy. An interesting idea here is that the picture elements do not need to be generated in sequence and, if they are generated in some pseudo-random order, a good impression oof the final picture can be obtained well before every picture element has been evaluated. This could be particularly useful in a CAD system where the user wishes to change his view of an object very rapidly.

Note, that this structure is not related to the raytracing algorithem and is suitable for any problem which can be broken into independent parts.

4. PHYSICAL ARCHITECTURE

4.1. Basic design

It appears, a first sight, that the above architecture cannot be maped directly into a network of transputers because of the fixed number of communication links available.

There is a partitioning which can aid the understanding and implementation of the structure of parallelism. The data processed on calculators consist a sequence of values (picture element), then all of the processes can be executed concurrently, even those which process the data in sequence. Alternatively the constructed process can be replicated over a number of calculating transputers each of which will execute the construct on a subset of the data structure as illustrated in Fig. 2. This hardware realisation named proces replication is mapped onto a network of transputers, active data structure is mapped onto the reconfigurable processor array (RPA). Both are ideal for occam process virtualisation. The occam model adopted in the RPA system, uses point to point communications to synchronise processes. A processprocessor mapping is implemented by providing a physical network of transputers, which is isomorfic to the process structure, but only at the chosen point in the hierarchy of the occam program.

It is very simple to arrange for the controller to communicate with any transputer in a network by passing messages through the intervening

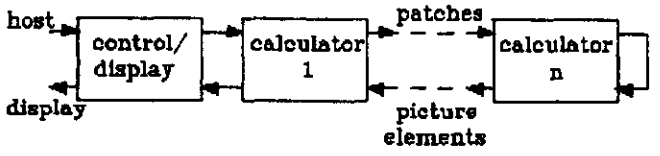


Fig. 2: Physical architecture

transputers. For simplicity, the raytracing algorithm was mapped on to a linear array of transputers /8/.

In Fig. 2 where the basic architecture is shown we see that each transputer link implements two occam channels, one in each direction. Therefore, this mapping uses only two of the four links available on a transputer. Control and display processes are executed in parallel on one transputer (control/display transputer) while the rest of the transputers do the intersection and shading calculation processes (calculating transputers). The control/display transputer also does these calculations and the same, parameterised, program is loaded into every calculating transputer. Such method of mapping processes requires that each calculating transputer also execute routing processes, i.e. commands and data are passed from the controller along the array and results are passed back for display. This linear connection of transputers implies some sort of command protocol for identifying the nature and destination of data, consequently the routing process on each transputer only needs to decide whether a message is to be accepted locally or passed on to be dealt with elsewhere.

4.2. The control/display transputer

There are two processes executed by the control/display transputer (Fig. 3):

- sendPatches,
- loadBalance.

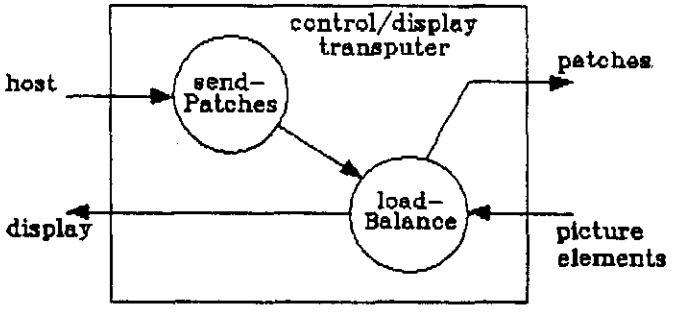


Fig. 3: Processes on control/display transputer

SendPatches interfaces to the host computer to receive the description of the scene being modelled and other commands. It passes the world model out to all the other transputers and then sends out requests for picture elements to be evaluated. Square areas of the screen, called patches, rather than individual picture element, are given to each transputer for two resons:

a) to give better ratio of calculation to communication;
b) to enable segments of data to be transmitted.

A segment communication transmits an array of words as a single operation. This has two big advantages over the transmission of individual words. Firstly, there is the same processor overhead for setting up the links to transmit a single word as for a million words. This better exploits the autonomous transputer links and allows the processor to continue calculating at very nearly full speed. Secondly, it gives a better ration of communication to computation.

LoadBalance coordinates the sending of data to the other transputers and the display of the generated picture elements. The first thing it does is to determine the number of transputers in the system. This is done by sending a count, incremented by each transputer, around the loop. If there are n transputers then loadBalance passes on primitives containing 2n picture elements from the process sendPatches and then waits until a result is returned before passing out another request.

4.3. The calculating transputer

The work of each of calculating transputers is organised as three processes (Fig. 4):

- througput,
- render,
- feedback.

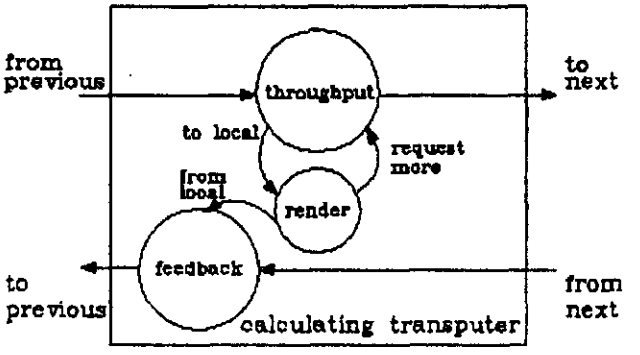


Fig. 4: Processes on the calculating transputer

The throughput process receives patch requests from the previous transputer and either pass them on to the next or keep them for processing locally. It is also able to hold one request in an internal buffer so, initially, it accepts two requestes. The first is passed immediately to render process for evaluation and the second is held until needed. Any further patches received are passed on to be evaluated elsewhere. As soon as render has finished the computation of its patch, throughput passes it the buffered patch to work on and is then ready to accept another. Since the time taken to compute a patch is less than the time before throughput receives the next patch request, the render process is kept busy.

The render process is given patches to evaluate, i.e. it does all the calculations to find intersections, build the bundle of rays and then traverse this bundle to get the final picture element value. When the picture ele-

ments in the patch are evaluated then another patch is requested from throughput and the picture elements are passed out to the feedback process. This is a completely sequential part of program which can be written in any standard programming language.

The feedback process multiplexes the local result and those received from other transputers and passes them back towards the display transputer via the shortest route.

4.4. Occam implementation

As we have said the transputer architecture simplyfies system design by the use of processes as standard software and hardware building blocks. The ability to specify a hardwired function as an occam process provides the architectural framework for transputers with specialized capabilities (e.g. graphics). The required function (e.g. graphics drawing and display engine) is definied as an occam process, and implemented in hardware with a standard occam channel interface.

The occam program of the proposed architecture (see Fig. 2) contain:

- the description of the entire transputer system,
- the control/display transputer program,
- the calculating transputer program.

For simplicity, only the essential outline of the mentioned above occam programs is given here.

The system description is as follows:

```
VAL number.transputers IS 24:
VAL last IS number.transputers - 1:

CHAN host, display, loopback,
     forward[number.transputers],
     return[number.transputers]:

PLACED PAR

-- transputer 0 = control/display transputer

 PROCESSOR 0 T4
       -- data from host
  PLACE host       AT link0in :
       -- to display
  PLACE display  AT link0out:
       -- patches out
  PLACE forward[0] AT link1out:
       -- picture element value back
  PLACE return[0]  AT link1in :

  control ( host, display,
            forward[0], return[0] )

-- the main body of the pipeline of calculators

 PLACED PAR i = 1 FOR number.transputers - 2
  PROCESSOR i T4
       -- patches in
   PLACE forward[i]  AT link0in :
       -- picture elements out
   PLACE return[i]   AT link1out:
       -- patches out
   PLACE forward[i+1] AT link1out:
       -- picture elements in
   PLACE return[i+1]  AT link0in :

   calculate ( forward[i], forward[i+1],
               return[i+1], return[i] )

-- the last transputer is a special case as it
-- has no one else to talk to. The fact that
-- the channel 'loopback' is not placed means
-- that an internal ("soft") channel will be
-- created. In fact this channel is never used
-- but is required as a parameter.

PROCESSOR last T4

 PLACE forward[last] AT link0in :
 PLACE return[last]  AT link0out:

 calculate ( forward[last], loopback,
             loopback, return[last] )
```

The program running on the control/display transputer is:

```
PROC control ( CHAN fromHost, toDisplay,
                    toCalculators, pixelsIn )

 ... definition of sendPatches procedure
 ... definitions of loadBalance procedure

 CHAN data:
 PAR
  sendPatches ( fromHost, data )
  loadBalance ( data, toCalculators,
                pixelIn, toDisplay )
```

Finally, each of the calculating transputers runs the following program:

```
PROC calculate ( CHAN fromPrev, toNext,
                      fromNext, toPrev )

 ... definition of the throughput procedure
 ... definition of the render procedure
 ... definition of the feedback procedure

 CHAN toLocal, fromLocal, requestMore:
 PRI PAR

  -- run these at high priority for
  -- fastes response to messages

  PAR
   throughput ( fromPrev, toNext, toPrev,
                toLocal, requestMore )
   feedback ( fromLocal, fromNext, toPrev )

  -- and this is at low priority

  render ( toLocal, fromLocal, requestMore )
```

5. CONCLUDING REMARKS

5.1. Performances

Whithout doubt, the processing speed of the system is directly related to the number of transputers used. A number of factors contribute to this aspect of the system. Most of these were carefully worked out design decisions but one had to be determined empirically. The transputers require only two words of data to specify the position of the all picture elements in the patch. If the work were distributed on a picture element by element basis then two words of data would be required for every element. This would mean a worst ratio of communication to processing. A more intelligent approuch use of segment communication for data (paragraph 4.2.). These means less processor overhead per word sent and allows a greater amount of concurrency between the link engines and the processor. It is important to say that the message routing processes are had to be run at high priority to ensure that incoming messages can be examined and forwarded immediately it is received. Carefully ordered priority in the ALT constructs of these processes are en-

sure that patches are returned to the control transputer as quickly as possible. Withholdings in the work in throughput and feedback processes, are reduced by introducing software buffers into two input channels of these processes. Channel buffers are frequently used, and easy to implement, in occam programs. Buffers introducing made a significant difference to the performance from

$$speed = (transputers + 1) * K/2$$

to very nearly

$$speed = transputers * K,$$

where K is the performance of a single transputer.

5.2. Robustness and reliability

The system described above is already remarkably robust. It should be possible to exploit the number of transputers with some degree of redundancy. If a calculating transputer fails then the system will progressively deadlock only if the neighbour, on the controller side, attempts to communicate with it. In order to make the system more robust it must be possible to detect when a failure has occured. This reqire the using a timeout on all communications. Secondly it must be possible to ensure that, if a communication does fail, all the input and output processes will terminate. These desired function are performed by a number of predefined procedures which allow an input or output to be attempted within a time limit, an recovery from a failed communication /9/. The use of these procedures means that failure of a transputer can be detected by its neighbour. The controlling transputer could then be informed and so take action to recover or regenerate the lost data. Detection of the failure of a transputer implies that facilities could be added to allow the defective transputer to be bypassed. As we remember, on each transputer two communication links are unused. So this can be done with no extra hardware in such a way that the precedent of fail transputer switches it to the other link to communicate with the next transputer along.

6. REFERENCES

/1/ Whitby-Strevens, C., The Transputer, Proc. 12th Annual Int'l Symp. Computer Arch., Boston, Massachusetts, 1985, pp. 292-300.

/2/ Taylor, R., Transputer Communication Link, Microprocessors and Microsystems, Vol.10, No.4, 1986, pp. 211-215.

/3/ Mihovilović, B., S. Mavrič, P. Kolbezen, Transputer - The Basic Componnent of Multiprocessor Systems, Informatica, Vol.10, No.4, 1986, pp. 81-84.

/4/ Mihovilović, B., P. Kolbezen, J. Silc, Communicating Processes in Transputer Systems, Informatica, Vol.11, No.2, 1987, pp. 74-77.

/5/ May, D., R. Taylor, Occam - An Overview, Microprocessors and Microsystems, Vol.8, No.2, 1984, pp. 73-79.

/6/ Curry, B. J., Occam Solves Classical Operating System Problems, Microprocessors and Microsystems, Vol.8, No.6, 1984, pp. 280-283.

/7/ Whitted, T., An Inproved Illumination Model for Shaded Displey, Comm. ACM, Vol.23, No.6, 1980, pp. 343-349.

/8/ Packer, J., Exploiting Concurrency; A Ray Tracing Example, Tech. Note 7, INMOS Ltd, Bristol.

/9/ Shepherd, R., Extraordinary Use of Transputer Links, Tech. Note 1, INMOS Ltd, Bristol.

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

UDK 681.3.068

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

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

1 INTRODUCTION

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

2 DESCRIPTION OF MODELS

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

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

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

2.1 Model 1

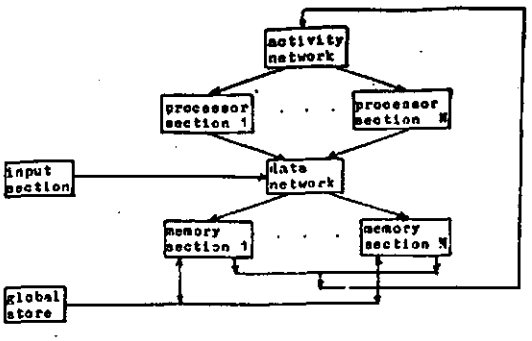


Figure 1:The model 1

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

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

2.2 Models Connected In Clusters

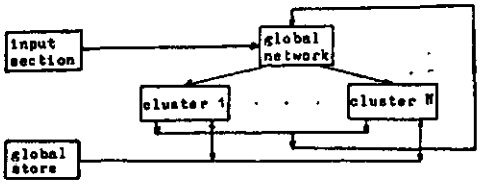


Figure 2:Models connected in clusters

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

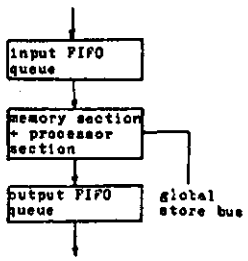


Figure 3

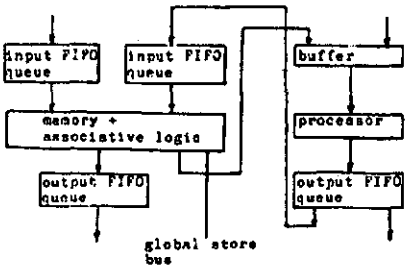


Figure 4

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

2.2.1 Model 2

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

2.2.2 Model 3

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

2.2.3 Models 4 And 5

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

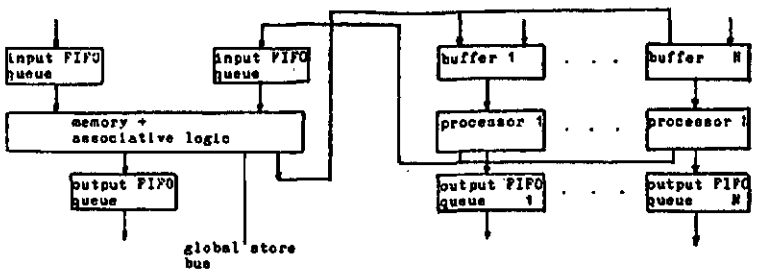


Figure 5:The cluster of model 4

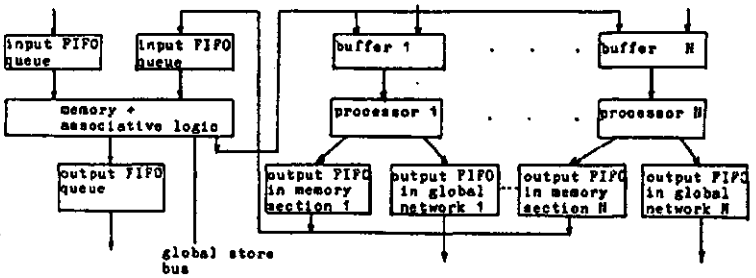


Figure 6:The cluster of model 5

3 SIMULATION RESULTS

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

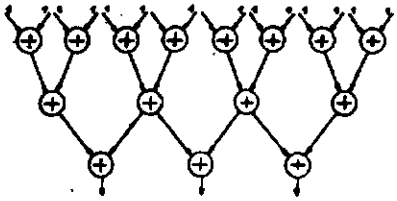


Figure 7

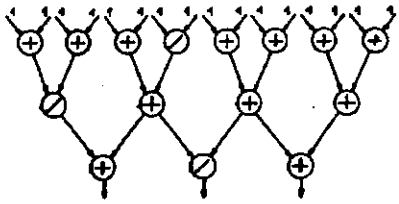


Figure 8

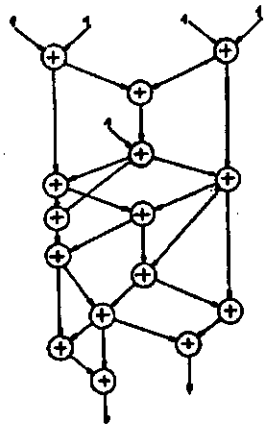


Figure 9

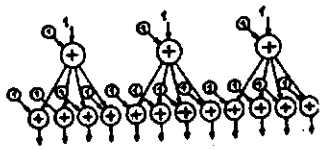


Figure 10

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

Our simulation is executed with these constant input parameters:

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

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

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

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

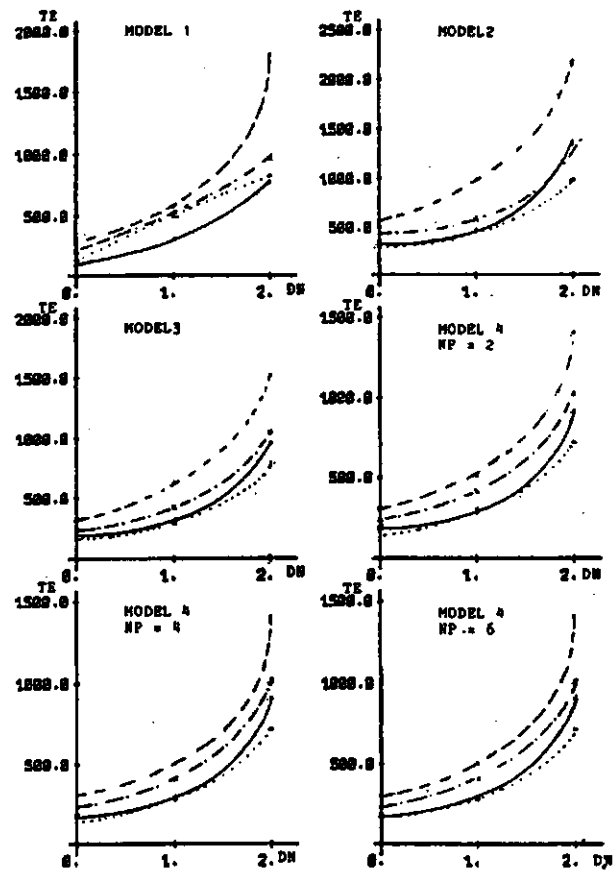
We have reduced the number of observed output parameters to the minimum following ones: a program execution time requested for execution of 200 instructions, an average number of busy processors, an utilization of processors, an average number of busy memory logic in memory sections, an average number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

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

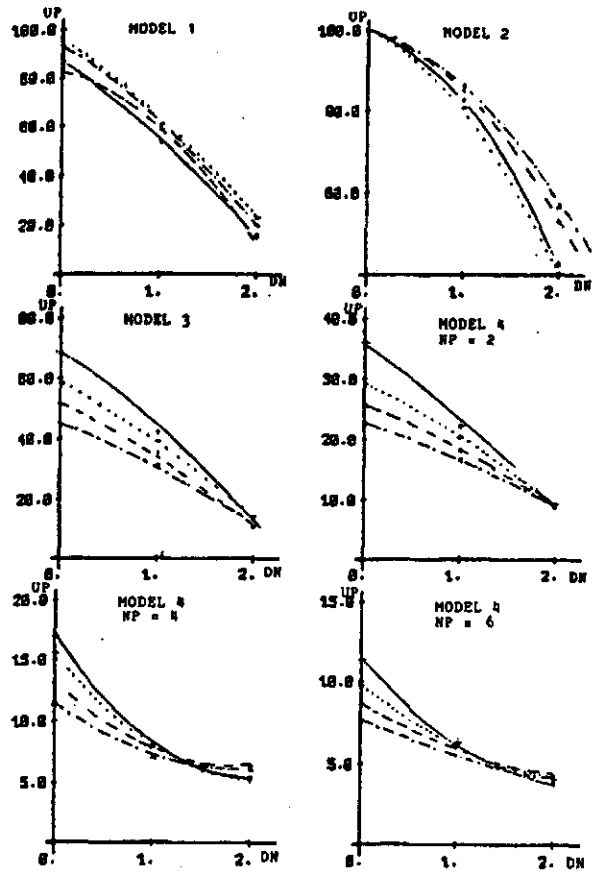
———	algorithm shown on Figure 7
- - - -	algorithm shown on Figure 8
—·—·—	algorithm shown on Figure 9
·····	algorithm shown on Figure 10

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

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



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



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

4 FAULT TOLERANT DATAFLOW COMPUTER

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

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

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

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

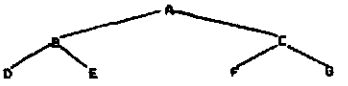


Figure 11: Task Invocation Structure

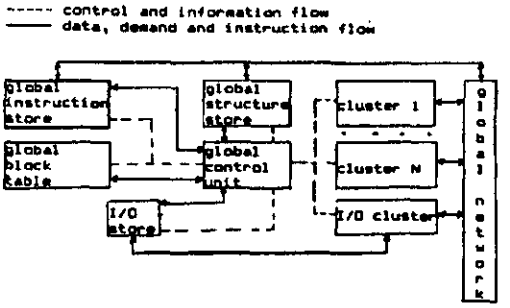


Figure 12: Global level of Fault Tolerant Dataflow Machine (FTDFM)

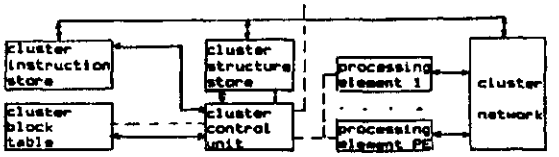


Figure 13: Cluster level of FTDFM

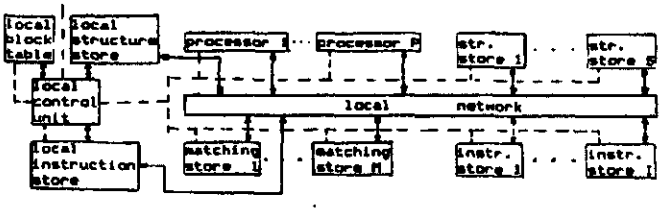


Figure 14: Processing Element of FTDFM

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

object.

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

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

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

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

5 CONCLUSION

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

6. References

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

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

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

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

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

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

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

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

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

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

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

ALTERNATIVNI POSTUPAK ODREĐIVANJA TIPOVA ENTITETA I AKCIJA U JSD METODI PROJEKTOVANJA INFORMACIONIH SISTEMA

INFORMATICA 4/87

Dubravko Živković
Savezna direkcija za promet i rezerve proizvoda sa posebnom namenom, Beograd

UDK 681.3:519.863

ABSTRACT. This paper presents alternative way of determining an initial set of entities and actions in the Jackson System Development - JSD - method of information systems design. The proposal is based on analysis of system input documents insted of determining entities and actions by studying kinds of words from system description as suggested by Jackson in /JACK 83/. This approach has useful application in design of busines oriented information systems using JSD method.

SAŽETAK. Ovaj rad pokazuje alternativni postupak kojim se određuje inicijalni skup entiteta i akcija u Jackson System Development - JSD - metodi projektovanja informacionih sistema. Predlog se zasniva na analizi ulaznih dokumenata u sistem umesto na odredivanju entiteta i akcija proučavanjem vrsta reči iz opisa sistema, što sugeriše Jackson u /JACK 83/. Ovaj pristup ima korisnu primenu kada se projektovanje poslovno orijentisanih informacionih sistema vrši JSD metodom.

1. UVOD - JSD METODA PROJEKTOVANJA INFORMACIONIH SISTEMA

Jackson System Development ili skraćeno JSD metoda projektovanja informacionih sistema (/JACK 83/, /CAME 86/) nastala je iz JSP metode projektovanja programa, (/JACK 75/, /INGE 79/), primenom istih principa na širu klasu problema. Po JSD metodi, projektovanje počinje fazom specifikacije sistema, koja se odvija kroz prvih pet koraka, a završava se fazom implementacije koja je ujedno i zadnji, šesti, korak JSD procedure razvoja projekta.

Koraci JSD procedure su:

1. Entiteti i akcije
2. Struktura entiteta
3. Inicijalni model
4. Funkcije
5. Vremensko usklađivanje
6. Implementacija

Na slici 1. ovi koraci su prikazani kao listovi stabla, a redosled njihovog izvođenja definisan je prefiks prolazom kroz stablo.

U praksi, međutim, razvoj sistema ne odvija se striktno po opisanoj proceduri, kretanjem od njenog jednog do drugog koraka, bez potrebe za povremenim povratkom na neki od prethodno predenih. Iteracije svakako postoje, ali metoda raspolaže mehanizmima da ih svede na minimum.

2. ODREĐIVANJE TIPOVA ENTITETA I TIPOVA AKCIJA

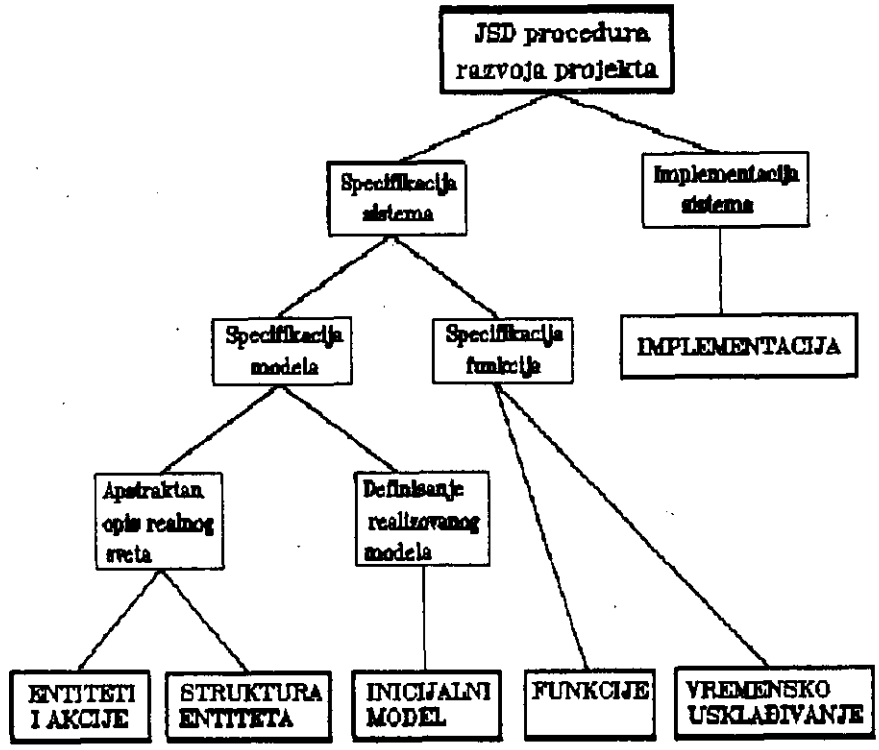
Prvim korakom JSD procedure - entiteti i akcije - počinje opisivanje realnog sveta koje se okončava u koraku struktura entiteta. Naime, proučavanjem raspoložive dokumentacije, stručne literature, intervjuisanjem korisnika i ostalim načinima upoznavanja sa problemom, projektant nastoji da identifikuje što veći broj grupa, pojedinaca, dokumenata, organizacionih delova, jednom rečju objekte u sistemu, koje može proglasiti entitetima[1])

---

1) Značenje pojma "entitet" u JSD povezano je sa idejom procesa u modelu i idejom vremenskog uređenja njegovih akcija. Zbog toga se i razlikuje od značenja koje se, za isti termin, sreće u vezi sa bazama podataka, gde je entitet statičan, tj. niti provodi niti je izložen dejstvu akcija.

(kandidati za entitete). Osim potencijalnih entiteta, proučavanjem događaja koji se odigravaju u realnom svetu, projektant formira i listu mogućih akcija.

Rad na projektu, kao što je rečeno, projektant započinje izborom kandidata za entitete i akcije. Da bi spisak potencijalnih entiteta i akcija bio što širi, jer je to važan zahtev u ovoj fazi rada, a da se mogućnost pogrešnog izbora svede na minimum, Jackson



Slika 1. Procedura razvoja JSD projekta.

projektantu predlaže jednostavan postupak: svaka imenica koja se javlja u opisu realnog sveta identifikuje potencijalan entitet, a svaki glagol moguću akciju. U toj preporuci se pod "opisom" podrazumevaju sve forme upoznavanja sa područjem interesovanja. Preporuka se zasniva na sličnosti između JSD modela i prirodnog jezika, tj. na predpostavci da ljudi svoje misli izražavaju u terminima entiteta, akcija koje ti entiteti vrše i/ili čijem su dejstvu izloženi. Međutim, ovaj predlog propraćen je komentarom da je potrebno obratiti pažnju na činjenicu da mnoge imenice imaju svoju glagolsku formu (glagolske imenice) i obrnuto, što može izazvati određene probleme. Osim toga, može se ponekad dogoditi da neki važan entitet nema ime, tj. da se u opisu pojavljuje u obliku okolišne, zaobilazne fraze, što otežava njegovu identifikaciju.

Ovakav pristup, pogotovu kada su u pitanju složeni sistemi, može generisati obimnu listu kandidata za entitete i akcije, sa kojom bi dalji rad bio znatno otežan. Problem se može donekle ublažiti na taj način što se već u procesu formiranja spiska pristupa sa određenom dozom kritičnosti, tako da se u spisak ne uključuju objekti ili događaji za koje je evidentno da nisu interesantni.

Sličnu proceduru, premda u drugom kontekstu, predlažu i drugi autori (/CHEN 83/, /BOOC 83/).

Po okončanju ovog posla, dobijeni spisak potencijalnih entiteta i akcija treba suziti, imajući u vidu sledeća pravila:

1. Opšta pravila za izbor akcija:

   - Akcija se uvek dešava u određenom vremenskom trenutku i u načelu nema trajanje. Na primer, u projektu LD, "REDOVAN-RAD" ne može biti dopustiva akcija entiteta "RADNIK" pošto rad radnika u redovnom radnom vremenu ima atribute koji se direktno odnose na trajanje redovnog rada, osim ako je vremenska skala projekta tako određena da se čitav period vremena u kome je radnik radio smatra kao trenutni događaj.
   - Za akciju može biti odabran događaj koji se dešava u realnom svetu, a ne u sistemu. U tom svetlu, na primer, "OBRAČUN-LD" ne može biti važeća akcija jer se ne odigrava van sistema.
   - Akcije su nedeljive, tj. ne mogu se dekomponovati na podakcije ili neke druge sastavne delove.

2. Opšta pravila za izbor entiteta:

   - Entitet mora provoditi ili biti izložen dejstvu akcija.
   - Entitet mora postojati u realnom svetu van računara, a karakteriše ga određeno trajanje.
   - Entitet karakteriše uređenost akcija, odnosno definisan redosled po kome se akcije odvijaju u vremenu (struktura entiteta), a koja se prikladnim sredstvima može opisati.
   - Stanje entiteta se menja tokom vremena.

Osim nabrojanih opštih kriterijuma, projektant je slobodan u izboru posebnih koji su prihvatljivi za projekat koga razvija, imajući u vidu da entitet ili akciju treba uključiti u konačan spisak entiteta i akcija ako se planira da sistem u eksploataciji treba da proizvodi ili koristi informacije o njima.

Odabranim tipovima entiteta i akcija projektant daje kratka i razumljiva imena. Međutim, opisivanje entiteta i akcija samo

njihovim imenima nije dovoljno, jer bi različite osobe mogle na različite načine ova imena tumačiti. Zbog toga se još zahteva da akcije, pored naziva, budu dodatno opisane sa:

1. Kratkim neformalnim opisom koji ima za cilj detaljniju specifikaciju semantike naziva akcije.
2. Atributima, čime se značenje akcije još bliže određuje. Entiteti, međutim, ne zahtevaju dodatan opis jer su dovoljno opisani akcijama u njihovom sastavu.

Glavni dokumenat koraka entiteti i akcije je (pročišćen) spisak tipova entiteta i tipova akcija. Akcije su, saglasno zahtevu metode, dodatno specificirane opisom i atributima. Sa ovim spiskom kao ulazom, započinje rad na sledećem koraku JSD procedure.

3. ALTERNATIVNI POSTUPAK ZA ODREĐIVANJE TIPOVA ENTITETA I AKCIJA

U ovom odeljku biće prezentiran alternativni postupak za odredivanje tipova entiteta i tipova akcija koji se može koristiti prilikom projektovanja poslovno orijentisanih informacionih sistema (IS). U osnovi tog postupka je analiza dokumenata koji iz raznih izvora ulaze u sistem. Ovakav pristup ima opravdanje u činjenici, da poslovno orijentisani sistemi mogu preći iz jednog stanja u drugo samo ako postoji odgovarajući dokumenat koji opravdava ili registruje takvu promenu. Sa druge strane, promena stanja sistema u JSD terminima nastaje pod dejstvom akcija određenih tipova entiteta. Iz toga proizlazi da je razumno predpostaviti postojanje određenog stepena korelacije između skupa akcija u JSD modelu i skupa dokumenata u posmatranom poslovnom sistemu. Dalja razmišljanja u tom pravcu navode na zaključak da se, u ovoj fazi, izvori dokumenata mogu poistovetiti sa JSD entitetima. Inicijalan skup tipova entiteta i tipova akcija, koji je dobijen na opisan način, potrebno je dalje analizirati korišćenjem koncepta funkcionalne zavisnosti, a pridržavajući se i opštih JSD kriterijuma za izbor entiteta i akcija, da bi se dobio njihov konačan skup.

Ovakav postupak, nesumnjivo, daleko brže dovodi do završetka prvog koraka JSD procedure od onoga koji je predložen u referenci /JACK 83/, jer je broj dokumenata i izvora njihovog nastajanja daleko manji od broja imenica, glagola i drugih vrsta reči pomoću kojih se opisuje funkcionisanje dotičnog poslovnog sistema. Samim tim je i posao projektanta manje obiman, njegov rad je produktivniji.

U ovom odeljku, dakle, biće izložen postupak za odredivanje tipova entiteta i akcija prilikom projektovanja poslovno orijentisanih informacionih sistema koji se zasniva na analizi ulaznih dokumenata. Nakon neophodnih objašnjenja, izlaganja će biti zaključena definisanjem procedure koju projektant u radu treba da sledi.

3.1. ODREĐIVANJE TIPOVA ENTITETA I AKCIJA ANALIZOM ULAZNIH DOKUMENATA

Osnovni zadatak projektanta, za vreme dok je angažovan na upoznavanju sa sistemom, sastoji se u tome da identifikuje sve tipove ulaznih dokumenata koji su važni za rad sistema. Uporedo sa tim, on treba da odredi i objekte iz okruženja koji predstavljaju izvore, uzročnike, njihovog nastanka. Na taj način biće formirana inicijalna lista tipova akcija i entiteta. Akcije će u ovoj fazi biti poistovećene sa odgovarajućim tipom dokumenta, a entiteti sa uzročnikom njihovog nastanka. Opravdanost za takav postupak može se ilustrovati na mnoštvu primera. U sistemu LD, na primer, poverilac upućuje u RO administrativnu zabranu, pa se zbog toga može tretirati kao potencijalni tip entiteta koji provodi akciju tipa "zabrana". Sličan je primer, takođe, sa dobavljačem - tip entiteta - koji je poslao fakturu - tip akcije - za isporučenu robu, itd. itd.

Dokumenat koga projektant izgraduje dok radi na ovom zadatku, sličan je finalnom dokumentu iz koraka entiteti i akcije. Razlika postoji jedino u načinu tretiranja atributa akcija, kojima se, kada je u pitanju metod zasnovan na analizi vrsta reči, pridaje manja važnost u odnosu na onu koju imaju kada se entiteti i akcije određuju na osnovu ulaznih dokumenata. Zbog toga, skup atributa neke akcije može, u prvom slučaju, biti i nekompletan, jer služi prvenstveno zbog toga da poboljša semantiku akcije. Kada se entiteti i akcije određuju na osnovu ulaznih dokumenata, tada se svi podaci koji se nalaze na određenom dokumentu tretiraju kao atributi akcije koja je izvedena na osnovu tog dokumenta. Dodatan zahtev, koji se tom prilikom pred projektanta postavlja, sastoji se u tome da treba izvršiti unifikaciju naziva atributa, odstranjivanjem tzv. sinonima i homonima.

Skup potencijalnih entiteta i akcija koji je određen poistovećivanjem ulaznih dokumenata sa akcijama i uzročnicima njihovog nastanka sa entitetima, u nekim slučajevima nije u dovoljnoj meri precizan. Ta nepreciznost može se odnositi kako na dobijene tipove entiteta tako i na tipove akcija. Regularni tipovi entiteta su u takvim slučajevima podložni pogrešnoj identifikaciji, dok je za skrivene tipove entiteta karakteristično da su jednostavno izostavljeni iz razmatranja. Nepreciznosti prilikom odredivanja tipova akcija najčešće su takve prirode, da se manifestuju nerazlikovanjem zajedničkih akcija više tipova entiteta. Od projektanta se, radi toga, očekuje da sa inicijalnim skupom entiteta i akcija nastavi rad. U tom smislu njegova pažnja treba da se usredsredi na atribute akcija, sa ciljem da se identifikuju funkcionalne zavisnosti koje postoje među atributima. Na osnovu tih zavisnosti, potrebno je odrediti ključeve relacija koje korespondiraju inicijalno određenim akcijama. Nakon toga, provodi se, eventualna, korekcija identifikovanih tipova entiteta i preciznije određuju tipovi akcija.

Zbog kontinuiteta izlaganja, u nastavku je dat primer koji ilustruje postupak odredivanja ključa realacije kada su poznate funkcionalne zavisnosti, a za detalje s tim u vezi potrebno je konsultovati literaturu, npr. /ULLM 82/, /ALAG 84/ ili /LAZA 85/.

PRIMER.

Posmatra se relaciona šema:

$$R(A_1, A_2, A_3, A_4, A_5, A_6)$$

za koju važi sledeći skup funkcionalnhih zavisnosti F:

1. $A_1 \longrightarrow A_4$
2. $A_1 \longrightarrow A_5$
3. $A_2 \longrightarrow A_3$
4. $A_2 \longrightarrow A_6$

Po definiciji, ključ relacije treba da zadovolji osobine jednoznačnosti i

neredundantnosti. Ove osobine zadovoljava jedino par atributa A_1 A_2, pa oni predstavljaju ključ relacije R. ■

Kada se svim kandidatima za akkcije, odnosno relacijama koje ih predstavljaju, u nekom konkretnom primeru u praksi odrede ključevi, dobijeni rezultati mogu se prezentirati tabelarno na način koji je prikazan na slici 2.

Ključevi / Akcije	K_1	K_2	K_3	...	K_n
A_1	x	x			
A_2		x	x		
.					
.					
.					
A_m					x

Slika 2. Tabela koja sumira podatke o ključevima relacija koje su izvedene iz ulaznih dokumenata u sistem.

Kada su u tabelu uneženi podaci o svim akcijama, projektant je izvršio neophodne pripreme za odlučujuću fazu odredivanja entiteta i akcije.

Skup entiteta odreden je skupom ključeva koji su upisani u zaglavlju tabele (K_1, K_2,..., K_n). Svaki ključ identifikuje jedan entitet, pa u apstraktnom primeru, čija je tabela prikazana na slici 2., postoji n tipova entiteta. Na osnovu podataka o ključevima, medutim, nije moguće odrediti da li neki entitet pripada kategoriji regularnih ili skrivenih entiteta. Da bi se napravila ova razlika medu entitetima, može se postupiti na dva načina.

Prvi se zasniva na analizi preslikavanja koja postoje izmedu ključnih atributa. U tom slučaju, ako izmedu dva ključa postoji preslikavanje (1:M), tada je entitet predstavljen ključem koji u preslikavanju učestvuje sa kardinalnošću M, kandidat za skriveni tip entiteta. U sistemu LD, na primer, izmedu matičnog broja radnika (ključ K_1) i broja partije koji predstavlja deo ključa entiteta ODBITAK (ključ K_2), postoji preslikavanje K_1 <——>-> K_2, tj. preslikavanje (1:M). Ovaj tip preslikavanja postoji i izmedu broja partije i šifre poverioca (ključ K_3), za koje se takode može pisati K_3 <——>-> K_2. Zbog toga se odbitak može tretirati kao kandidat za skriveni tip entiteta.

Drugi način u analizu uključuje akcije, odnosno preciznije, akcije i pridružene ključeve. Po tom postupku, ako neki entitet E_1, koji je predstavljen ključem K_1, nema ni jednu akciju kojoj je pridružen samo jedan ključni atribut, u konkretnom slučaju ključni atribut K_1, predstavlja kandidat za skriveni tip entiteta. Do tog zaključka se dolazi ako se ima u vidu da skriveni tip entiteta karakterišu isključivo zajedničke akcije sa drugim tipom ili tipovima entiteta. Zajedničke akcije su, dakle, one kojima je u tabeli pridružen složeni ključ. Ako se kao primer i ovog puta uzme sistem LD i zadrže uvedene oznake za ključeve, tada će administrativna zabrana (koja je u prethodnim izlaganjima bila predstavljena akcijom ZABRANA) biti identifikovana ključevima K_1, K_2 i K_3. Slična je stvar i sa ostalim tipovima akcija za koje je u tabeli naznačeno da sadže K_2. Svi oni, pored K_2, imaju i druge atribute (K_1 ili K_3) koji formiraju složeni ključ. S druge strane, nalog da se poverilac uključi u registar poverilaca (akcija EVIDENTIRANJE) ili odluka o prijemu radnika u radni odnos (akcija ZASNIVANJE-RADNOG-ODNOSA), imaju ključeve koji se sastoje od jednog atributa, K_3 i K_1 respektivno. Samim tim, u sistemu LD, projektant ima osnova da zaključi da su RADNIK i POVERILAC regularni, a da je ODBITAK skriveni tip entiteta.

Važno je primetiti da entitet tipa ODBITAK nije ni identifikovan, niti je to mogao biti, u prvoj fazi rada. Do te spoznaje moglo se doći tek nakon analize funkcionalnih zavisnosti.

Na sličan način bi se provodila analiza imajući u vidu i ostale tipove dokumenata odnosno akcija, tako da njenim okončanjem projektant dobija konačnu listu entiteta i akcija sa kojom može otpočeti aktivnosti iz sledećeg koraka JSD procedure.

Razume se da je projektant slobodan u izboru postupka koje će u radu koristiti, a, može ih i kombinovati, tako da rezultate do kojih je došao na jedan način proveri pomoću drugog. Medutim, bilo kakvo da je njegovo opredeljenje, on treba u toku rada uvek da ima na umu i opšte JSD kriterijume za izbor entiteta i akcija.

Da bi skup entiteta i akcija odreden analizom ulaznih dokumenata bio kompletan, potrebno je da projektant u obzir uzme sve tipove dokumenata koji su bitni za rad odredenog poslovnog sistema. To neće uvek biti lako, a razlozi koji do toga dovode mogu biti različiti. Jedan od njih je svakako neiskustvo projektanta, a zatim sledi njegovo nedovoljno poznavanje oblasti poslovanja, za koju treba projektovati IS u fazi kada je potrebno odrediti entitete i akcije. Medutim, isti problemi postoje i ako se entiteti i akcije odreduju postupkom koji je predložio Jackson. U slučaju poslovno orijentisanih informacionih sistema, oni su čak i izraženiji ako se umesto analize ulaznih dokumenata koristi neki drugi postupak. Problemi takode mogu nastati i u slučaju kada je stepen organizovanosti konkretnog poslovnog sistema niži od uobičajenog, kada u sistemu postoje mnoge priručne evidencije koje nemaju zvanični karakter, kada iz okruženja dolazi loše kreirana dokumentacija itd. Istraživanje odnosa apstraktnog skupa ulaznih dokumenata ("idealan" skup dokumanata), koji bi omogućio jednoznačnu identifikaciju entiteta i akcija u nekom sistemu, prema realnom skupu dokumenata, verovatno bi rezultiralo interesantnim zaključcima koji bi se koristili u procesu odredivanja entiteta i akcija. U ovom pravcu bi se mogla odvijati dalja istraživanja!

Uz malo dodatnog truda projektant može, ako entitete i akcije odreduje postupkom analize ulaznih dokumenata, jednim potezom obaviti dva posla, tj. definisati skup entiteta i akcija i izvršiti pripreme za odredivanje strukture entiteta. Radi se o tome da je potrebno, u toku upoznavanja sa dokumentima i njihovim izvorima, prikupiti podatke i o tragovima potencijalnih entiteta. Do tih podataka on može doći pažljivijim proučavanjem postojećeg sistema evidencije koja služi za registrovanje primljenih dokumenata.

Ako se u postupku odredivanja entiteta i akcija analizom ulaznih dokumenata, angažovanje projektanta na uočavanju tragova smatra pozitivnim, to se bez izvesne ograde ne može prihvatiti u vezi zahteva da je potrebno, ako se postupak želi korektno primeniti, proučiti funkcionalne zavisnosti koje postoje medu atributima. Naime, poznavanje funkcionalnih zavisnosti nesumnjivo je korisno i sigurno će biti iskorišćeno za projektovanje baze

podataka, ali se angažovanje projektanta na tom poslu u koraku entiteti i akcije smatra preuranjenim. Taj zahtev je zbog toga i najveći nedostatak ovog postupka.

3.2. PROCEDURA ZA ODREĐIVANJE TIPOVA ENTITETA I AKCIJA

Na kraju, postupak za odredivanje entiteta i akcija metodom analize ulaznih dokumenata biće izložen u formi procedure. Ona treba da olakša primenu opisanog postupka u praksi, pošto doprinosi da korak entiteti i akcije dobija strukturniju formu.

Procedura se sastoji iz 6 koraka i to:

1. korak. Proučavanjem dokumenata koji postoje u RO, čitanjem stručne literature, iz saradnje sa korisnikom i drugih raspoloživih izvora, potrebno je identifikovati različite vrste dokumenata koji iz okruženja dolaze u poslovni sistem čiji rad treba automatizovati. Za svaki tip dokumenta potrebno je odrediti mesto ili osobu koja je odgovorna za njegovo kreiranje.

2. korak. Poistovetiti svaki dokumenat sa odredenim tipom akcije, a entitete sa mestima odnosno osobama koji dokumente generišu. Rezultate treba prezentirati u formi spiska koji je sličan konačnom dokumentu za korak entiteti i akcije, ali se u ovoj fazi od projektanta ne zahteva da akcije dopuni tekstualnim opisom. Medutim, atributima akcija treba pokloniti veću pažnju. U tom smislu podaci sa dokumenta postaju atributi akcija, a od projektanta se očekuje da njihove nazive ujednači i izbegne pojavu sinonima i homonima.

3. korak. Identifikovati skup funkcionalnih zavisnosti nad skupom atributa svake akcije.

4. korak. Na osnovu poznatih funkcionalnih zavisnosti odrediti ključeve relacija koje korespondiraju inicijalno odredenim akcijama odnosno ulaznim dokumentima. Rezultate prezentirati u tabelarnoj formi na takav način da se za svaku akciju naznači njen ključni atribut ili atributi.

5. korak. Koristeći se tabelom koja je uradena u 4. koraku, preciznije definisati skup entiteta i akcija od onoga koji je identifikovan u drugom koraku procedure. Konačna odluka o tome da li će neki od kandidata za entitete i akcije biti odabran za nastavak rada, koji će od njih biti uključen u finalni dokumenat ove procedure, odnosno finalni dokumenat koraka entiteti i akcije, zavisi i od toga da li je on u skladu sa opštim JSD kriterijumima za odredivanje entiteta i akcija.

6. korak. Kada su odabrani entiteti i definisane njihove akcije, odluke koje su tom prilikom donešene potrebno je na prikladan način dokumentovati a to se postiže izradom spiska entiteta i akcija. Ovaj dokumenat treba da udovolji zahtevima JSD metode, što znači da svaka akcija i entitet treba da imaju kratko i razumljivo ime i da se akcije, osim atributima, dopune i kraćim neformalnim opisom.

Definisanjem procedure za odredivanje entiteta i akcija metodom analize ulaznih dokumenata, okončava se i izlaganje koje se na ovu temu odnosi.

4. ZAKLJUČAK

U ovom radu predložen je alternativni postupak za odredivanje tipova entiteta i akcija u JSD metodi projektovanja IS. On se zasniva na analizi ulaznih dokumenata u sistem, a ne na analizi vrsta reči sa kojim je sistem opisan, kao što predlaže Jackson. Kod poslovno orijentisanih IS predloženi postupak brže dovodi do rešenja, a definisana procedura prvom koraku JSD procedure projektovanja IS, "Entiteti i akcije", daje strukturniju formu.

Nedostatak predloženog postupka je preuranjena analiza funkcionalnih zavisnosti medu atributima, koja se provodi da bi se preciznije odredio skup entiteta i akcija.

LITERATURA

/ALAG 84/ S. Alagić, "Relacione baze podataka", Svjetlost, Sarajevo, 1984.

/BOOC 83/ G. Booch, "Software Engineering with Ada", The Benjamin/Cummings Publishing Company, 1983.

/CAME 86/ J. R. Cameron, "An Overview of JSD", IEEE Transactions on Software Engineering, No 2, 1986.

/CHEN 83/ P. P. Chen, "English Sentence Structure and Entity-Relationship Diagrams", Information Sciens, Elsevier Science Publishers, B. V. (North-Holland), 1983.

/INGE 79/ L. Ingevaldsson, "JSP - A Practical Method of Program Design", Studentlitteratur, 1979.

/JACK 75/ M. Jackson, "Principles of Program Design", Academic Press, 1975.

/JACK 83/ --- , "System Development", Prentice-Hall, 1983.

/LAZA 85/ B. Lazarević i dr. "Projektovanje informacionih sistema", Naučna knjiga, 1985.

/ULLM 82/ J. D. Ullman, "Principles of Database Systems", Pitman Publishing, Ltd., 1983.

UDK 681.3.06 MODULA – 2

Andrej Brodnik, Marjan Špegel, Tadej Lasbaher
Institut »Jožef Stefan«, Ljubljana

Ta prispevek obsega primerjavo module-2 s programskimi jeziki pascal, ada in c, splošen opis prevajalnika za modulo-2 in opis najbolj vidnih prevajalnikov za različne operacijske sisteme. Sklepni del prispevka je namenjen kratkemu opisu zanimive združitve zmožnosti funkcionalnega in proceduralnega jezika (module-2 in prologa). V dodatku navajamo dva primera programov v moduli-2, ki naj bi predstavila tako osnovna načela sistemskega programiranja, kot postopek modularnega programiranja. Ta prispevek je nadaljevanje prej objavljenega članka in zaokrožuje kratko predstavitev jezika modula-2.

Programming With Modula-2 In this paper the comparisson of Modula-2 with Pascal, Ada and C is given. Further, the general overview of compiler is presented with some specific examples for several operating systems. Finally, the very smart connection of functional and procedural language (Prolog and Modula-2) is given. The appendices contain two examples of programmes which should give a flavour of modular and system programming in Modula-2. This paper with prevecously published one closes the sketching of Modula-2 programming language.

1. UVOD

Če je bil prvi del prispevka namenjen jeziku samemu, potem poskušamo v tem delu jezik postaviti nekam v okolje. Tako navajamo nekatere podatke, ki bodo pomagale oceniti razmerje module-2 do drugih programskih jezikov. Temu so namenjena prva tri poglavja.

V nadaljevanju opisujemo sam princip prevajanja programov napisanih v moduli-2 in nivojski ustroj splošnega prevajalnika. V zaključnih dveh poglavjih pa poskušamo prikazati nekaj zelo zanimivih primerov uporabe jezika na dokaj različnih primerih zunaj in pri nas.

V posebnem dodatku vam predstavljamo dve lupini programov, ki sta napisana v moduli-2 in sicer razporejevalnik procesorskega časa in program za pogovor med dvema terminaloma.

2. PASCAL IN MODULA-2

Že v prejšnjem sestavku o moduli-2 smo opozorili na to, da je modula-2 kakovostna nadgraditev pascala. Po njem je podedovala predvsem bločno zgradbo in vse osnovne tipe stavkov, od njega pa se bistveno razlikuje predvsem v naslednjih pogledih [5]:

a) gradnik "modul" omogoča
   - ločeno prevajanje programskih in podprogramskih modulov
   - enostavno gradnjo modulov s programskimi orodji
   - knjižnice, ki so podprte s sistemom jezika

b) nizkonivojsko ali sistemsko programiranje
   - dostop do poljubnega stalnega mesta v pomnilniku
   - dostop do posameznih procesorskih registrov
   - pisanje prekinitvenih podprogramov
   - pisanje splošnih sorutin

c) podprogram je definiran kot poseben tip

d) sintaktične razlike

Razlike pod točkami a, b i c so bistvene in so semantične narave, medtem ko so razlike pod točko d predvsem oblikovne oziroma sintaktične narave. Ker pascal ne pozna pojma modula in ker ne omogoča sistemskega programiranja, ga na teh dveh področjih tudi ne moremo primerjati z modulo-2.

2.1 Podprogram kot tip

Podprogram kot tip je v zelo omejenem smislu sicer poznal že pascal in sicer kot parameter podprograma [6, 16], ki je moral biti brez parametrov. V moduli-2 je povsem pravilna definicija naslednjega tipa:

```
TYPE
  ProcType = PROCEDURE (Type1, Type2): Type3;
```

Parametri podprogramov so prav tako lahko kakršnegakoli tipa.

2.2 Sintaktične razlike

Modula-2 je podobna pascalu tudi po zunanjem videzu. Vendar vseeno obstajajo nekatere razlike med obema jezikoma in jih navajamo tukaj:

a) modula-2 loči med velikimi in malimi črkami. Na primer: **ena** ni ista spremenljivka kot **ENA**
b) pascal pozna dva tipa komentarjev, medtem ko je v moduli-2 dovoljen samo eden, ki pa ga lahko gnezdimo. Tako v moduli-2 { to ni komentar }, (* ampak to *), (* ki je lahko (* gnezden *) *).
c) modula-2 pozna več osnovnih tipov kot pascal.
d) v moduli-2 so dovoljene različne okrajšave za veččrkovne operatorje. Znak neenakosti <> se zapiše kot #; logični **AND** je dovoljeno okrajšati v &.

e) modula-2 ne pozna sestavljenega stavka, kajti izkaže se, da je nepotreben. Tako je besedica **begin** v naslednjem primeru povsem odvečna:

```
pascal:
  for i:=1 to 10 do
    begin
      stavek1;
      stavek2;
       .  .  .
      stavekn
    end;

modula-2:
  FOR i:=1 TO 10 DO
    stavek1;
    stavek2;
     .  .  .
    stavekn
  END;
```

f) modula-2 pozna brezpogojno zanko (LOOP stavek).
g) stavka IF c1 THEN s1 ELSIF c2 THEN s2 ELSE s3 END; pascal ne pozna.
h) v moduli-2 se vsi stavki zaključijo z rezervirano besedo END, kar je razvidno že iz primera za sestavljene stavke.
i) v moduli-2 je povsem prepovedan **goto** stavek.
j) razlika je pri definiciji funkcijskih podprogramov.
k) modula-2 ne pozna standardnih podrogramov get, put, read, readln, write, writeln za delo z vhodno izhodnimi enotami.
l) modula-2 ne pozna tipa **file**.

Tu naštete razlike seveda niso edine, vendar so po našem mnenju najpomembnejše.

3. ADA IN MODULA-2

Ada [21, 24] in modula-2 sta po letnici rojstva vrstnici, zato ni nič čudnega, če v obeh jezikih najdemo nekatere bistvene podobnosti, kljub bistvenim sintaktičnim, zlasti pa semantičnim razlikam. Zelo pomembna razlika je že kar velikost jezika. Kot navaja Pomberger [20], obsega formalna definicija ade več sto strani, medtem ko za modulo-2 zadostuje že dobrih 25. Temu primerna je tudi razlika v velikosti prevajalnikov in kode, ki jo naredi prevajalnik. Zato na majhnih računalnikih popolnih prevajalnikov za ado praktično ni, medtem ko jih za modulo-2 najdemo kar nekaj [9, 22].

Podobnosti med ado in modulo-2 si oglejmo na primerjavi podatkovnih in ukaznih struktur, na primerjavi modularnosti in možnosti ločenega programiranja ter na primerjavi sposobnosti jezikov za nizkonivojsko programiranje in za vzporedno izvajanje postopkov.

3.1 Podobnost podatkovnih in ukaznih struktur

Ada in modula-2 poznata podobne osnovne podatkovne tipe, le da ada obvlada tudi števila v takoimenivani stalni vejici. V moduli-2 realiziramo ta števila s sestavljenim tipom:

```
FixedPoint = RECORD
               num: CARDINAL;
               base, delta: REAL
             END;
```

Razen tega ada pozna podtipe, ki jih v moduli-2 lahko vodimo kot delne tipe, po drugi strani pa ada ne pozna tipa PROC. Poleg tega modula-2 pozna tip množice, ki ga v adi realiziramo s tabelo:

```
BITSET is array (0..15) of BOOLEAN;
```

in podatkovni tip ADDRESS, ki ga lahko v adi definiramo kot

```
ADDRESS is access WORD;
```

Razlike med ukaznimi strukturami jezikov so bistveno manjše. Oba jezika poznata podobne osnovne stavke, vendar ima ada pri tem nekaj več možnosti. Modula-2, na primer, ne pozna stavka **declare** (nadomestimo ga lahko z uporabo novega modula) in **delay** stavka (nadomestimo ga s klicem ustreznega podprograma). Modula-2 tudi povsem prepoveduje uporabo GOTO stavka, ki pa je v adi dovoljen. Naslednji stavek, ki ga najdemo v adi in ne v moduli-2, je **null** stavek. V resnici obstaja tudi v moduli-2, vendar ga tam ne napišemo, saj je predstavljen s praznim stavkom, ki ima obliko samo podpičja ali celo ničesar. Kako v moduli-2 nadomestimo **raise** stavek iz ade, pa bomo opisali na primeru nadomeščanja strukture **exception** in njene uporabe.

Ada je tudi precej močnejša kot modula-2 pri uporabi podprogramov, vendar lahko prednosti ade dosežemo s pametnim programiranjem tudi v moduli-2. Tako adino prednost, da je rezultat funkcijskega podprograma lahko tudi strukturiran tip, v moduli-2 nadomestimo s tem, da izračunani rezultat vrnemo preko parametra, ki je bil klican po naslovu (referenci). Pač pa v moduli-2 ne obstaja možnost ponovnega definiranja osnovnih operacij, kot so na primer množenje, seštevanje in podobno (infiksne operacije), in si moramo zato pomagati s klici ustreznih podprogramov.

3.2 Posebnosti in osnovni podprogrami

V adi obstajata dve strukturi, ki jih modula-2 nima: prva je **exceptions** (posebnosti) in je namenjena predvsem za obravnavanje posebnih stanj (na primer napak) v izvajanju programa. Te strukture v moduli-2 v bistvu sploh ne potrebujemo; če smo pravilno razbili naš program, moduli sami prestrežejo vse napake in jih ostali program sploh ne zazna.

Druga struktura, ki jo najdemo samo v adi, izhaja iz dejstva, da je nek algoritem (recimo za vodenje sklada) povsem neodvisen od tega, kakšni elementi so v skladu. V ta namen pozna ada takoimenovane osnovne podprograme (**generic**), kar je v moduli-2 še najtežje nadomestiti. Pri tem si pomagamo z splošnimi tipi (ADDRESS, WORD, BITSET, ARRAY) in predvsem z možnostjo ločenega prevajanja, kjer uporabimo za dano različico dani modul.

3.3 Modularnost in ločeno prevajanje

Načela modularnosti in možnosti ločenega prevajanja smo si že ogledali v četrtem poglavju prvega dela našega prispevka. Na tem mestu pa si oglejmo le podobne strukture ade.

Modularno programiranje in ločeno prevajanje omogoča ada enako dobro kot modula-2. Pojem modula nadomešča v adi pojem paketa (**package**). Kot modula-2 tudi ada loči med definicijskim in delovnim modulom. Definicijski modul je zaznamovan z besedo **package**, delovni pa z **package body**. Kot v moduli-2 so tudi v adi definirane strukture za oddajo in prevzem tipov, spremenljivk in podprogramov (**use**, **private**). Bistvena razlika med obema jezikom pa je v programskem modulu, ki je pri adi podprogram v moduli-2 pa zopet modul.

V sistemu jezika modula-2 po definiciji obstaja še ena bistvena prednost pred ado. Sistem jezika modula-2 namreč v postopku prevajanja in povezovanja sam preverja, da je modul, ki prevzema neko strukturo iz drugega modula, preveden kasneje. Tega ada v osnovi ne pozna.

3.4 Nizkonivojsko programiranje in vzporednost

Na tem področju je ada v definicijah precej šibkejša od module-2. Adino spremenljivko, na primer, ni možno namestiti na željeno mesto v pomnilniku in programer ima na voljo le še dinamičen način dostopa.

Za programiranje vzporednih postopkov ima ada enoto **task**, ki je podobna procesu v moduli-2. Prednost pred modulo-2 pri vzporednem programiranju pa predstavljajo adine ukazne strukture, ki omogočajo vsklajevanje vzporednih procesov; v moduli-2 mora programer te podprogram napisati sam. Kot modula-2 tudi ada omogoča posluževanje prekinitev.

Collins [14] pravi, da bi bila primerjava module-2 in ade nekaj podobnega kot primerjava med pascalom in PL/1: kar je prepuščeno v moduli-2 programerju, je v adi že vgrajeno v jezik. Tu lahko tudi iščemo vzrok obsežnosti ade.

4. C IN MODULA-2

Modula-2 in jezik c sta zelo različna jezika. Zato se bomo omejili le na primerjanje možnosti za programiranje na spodnjem nivoju, torej na nivoju sistemskega programiranja. Razen tega je smiselna primerjava med obema jezikoma v pogledu prenosljivosti programov, medtem ko bi bila primerjava višjih programskih struktur brezpredmetna, ker jih v c-ju ni. Zato v c-ju ne poznamo principov modularnega programiranja in ločenega prevajanja. Tudi v podatkovnih strukturah je c dokaj reven.

Jezik c je nastal hkrati z operacijskim sistemom Unix [7, 23] ter postal osnovni jezik tega močno razširjenega operacijskega sistema. Zato program napisan v c-ju lahko dandanes brez večjih sprememb poženemo na poljubnem drugem računalniku z operacijskim sistemom Unix. Prav prenosljivosti programov je tisti pojem, po katerem je smiselno primerjati modulo-2 in c. Kot pri jeziku c, je torej tudi pri moduli-2 zagotovljena prenosljivost programov, vendar v primeru programov v moduli-2 ne samo med različnimi računalniki z istim operacijskim sistemom, ampak celo med računalniki z različnimi operacijskimi sistemi. Doseči ta cilj v kar se da veliki meri je naloga delovne skupine projekta OSSI [2]. Točneje, izdelati želijo nekakšen vmesnik med modulo-2 in poljubnim operacijskim sistemom.

Jeziku c običajno pripisujejo moč in prednost predvsem pri nizkonivojskem programiranju, vendar zlahka uvidimo, da ima vse te zmožnosti tudi modula-2, ki omogoča celo večji nabor sistemskih operacij.

Največja moč jezika c je v naslavljanju pomnilnika, vendar prav c ne pozna statične definicije spremenljivke na določenem mestu v pomnilniku. Pač pa c lahko poljubno spremenljivko pretvori v kazalec na naslov (to dosežemo z uporabo posebnega znaka *), kar v moduli-2 dosežemo s klicem funkcije ADDRESS. V c-ju lahko vedno ugotovimo naslov v pomnilniku, kjer se nahaja določena spremenljivka, v moduli-2 pa to opravimo s funkcijo ADR.

Ker c izhaja po svojih definicijah iz zbirnika, ki mu poskuša ostati čimbolj podoben, je temu primerno bogat tudi po načinih naslavljanja. Avtomatično dodajanje (x++) ali odvzemanje (x-=i) v c-ju, je v moduli izvedljivo z dodatnim klicem podprograma INC.

V c-ju obstaja poseben način definiranja posameznih spremenljivk, ki prevajalniku pove, naj poskuša shraniti spremenljivke v procesorske registre (**register**). V moduli-2 to ni potrebno, ker so registri neposredno dostopni, drugi cilj omenjene c-jeve strukture - optimizacija kode, pa je v bistvu problem prevajalnika in ne jezika samega.

Delo s posameznimi biti, ki je v c-ju zelo enostavno, je v moduli-2 izvedljivo s spremenljivkami tipa BITSET in klicem podprogramov INCL ter EXCL.

Za razliko od module-2 pa v c-ju ne moremo uporabljati procesov in tudi posluževanje prekinitev po definiciji ni del jezika [1, 7, 19].

Nizkonivojsko in sistemsko programiranje, za kar je c dokaj primeren, je enako dobro podprto tudi v moduli-2. Ob tem pa nam modula-2 ponuja še vse prednosti strukturiranega programiranja, ločenega prevajanja, preverjanja sovpadanja tipov in še cele vrste prednosti, ki jih zahteva sodoben, učinkovit programski jezik, s čimer se c ne more ravno pohvaliti.

5. Nekateri prevajalniki za modulo-2

V tem poglavju si bomo ogledali tri prevajalnike in en interpreter ter z njimi bomo prehodili kratko zgodovino module-2.

4.1 Prevajalnik M2M in M2RT11

Prevajalnik M2RT11 [10, 11, 12, 15] iz ETH Zurich je bil prvi široko uporabljani prevajalnik za modulo-2; pisan je v moduli-2 za računalnik PDP-11 (LSI-11) pod operacijskim sistemom RT-11. Avtor module-2 Wirth [25] je novi jezik dokončno oblikoval in realiziral skozi ta prevajalnik.

Dokumentacija obsega pet zvezkov, ki opisujejo samo sistem jezika brez primerjave s kakšnim drugim jezikom. V njih tudi zasledimo, da obstaja različica tega prevajalnika za operacijski sistem Unix.

Osnovni prevajalnik (M2RT11) je petprehodovni, kakršni so praviloma tudi vsi drugi. Prav počasnost prevajanja, ki izvira iz števila prehodov, je ena največjih slabosti module-2. Ker so vsi novejši prevajalniki nastali prav na podlagi tega osnovnega prevajalnika, si ga oglejmo podrobneje.

Sistem jezika sloni na kosu programa, ki je napisan v zbirniku in se imenuje Run Time System. V njem so vsebovani vsi podprogrami, ki jih oddaja modul SYSTEM in še nekaj splošnih podprogramov. Klic vseh teh podprogramov je izveden preko programskih prekinitev - pasti. Vedno obstaja možnost preprogramiranja tega dela programa tako, da lahko potem poganjamo program brez podlage operacijskega sistema (stand alone).

V sistem je vključen prevajalnik, ki obsega naslednjih pet prehodov:

1. preverjanje sintaktične pravilnost prevajane enote in branje simbolne datoteke vnešenih modulov
2. preverjanje pravilnosti deklaracij in izdelava referenčne datoteke

3.: preverjanje sovpadanja tipov v telesih podprogramov
4.: generiranje kode za izraze
5.: generiranje kode za posamezne stavke in izdelava izhodne datoteke

Kasneje so definirali neko obliko vmesne kode, podobno kot p-kodo pri pascalu, in jo imenovali M-koda. Prvič je bila ta oblika kode uporabljena pri prevajalniku M2M, kjer sta 4. in 5. prehod združena v enega samega in sicer:

4.: generiranje vmesne, M kode

V dokumentaciji je dodan tudi interpreter te kode in njen opis [13] Sicer pa se vsi prehodi izvedejo le pri prevajanju delovnih in programskih modulov, medtem ko so za definicijske module dovolj že prvi trije.

Poleg prevajalnika vsebuje sistem jezika še povezovalnik in iskalnik napak (debugger) ter vrsto uporabniških modulov z orodji, ki omogočajo delo z datotekami in podobno.

Sistem razpolaga z več vrstami datotek, ki jih loči po podaljških:

- .MOD: programski ali delovni modul
- .DEF: definicijski modul
- .LST: izpis prevedenega programa (potrebuje ga iskalec napak)
- .REF: referenčna tabela (potrebuje jo iskalec napak)
- .LNK: izhodna datoteka iz prevajalnika, ki je primerna za povezovanje
- .LOD: izhodna datoteka iz povezovalnika, ki je primerna za izvajanje v okviru sistema
- M2M in M2S: sistemske datoteke

Vsi naslednji primeri prevajalnikov so nekakšne izpeljanke iz osnovnega M2M.

5.2 Modula-2 za Z80 pod CP/M

Tudi ta prevajalnik prihaja iz Švice, je pa precej okrnjena različica, saj ne pozna pojma procesa in vzporednosti delovanja. Razen tega tudi ne omogoča posluževanja prekinitev. Kot nadomestilo pa ohranja povezavo z zbirnim jezikom: uporabnik lahko po želji določene delovne module napiše v zbirniku.

Sistem sestoji iz podobnih delov kot M2M in zopet je priložena množica delovnih modulov, ki omogočajo delo s samim operacijskim sistemom. Dokumentacija te implementacije sistema modula-2 [14] podrobno opisuje razlike med pascalom in modulo-2.

5.3 Modula-2 razvojni sistem

Je interpreter za modulo-2 za računalnik v tem primeru IBM-PC. Med predstavljenimi primeri je to tudi edini interpreter. Nastal je leta 1984 pri tvrdki Modula Corporation. Ta sistem je pravzaprav celoten prevajalnik, ki pa mu manjka peti prehod, torej generacija kode, ki bi bila izvedljiva na ciljnem računalniku. Implementacija module-2 je dokaj celovita, saj iz definicije jezika manjka samo možnost posluževanja prekinitev (podprogram SYSTEM.IOTRANSFER). Ta sistem je še najbolj primeren za nadaljne razvojno delo na moduli-2, predvsem v smeri razvoja in izdelave petega prehoda, ki bi izdeloval resnično izvedljivo kodo. Možno pa bi bilo tudi dodati še en prehod za optimizacijo M-kode, ki bi ga vstavili med četrti in peti prehod.

5.4 SDS-XP

Sistem SDS-XP sodi trenutno med najboljše prevajalnike za modulo-2 za računalnik IBM-PC in operacijski sistem PC-DOS.

Implementacija module-2 je povsem popolna in vključuje možnost nadzora nad prekinitvami ter vzporedno poganjanje procesov. Pa tudi sicer lahko o sistemu zapišemo samo najboljše, saj poleg običajnih delov vsebuje še poseben urejevalnik, v katerem so definirane posebne tipke, ki omogočajo enostavno pisanje ukaznih struktur (zato tudi odpade prvi prehod) ter možnost uporabe knjižnice modulov, česar ne omogoča nobena druga od opisanih implementacij module-2. Razen tega SDS-XP odlikuje z izredno bogatimi orodji, ki so vključena v sistemski knjižnici ITCLIB.

S tem smo zaključili pregled nekaterih vidnih primerov prevajalnikov za modulo-2. Ta pregled pa ni popoln, saj nismo omenili sistemov za večje računalnike (VAX pod VMS) in podobno.

6. Modula-2 in prolog

Zadnjih nekaj vrstic tega pregleda namenimo zelo zanimivemu paketu za povezavo module-2 in prologa. Razvit je bil v Zurichu pod vodstvom Carla Mullerja [17, 18]. Povezava med obema programskima jezikoma je dvosmerna, saj lahko uporabljamo v prologu podprograme napisane v moduli-2, ali pa prologova načela logičnega iskanja po podatkovni bazi v moduli-2. Prva možnost je zelo enostavna. Definirati moramo le obliko podprograma vsebovanega v nekem modulu. Ta podprogram nam v postopku interpretacije prologovega programa predstavlja nek prologov stavek (klavzulo). Tako lahko resnično pričnemo razvijati nek program v prologu in postopoma stavek za stavkom prevajamo v modulo-2. Na koncu imamo celoten program v moduli-2, ga prevedemo in s tem pridobimo na hitrosti, ki je najbolj šibka točka prologovih interpreterjev.

Obraten, manj pogost primer uporabe, sloni na klicu prologovega interpreterja, ki je samo nek podprogram. Temu podprogramu predamo kot parameter nek niz, ki je v bistvu prologov stavek, katerega interpreter obdela na enak način, kot vsak drug prologov stavek. Opisani sistem je bil razvit na računalniku Lilith, sedaj pa je na volju tudi za računalnike IBM-PC pod MS-DOS in VAX pod VMS.

Enostavna prenosljivost paketa je prav plod uporabe OSSI standarda, ki ga paket uporablja za vse posege proti operacijskemu sistemu.

7. Lastne izkušnje

V Odseku za računalništvo in informatiko Inštituta Jožef Stefan že nekaj časa sistematično zbiramo dokumentacijo in orodja za programiranje v moduli-2. Do sedaj smo v moduli-2 realizirali nekaj manjših poskusnih projektov, od katerih jedra dveh navajamo v dodatku. Med drugim smo v moduli-2 razvili razporejevalnik sistemskega časa in s tem omogočili pisanje resničnih sočasno tekočih procesov na enem procesorju v nasprotju z definicijo module-2, ki pozna samo sorutine.

Naslednji korak pa je implementacija večih sočasno tekočih procesov na večih procesorjih. Projekt razvijamo v okviru razvoja poskusnega okolja PS-11 za programiranje vzporedno delujočih procesorjev na vodilu Q [3, 8]. Dosedanji rezultati so dokaj vzpodbudni: doslej

smo uspeli pognati več vzporednih, vendar zaenkrat nesodelujočih procesov, ki si delijo skupni pomnilnik.

Poleg tega nastaja v naših laboratorijah prva implementacija OSSI standarda za operacijski sistem RT-11 in MicroVMS.

7. Zaključek

Modula-2 se je v preteklih nekaj letih uveljavila kot jezik za sistemsko programiranje in kot jezik za pisanje velikih programskih sistemov. Obstaja ocena, da je v moduli-2 napisane že okoli 15% novejše sistemske programske opreme. Razlog za hitro razširjenje jezika je brez dvoma v njegovi enostavnosti in učinkovitosti. Prav gotovo si bo modula-2 s svojo preglednostjo in majhnostjo samega sistema, ter enostavnostjo rokovanja pridobila v prihodnje še veliko večjo veljavo in že morda kar kmalu nadomestila "stari dobri pascal".

Zahvala

Za vso pomoč pri iskanju novih znanj in dragocene nasvete pri oblikovanju besedila se najlepše zahvaljujem prof.dr.Boštjanu Vilfanu.

Literatura

[1] V.Batagelj, A.Brodnik in ostali: Proceedings of 4th Summer School of Computer Sciences, IJS Report 4352, Ljubljana, 1986.
[2] E.Biagioni, K.Hinrich, G.Heiser, C.Muller: A Portable Operating System interface and Utility Library, IEEE Software, November 1986.
[3] A.Brodnik, S.Mavrič, R.Trobec: Q-bus Base Multiprocessor System, CompEuro, Hamburg, 1987.
[4] S.Collins, B.Marshall, N.King: Modula-2 - A Babel Fish for Chips?, System International, December 1986.
[5] R.Gleaves: Modula-2 for Pascal Programmers, Springer-Verlag, 1984.
[6] K.Jensen, N.Wirth: PASCAL User Manual and Report, Springer-Verlag, 1975.
[7] B.W.Kernighan, D.M.Ritchie: The C Programming Language, Prentice-Hall, 1978.
[8] S.Mavrič, A.Brodnik, R.Trobec, M.Spegel, P.Kolbezen: PS-11: Večprocesorski sistem na vodilu Q, Mipro 1987, Opatija, 1987.
[9] Modula-2 Development System, Modula Corporation Provo, Utah, 1984.
[10] Modula-2 Installation Instructions, ETH Institut fur Informatik, Zurich, 1981.
[11] Modula-2 Overview of the Modula-2 Compiler, ETH Institut fur Informatik, Zurich, 1981.
[12] Modula-2 Overview of Modula-2 Debugger, ETH Institut fur Informatik, Zurich, 1981.
[13] Modula-2 Overview of Modula-2 System M2M, ETH Institut fur Informatik, Zurich, 1982.
[14] Modula-2 System for Z80 CP/M, Hochstrasser Computing AG, Zurich, Switzerland, 1984.
[15] Modula-2 User Guide, ETH Institut fur Informatik, Zurich, 1981.
[16] B.Mohar, E.Zakrajšek: Uvod v programiranje, DMFA, Ljubljana, 1982.
[17] C.Muller: Modula-Prolog (User Manual), ETH Institut fur Informatik, Zurich, Juli 1985.
[18] C.Muller: Modula--Prolog: A Software Development Tool, IEEE Software, November 1986.
[19] Osnove programskega jezika C, Delovni zvezek Izobraževalni center Delta, 1984.
[20] G.Pomberger: Software Engineering and Modula-2, Prentice-Hall, 1984.
[21] I.C.Pyle: The Ada Programming Language, Prentice-Hall, 1981.
[22] SDS-XP, Extended Performance Modula-2 Software Development System, Interface Technologies, Houston, Texas, 1985.
[23] R.Thomas, L.R.Rogers, J.L.Yates: Advanced Programmer's Guide to UNIX System V, McGraw-Hill, 1986.
[24] P.Wegner: Programming with Ada, Prentice-Hall, 1980.
[25] N.Wirth: Programming in Modula-2, Third, Corrected Edition, Springer-Verlag, 1985.

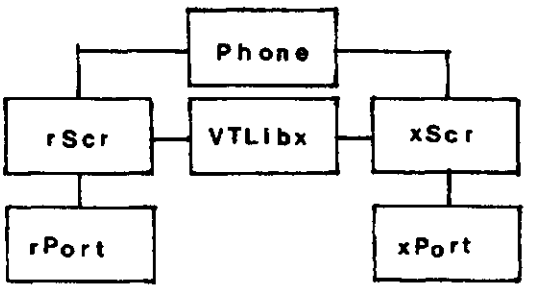
Dodatek A - Phone

Program Phone omogoča pogovor med dvema terminaloma priključenima na računalnik LSI-11 pod operacijskim sistemom RT-11. Ker je RT-11 enouporabniški sistem, je lahko hkrati aktiven le en terminal. Tu program poženemo in izberemo drugega. Pri obeh se na zaslonu pojavi obrazec. Na zgornjo polovico zaslona se izpisujejo znaki, sprejeti od drugega terminala, na spodnjo pa znaki, ki jih terminal pošilja drugemu.

Program sestoji iz šestih modulov:

- **Phone:** osnovni program
- **rScr:** modul za delo z enim zaslonom
- **xScr:** skrbi za drugi zaslon
- **VTLibx:** posebni ukazi za delo s terminali tipa VT100
- **rPort:** posluževanje prekinitev na enem vhodu
- **xPort:** posluževanje prekinitev na drugem vhodu

Na sliki 1 je še prikazana podrobnejša shema medsebojne odvisnosti posameznih modulov.



slika 1

Definicijski moduli:

```
DEFINITION MODULE rScr;
  (* modul za delo s prvim terminalom *)
  EXPORT QUALIFIED
    rPutCh, rPripraviEkran, rBrisi, rGetCh;

  PROCEDURE rPutCh (ch: CHAR; flag: BOOLEAN);
    (* izpiše znak na prvi zaslon *)
  PROCEDURE rPripraviEkran (VAR rVect,
                            rCSR: CARDINAL);
    (* pripravi zaslon terminala *)
  PROCEDURE rBrisi;
    (* zbriše zaslon terminala *)
  PROCEDURE rGetCh (VAR ch: CHAR;
                    VAR OK: BOOLEAN);
    (* pobere zlog s terminala *)
END rScr.

DEFINITION MODULE VTLibx;
  (* Ukazi za delo s terminali tipa VT 100 *)
  EXPORT QUALIFIED sCursor, sErase, sWriteLine,
                   sScroll, sNewLine, entire,
                   PutString, Obrazec, DelCh;

  CONST   entire = 2; (* zbriše ves zaslon *)
```

```
  TYPE    WriteProc = PROCEDURE (CHAR);
    (* podprogram za izpis zloga *)

  PROCEDURE sScroll (from, to: CARDINAL;
                     write: WriteProc);
    (* določi območje premikanja zaslona *)
  PROCEDURE sErase (area: CARDINAL;
                    write: WriteProc);
    (* zbriše določen del zaslona *)
  PROCEDURE sCursor (x, y: CARDINAL;
                     write: WriteProc);
    (* postavi slednik na določeno mesto *)
  PROCEDURE sNewLine (write: WriteProc);
    (* premakne slednik v novo vrsto *)
  PROCEDURE sWriteLine (long: CARDINAL;
                        write: WriteProc);
    (* izpiše črto zahtevane dolžine *)
  PROCEDURE PutString (what: ARRAY OF CHAR;
                       write: WriteProc);
    (* izpiše niz *)
  PROCEDURE Obrazec (write: WriteProc);
    (* izpiše osnovni zaslonski obrazec *)
  PROCEDURE DelCh (write: WriteProc);
    (* zbriše zadnji znak v tekoči vrstici *)
END VTLibx.

DEFINITION MODULE xPort;
  (* posluževanje prekinitev na kanalu *)
  EXPORT QUALIFIED
    xGetByt, xInit, xClose, xPutByt;

  PROCEDURE xGetByt (VAR ch: CHAR;
                     VAR OK: BOOLEAN);
    (* pobere zlog iz vmesnika,
       če je na voljo *)
  PROCEDURE xInit (VAR xVect, xCSR: CARDINAL);
    (* nastavi vse prekinitvene podprograme *)
  PROCEDURE xClose;
    (* zapre in vse pospravi okoli kanala *)
  PROCEDURE xPutByt (VAR ch: CHAR;
                     VAR OK: BOOLEAN);
    (* vstavi znak v vmesnik, če je prostor *)
END xPort.
```

V primeru modulov rScr in xPort smo navedli samo po en modul za vsako stran. Druga sta namreč skoraj povsem enaka. Sledi še podoben pregled programskega in delovnih modulov. Navajamo samo delovna modula, ki sta zanimivejša (izpuščamo VTLibx).

```
MODULE Phone;
  (* glavni programski modul *)
  FROM xScr IMPORT
    xGetCh, xPutCh, xPripraviEkran, xBrisi;
  FROM rScr IMPORT
    rGetCh, rPutCh, rPripraviEkran, rBrisi;
  FROM TTIO IMPORT
    WriteString, Read, Write, SetMode;

  CONST ETX = 003C;   (* <CTRL><C> *)
  VAR xch, rch: CHAR;
      OK, OK: BOOLEAN;
      CSR, Vect: CARDINAL;

  PROCEDURE BeriCSR (niz: ARRAY OF CHAR;
                     VAR Vect, CSR: CARDINAL);
  (* določi vektorja in naslove statusnih
     registrov obeh terminalov *)
  VAR Enota: CHAR;
  BEGIN
    WriteString (niz);
    Read (Enota); Write (Enota);
    CASE Enota OF
        '1': Vect := 310B; CSR := 1765108
      | '2': Vect := 350B; CSR := 1765508
    END
  END BeriCSR;

BEGIN
  SetMode (0, FALSE);  (* prepove <CTRL><C> *)
  BeriCSR ('    Od TT (st.): ', Vect, CSR);
  xPripraviEkran (Vect, CSR);
  BeriCSR ('    Proti TT (st.): ', Vect, CSR);
  rPripraviEkran (Vect, CSR);
  REPEAT (* glavna zanka za izmenjavo znakov *)
    xGetCh (xch, OK);
    IF OK THEN
      rPutCh (xch, TRUE); xPutCh (xch, FALSE)
    END;
    rGetCh (rch, OK);
    IF OK THEN
      xPutCh (rch, TRUE); rPutCh (rch, FALSE)
    END
  UNTIL (xch = ETX) OR (rch = ETX);
  xBrisi; rBrisi
END Phone.

IMPLEMENTATION MODULE rScr;
  (* delo s sprejemnim zaslonom *)
  FROM rPort IMPORT
    rInit, rGetByt, rClose, rPutByt;
  FROM VTLibx IMPORT
    sCursor, sErase, sWriteLine, sScroll,
    sNewLine, entire, PutString, DelCh,
    Obrazec;

  CONST CR = 015C; LF = 012C; DEL = 177C;
        BS = 010C; REFR = 027C; ESC = 033C;

  VAR xm, xt, ym, yt, x, y: CARDINAL;

  PROCEDURE Send (ch: CHAR);
    (* pošlje znak na zaslon *)
  VAR ok: BOOLEAN;
  BEGIN
    REPEAT rPutByt (ch, ok) UNTIL ok
  END Send;

  PROCEDURE rPripraviEkran (VAR rVect,
                                rCSR: CARDINAL);
    (* nastavi terminal in izpiše obrazec *)
  BEGIN
    rInit (rVect, rCSR);
    xm := 1; xt := 1; ym := 11; yt := 22;
    x := xm; y := ym;
    Obrazec (Send)
  END rPripraviEkran;

  PROCEDURE rPutCh (ch: CHAR; flag: BOOLEAN);
    (* interpretira sprejeti znak *)
    PROCEDURE Uredi (VAR a, b: CARDINAL);
    VAR i: CARDINAL;
    BEGIN
      IF ch = REFR THEN
        FOR i:=1 TO 10 DO sNewLine (Send) END;
        a := 1
      END;
      IF ch = DEL THEN DelCh (Send); a:=a-1
      ELSIF ch = CR THEN sNewLine (Send); a:=1
      ELSE Send (ch); a := a + 1 END;
    END Uredi;
  BEGIN
    IF (flag) THEN
      IF x # xm THEN
        sScroll (3, 11, Send);
        sCursor (xm, ym, Send);
        x := xm
      END;
      Uredi (xm, ym)
    ELSE
      IF x # xt THEN
        sScroll (14, 22, Send);
        sCursor (xt, yt, Send);
        x := xt
      END;
      Uredi (xt, yt)
    END;
  END rPutCh;

  PROCEDURE rGetCh (VAR ch: CHAR;
                    VAR OK: BOOLEAN);
    (* dobi znak s terminala *)
  BEGIN
    rGetByt (ch, OK);
  END rGetCh;

  PROCEDURE rBrisi;
    (* zbriše ekran in zaključi komunikacijo *)
```

```
  BEGIN
    sErase (entire, Send); rClose
  END rBrisi;
END rScr.


IMPLEMENTATION MODULE xPort [7];
  (* podprogrami za delo z enim od kanalov *)

  FROM SYSTEM IMPORT
    ADR, ADDRESS, SIZE, WORD,
    NEWPROCESS, TRANSFER, IOTRANSFER;

  CONST N = 32;

  TYPE DLV11J  = RECORD
                   rCSR: BITSET;
                   rBUF: CARDINAL;
                   xCSR: BITSET;
                   xBUF: CARDINAL
                 END;

       TypeCSR = POINTER TO DLV11J;

  VAR rn, rin, rout, xn, xin, xout: CARDINAL;
      rbuf, xbuf: ARRAY [0..N-1] OF CHAR;
      rPRO, xPRO, CON: ADDRESS;
      rWSp, xWSp: ARRAY [0..177B] OF WORD;

      px1, px2, px3, px4: WORD;

      active: BOOLEAN;
      Vector: ADDRESS;
      CSR: TypeCSR;

  PROCEDURE xGetByt (VAR ch: CHAR;
                     VAR OK: BOOLEAN);
    (* pobere zlog iz sprejemnega vmesnika,
       če lahko *)
  BEGIN
    IF rn > 0 THEN
      OK := TRUE;
      ch := rbuf [rout]; rn := rn - 1;
      rout := (rout+1) MOD N;
    ELSE OK := FALSE
    END
  END xGetByt;

  PROCEDURE producer;
    (* sprejemni prekinitveni podprogram *)
  BEGIN
    LOOP
      IOTRANSFER (rPRO, CON, Vector);
      IF rn < N THEN
        rbuf [rin] := CHR (CSR^.rBUF);
        rin := (rin+1) MOD N;
        rn := rn + 1
      END
    END
  END producer;

  PROCEDURE xPutByt (VAR ch: CHAR;
                     VAR OK: BOOLEAN);
    (* vstavi zlog v oddajni vmesnik,
       če lahko *)
  BEGIN
    IF xn < N THEN
      OK := TRUE;
      xbuf [xin] := ch; xn := xn + 1;
      xin := (xin+1) MOD N;
      IF NOT active THEN
        INCL (CSR^.xCSR, 6); active := TRUE
      END
    ELSE OK := FALSE;
    END
  END xPutByt;

  PROCEDURE consumer;
    (* oddajni prekinitveni podprogram *)
  BEGIN
    LOOP
      IOTRANSFER (xPRO, CON,
                  ADDRESS (Vector+4));
      IF xn > 0 THEN
        CSR^.xBUF := ORD (xbuf [xout]);
        xn := xn - 1; xout := (xout+1) MOD N
      ELSE
        EXCL (CSR^.xCSR, 6); active := FALSE
      END
    END
  END consumer;

  PROCEDURE xInit (VAR xVect, xCSR: CARDINAL);
    (* inicializiramo registre, ki jih bomo
       potrebovali in poženemo nov proces *)
  VAR a: ADDRESS;
  BEGIN
    Vector := xVect; CSR := TypeCSR (xCSR);

      (* shranimo stare vrednosti *)
    a := ADDRESS (Vector);
    px1 := a^; INC (a, 2);
    px2 := a^; INC (a, 2);
    px3 := a^; INC (a, 2);
    px4 := a^;

    rn := 0; rin := 0; rout := 0;
    NEWPROCESS (producer, ADR(rWSp),
                SIZE(rWSp), rPRO);

    xn:=0; xin:=0; xout:=0; active:=FALSE;
    NEWPROCESS (consumer, ADR(xWSp),
                SIZE(xWSp), xPRO);

    WITH CSR^ DO
      INCL (rCSR, 6); TRANSFER (CON, rPRO);
      TRANSFER (CON, xPRO)
    END
  END xInit;

  PROCEDURE xClose;
    (* počistimo vse za seboj *)
  VAR a: ADDRESS;
  BEGIN
      (* obnovimo stare vrednosti *)
    a := ADDRESS (Vector);
    a^ := px1; INC (a, 2);
    a^ := px2; INC (a, 2);
    a^ := px3; INC (a, 2);
    a^ := px4
  END xClose;
END xPort.
```

Na sliki 2 pa podajamo še prikaz medsebojnih klicev modulov. Primer je tudi lep način razbitja na module in definicije njihovih stičnih točk - podprogramov.

Dodatek B - Razporejevalnik

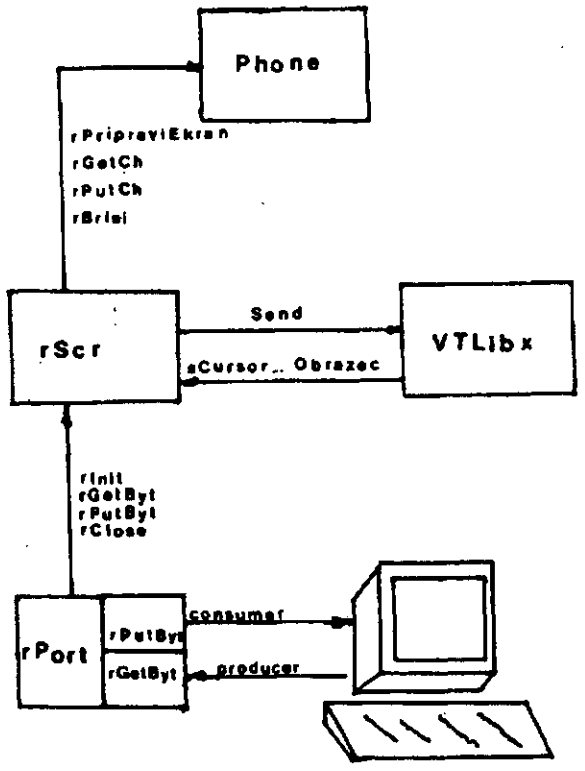
Tukaj navajamo samo primer definicijskega in delovnega modula razporejevlanika procesorskega časa in kratek primer z dvema procesoma.

```
DEFINITION MODULE Scheduler;
  EXPORT QUALIFIED ProcessGo;
  PROCEDURE ProcessGo (which: PROC;
                       space: CARDINAL);
    (* pozene nov proces, ki uporablja space
       velik prostor v pomnilniku *)
END Scheduler.

IMPLEMENTATION MODULE Scheduler[7];

  FROM SYSTEM IMPORT
    ADDRESS, PROCESS,
    NEWPROCESS, TRANSFER, IOTRANSFER;
  FROM Storage IMPORT ALLOCATE;

  CONST N = 10;
        (* najvecje stevilo procesov *)
        ClockVec = 100B;
          (* prekinitveni vektor ure *)
        TimerSpace = 200B;
          (* prostor za prekinitveni
```



slika 2

```
          podprogram za uro *)
      TicksPerPeriod = 50
        (* stevilo urinih enot, ki so
           dodeljeni enemu procesu *)

VAR StartNewProcess: BOOLEAN;
    wsp: ADDRESS;
    FirstProc,          (* starting process *)
    TimerAdd: PROCESS;(* here is timer *)
    Processes: ARRAY [1..N] OF PROCESS;
    n, current: CARDINAL;

PROCEDURE Timer;
  (* dusa razporejevlnika *)
VAR NumberOfTicks: CARDINAL;
BEGIN
  NumberOfTicks := TicksPerPeriod;
  LOOP
    IF n > 0 THEN
      IF StartNewProcess THEN
        StartNewProcess := FALSE;
        NumberOfTicks := TicksPerPeriod;
        current:=n
      ELSIF NumberOfTicks <= 0 THEN
        NumberOfTicks := TicksPerPeriod;
        current:=(current MOD n)+1
      ELSE
        NumberOfTicks := NumberOfTicks - 1
      END;
      IOTRANSFER (TimerAdd,
                  Processes[current],
                  ClockVec)
    ELSE
      NumberOfTicks := TicksPerPeriod;
      IOTRANSFER (TimerAdd, FirstProc,
                  ClockVec)
    END;
  END
END Timer;

PROCEDURE ProcessGo (which: PROC;
                     space: CARDINAL);
  (* pozene nov proces *)
VAR wsp: ADDRESS;
    OldCur: CARDINAL;
BEGIN
  IF n < N THEN
    ALLOCATE (wsp, space);
      (* poiscemo prostor za novi proces *)
    n:=n+1;
    NEWPROCESS (which, wsp, space,
                Processes[n]);
    StartNewProcess := TRUE;
    IF n = 1 THEN          (* ali je prvi *)
      TRANSFER (FirstProc, TimerAdd)
    ELSE          (* sicer ga pozene Timer *)
      TRANSFER (Processes[current],
                TimerAdd)
    END;
  END
END ProcessGo;

BEGIN
  n:=0; current:=1; StartNewProcess := FALSE;
  ALLOCATE (wsp, TimerSpace);
  NEWPROCESS (Timer, wsp, TimerSpace,
              TimerAdd);
  TRANSFER (FirstProc, TimerAdd)
    (* in pozeneo Timer *)
END Scheduler.

MODULE TestScheduler;

  FROM Scheduler IMPORT ProcessGo;
  FROM VTLibx IMPORT sCursor;
  FROM TTIO IMPORT Write, Read;

  CONST SpaceNeeded = 4008;
  VAR ch: CHAR;

  PROCEDURE One;
    (* na zaslon izpisuje "A" *)
  VAR i: CARDINAL;
  BEGIN
    ProcessGo (Two, SpaceNeeded); i:=1;
    LOOP
      sCursor (i, 4, Write); i:=(i MOD 80)+1;
      Write ('A')
    END
  END One;

  PROCEDURE Two;
    (* ta pa naj izpisuje "B" *)
  VAR i: CARDINAL;
  BEGIN
    i:=1;
    LOOP
      sCursor (i, 5, Write); i:=(i MOD 80)+1;
      Write ('B')
    END
  END Two;

BEGIN
  ProcessGo (One, SpaceNeeded)
END TestScheduler.
```

IMPLIKACIJA ORGANIZACIJE NERVNIH SISTEMA NA RAČUNARSKE SISTEME

UDK 681.327.2

Marković Pavle
Visoke vojne tehničke škole, Zagreb

Strukturna i funkcionalna organizacija i nacin interpretacije dolazecih informacija nervnih sistema su medju najslozenijim strukturama koje je priroda stvorila, i jos uvek su daleko od moguce tehnicke emulacije. Osnovna nacela te organizacije su od skoro poznata i iskoriscena su u ovom radu u implikaciji na racunarske sisteme. U radu su na osnovu izvrsene analize organizacije najviseg stadijuma nervnih sistema (mozga) postavljeni globalni zahtevi koje neki racunarski sistem mora ispuniti da bi mogao emulirati analiziranu nervnu organizaciju.

Kljucne reci: biloski sitemi, upravljacki sistemi, obrada informacija

IMPLICATION OF NERVOUS SYSTEMS ON COMPUTER ORGANIZATION. Structural and funcional organization and the manner of interpretation of incoming information of nervous systems are the most complicated structures in nature and they are far away of possible technical emulation. Basic principles are known and used in this paper in implication on computer systems. After analyzing the higest level of nervous system (human brain) the global requisments which same computer have to have in order to emulate the analyzed nervous organization are given.

Keywords: biological systems, control systems, information procesing

1. U V O D

Brzina racunara se od njihove pojave do danas visestruko povecala. Ipak, danasnji napori i pokusaji znacajnijeg povecanja brzine racunanja su tesko ostvarivi i veoma skupi. Uglavnom oni su usmereni u sledecim pravcima:

- tehnolosko povecanje brzine rada elektricnih komponenti
- promene u arhitekturi racunarskih sistema.

Tehnolosko povecanje brzine rada elektricnih komponenti nema vise onakav ostar gradijent porasta kao pre, tako da se glavna povecanja brzine racunanja ocekuju u promenama arhitektura racunarskih sistema. Iako, koncepti multiprocesorskih, multiracunaskih, distribuiranih, hijerarhijskih i drugih sistema su odavno poznati, njihova primena ne daje ona povecanja brzine racunanja koja su se ocekivala. U pomenutim organizacijama jos uvek kljune teskoce njihove iskoristivosti predstavljaju:

- problemi dekompozicije posla na grupe zavisnih i nezavisnih zadataka
- problemi komunikacije u sistemi
- pristup upravljanju (koordinaciji) u sistemu

Od gornja tri problema centralno mesto zauzima problem pristupa upravljanju u sistemu. Svi dosadasnji pristupi mogu se svrstati u jednu od sledece tri kategorije:

- centralizovani pristup
- samoupravni pristup
- decentralizovani pristup

U centralizovanom pristupu sve informacije o stanju svakog objekta upravljanja i vanjskim delovanjima dolaze u centralno mesto upravljanja. Tu se na osnovu informacija o stanju sistema i buducim zadacima odredjuju upravljacke akcije za svaki element sistema. Ovakva struktura se moze uciniti idealnom za pristup upravljanju, all ona u sebi nosi i dosta skrivenih poteskoca. Jedna od njih je i prevelika kolicina informacija koje dolaze u centralno mesto upravljanja. To automatski znaci tesko odredjivanje najpovoljnijeg rezima rada u realnom vremenu, upravo zbog preobilja informacija.

Za razliku od centralizovanog pristupa, samoupravni podrazumeva skoro potpunu autonomiju odlucivanja svakog elementa sistema. Pri tome svaki element sistema samostalno resava vlastite lokalne zadatke, dok u saradnji sa drugim elementima sistema resava zadatke od zajednickog interesa. Ostvarenje sistema u kome bi svaki nt imao totalnu autonomiju je prakticno neizvodljivo jer bi tada bila nemoguca uskladjenost rada citavog sistema u celini. Zato ovaj pristup upravljanju nosi u sebi obelezja i prvog i treceg pristupa.

Decentarailzovani pristut podrazumeva rascanjivanje sistema na delove (hijerarhijski sistemi) izmedju kojih se uspostavlja koordinacija. Tada upravljacki element viseg nivoa upravlja elementima nizeg i sve tako do najnizeg. Hijerarhijska struktura, slicno centralizovanoj, podrazumeva izdvajanje upravljackih elemenata sistema viseg nivoa od upravljackih elemenata nizeg nivoa, a funkcionalna podela rada smislena je samo na istom nivou, kao razdvajanje razlicitih podzadataka istog nivoa.

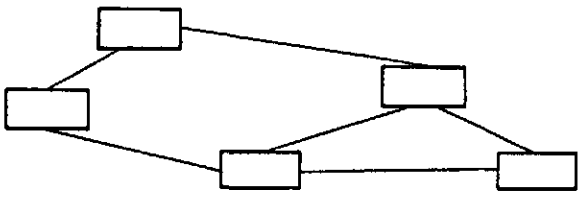
Realizacija bilo kog navedenog pristupa upravljanju u multiprocesorskim sistemima opste namene pokazala se nedovoljno efikasnom jer je unosila dodatna ogranicenja u primeni i upravljanju resursima. Zbog toga se krenulo u analizu onih bioloskih sistema za koje se sigurno moze tvrditi da spadaju u klasu multiprocesorskih sistema (nervni sistemi). Autor smatra da su ti sistemi najsiozenije strukture koje je priroda stvorila (mozda cak i optimalne) i da je danasnja nauka i tehnologija jos uvek jako daleko od njihove moguce implementacije. Na osnovu takve analize trebali su proisteci osnovni zahtevi koje neki racunarski sistem mora ispuniti kako bi iskoristio pristup upravljanju koji se nalazi ugradjen u nervnim sistemima. U analizu nervnih sistema posio se od njihovih elementarnih oblika da bi se zavrsila sa analizom najsiozenijih nervnih formacija tj. mozga coveka.

Analiza nervnih sistema izvrsena je sa stanovista njihove funkcionalne organizacije i nacina interpretacije ulaznih informacija. Ovako trazenom pristupu najbolje je odgovarao pristup ruskog akademika i osnivaca Neuropsihologije A.R.Lurije, a koji je on dao u knjizi 'Osnovi neuropsihologije', (prevod na srpskohrvatskom izdala je izdavacka kuca Nolit u biblioteci Sazvezdja). Autor nije imao nameru da prepricava pomenutu knjigu vec su sve diskusije vezane za organizacije nervnih sistema, a koje se nalaze u ovom radu, bili prepisane ili samo prilagodjene radu (zbog duzine tekstualnih objasnjenja).

Da se prilikom citanja teksta ne bi doslo u koliziju vazno je objasniti tumacenje pojma "nivo hijerarhije", u organizaciji covekovog mozga. Zbog tumacenja nivoa hijerarhije u pomenutoj knjizi uzeto je sledece znacenje: stari delovi mozga i mozdano stablo se nazivaju nizim tako da retikularna formacija predstavlja najnizi nivo hijerarhije ljudskog mozga a aparati nove kore najvisi. U istom smislu tumaci se znacenje nivoa u uporednoj implikaciji na organizaciju racunarskog sistema.

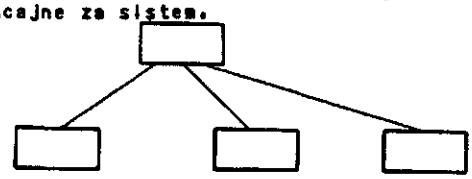
2. RAZVOJ NERVNIH SISTEMA

U evoluciji je poznato da se na najelementarnijim nivoima razvoja zivotinjskog sveta prijem signala i organizacija kretanja ostvaruje difuznim ili mrezastim nervnim sistemom. Na toj etapi evolucije ne postoji jedinstveni centar koji prima i obradjuje informacije, i protok drazi se odredjuje onim privremenim dominantnim centrima koji se stvaraju u ovom ili onom delu nervnog aparata. Uporedjujuci takav nervni aparat s racunarskim sistemima dolazi se do zakljucka da bi se taj sistem sastojao od vise 'izolovanih' racunara koji svaki radi svoj posao, a u pojedinim momentima oni medjusobno razmenjuju poruke koje bi govorile o ukupnom stanju sistema ili podsistema. Poruke su kratke i vezane samo za ukupno stanje sistema. Na danasnjem nivou racunarske tehnike ovom modelu najblize su distribuirane racunarske mreze (slika 1). U toku dalje evolucije dolazi do razvitka prednje ganglije koja polako pocinje da preuzima slozenije funkcije. Ona prima i preradjuje informacije sa receptora i prebacuje nadrazaje na silazne puteve koji se krecu ka motornim aparatima zivotinja.



Slika 1. Distribuirana računarska mreža

U ranim stadijima evolucije (crvi) prednja ganglija je imala jednostavnu funkcionalnu strukturu dok u kasnijim (insekti) dobija sve slozeniju funkcionalnu organizaciju (izdvajaju se neuroni koji reaguju samo na pojedine vrste drazi (vidne, mirisne, taktilne)). Gledano sa strane racunarske tehnike ovakvim nervnim sistemima odgovaraju centralizovane mreze (slika 2), gde centar pocinje da preuzima pojedine funkcije znacajne za sistem.



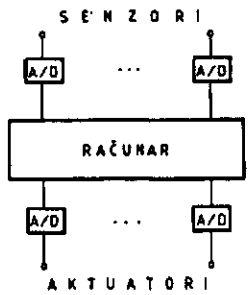
Slika 2. Centralizovana računarska mreža

Visim stadijumima ovih nervnih aparata odgovaraju takve strukture gde se centralnom racunaru predodeljuje sve vise funkcija tj. on pocinje da biva sve znacajniji i uticajniji na ukupni sistem.

U toku dalje evolucije nervni aparati prednje ganglije ne mogu da obezbede prilagodjavanje naglo promenjenim uslovima sredine. U takvim slucajevima ocuvanje vrste moguce je ili zahvaljujuci preobilnoj proizvodnji individua ili zahvaljujuci razradi individualno-promenljivog ponasanja.

Po ovoj drugoj liniji se odvija razvitak kicmenjaka. Ako kod nizih kicmenjaka vazi stari princip onda se kod kicmenjaka koji prelaze na egzistenciju na kopnu pojavljuje potreba za takvim nervnim aparatom koji ce obezbedjivati maksimalnu individualnu promenljivost ponasanja koja pak odgovara velikoj promenljivosti uslova zivota na zemlji.

Takvim bioloskim zadacima odgovara mozak. Na ovom nivou razvoja nervnih aparata moze se slobodno izjaviti da postoji jedan jedinstveni centar koji ne pruza nikakvu ili malu autonomiju rada. To vodi do zakljucka da postoji (racunarski gledano) jedan centralni sistem koji sve oko sebe kontrolise. Nema nikakvih distribuiranih mreza u kojima se informacije prenose s jednog na drugi kraj, vec sve informacije sa senzora dolaze na centralno mesto, tu se obradjuju i reakcije prenose na aktuatore (slika 3).



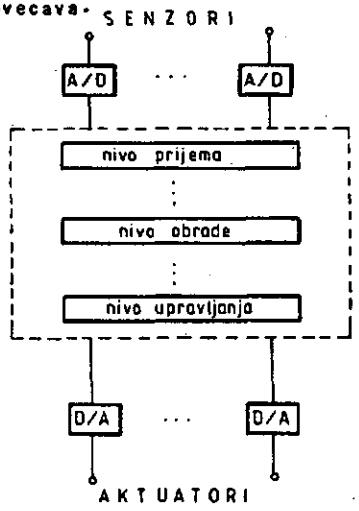
Slika 3. Računarski upravljan proces

Dolazi se do zakljucka da je to konacno resenje s jednim centralnim racunarskim sistemom koji obavlja sve funkcije. Medjutim, postavlja se problem velicine tog racunarskog sistema (propusna moc) koji bi kroz sve etape evolucije sisara zadovoljio sve zahteve koji se pred njim postavljaju. Ocigledno je da, gledajuci evoluciju od najprostijih sisara do coveka, da bi taj racunarski sistem verovatno rastao eksponencijalno (propusna moc) ili po nekoj slicnoj funkciji, obzirom na broj i slozenost funkcija koje bi on kroz uporedni evolucioni razvoj mozga morao da zadovolji. Ocigledna kvantitativnim poredjenjem se vidi da covek nema toliko proporcionalno veci mozak od nizih sisara, koliki bi po broju funkcija i slozenosci operacija trebalo ocekivati. Sama gornja cinjenica navodi na zakljucak da resenje problema lezi u unutrasnjoj organizaciji tog nazovimo ga racunarskog sistema.

2.1 Funkcionalna organizacija mozdanog aparata

Kao sto je poznato aparati mozga su u potpunosti sposobni da primaju i analiziraju informacije koje dolaze iz okoline sredine, preradjuju ih, formiraju nove veze i cuvaju njihove tragove. Oni su u stanju da urodjene programe ponasanja zamene slozenim, individualno-promenljivim, obezbedjujuci pri tome ne samo razradu uslovnih refleksa vec i formiranje mnogo slozenijih programa ponasanja.

Velike hemisfere mozga i mozdana kora postaju kod coveka najvazniji aparat za regulisanje ponasanja. Sa prelaskom od visih sisara ka coveku evolucija mozga povezana je sa povecanjem povrsine najslozenijih (tercijarnih) rnih) zona kore, dok povrsina elementarnih delova kore (primarnih i sekundarnih) prakticno se ne povecava.



Slika 5. Višenivovska organizacija računarskog sistema

Bilo bi nepravilno smatrati, da ako u coveka vodecu ulogu dobija mozdana kora, da se onda sve nervne formacije koje su na nizim nivoima evolucije predstavljale jedine aparate ponasanja sada u potunosti odstranjeni od rada. Ti prijasnji nervni aparati ostaju u njemu, ustupajuci pri tome vodece mesto novim formacijama i dobijaju novu ulogu. Oni sve vise postaju aparati koji obezbedjuju osnovu ponasanja sto aktivno ucestvuje u regulisanju stanja u kome se nalazi organizam, predajuci tako funkcije prijema, obrade i cuvanja informacija, a isto tako i funkciju stvaranja novih programa ponasanja, regulisanja i kontrole svesne delatnosti visim aparatima mozdane kore. Sve navedeno, gledana s racunarskog stanovista vodi do zakljucka da bi racunarski sistem koji bi koristio osnovne principe rada mozga morao biti realizovan kao neka visenivoska organizacija (slika 5). Sama veza izmedju nivoa nije definisana ali bi ona morala zadovoljiti osnovne principe evolucije, a to je da svaki nizi ili visi nivo mora predstavljati integralni deo ukupnog sistema s odeljenim funkcijama po nivoima, a opet u cilju sveukupnog delovanja celog sistema. To znaci da taj racunarski sistem po svojoj strukturi mora biti jedan dinamicki sistem koji mora obezbediti neku promenjivost (prilagodljivost) funkcija po nivoima. Nizi nivoi bi preuzimali aktivno ucesce u organizaciji i pripremi aktivnosti za vise nivoe.

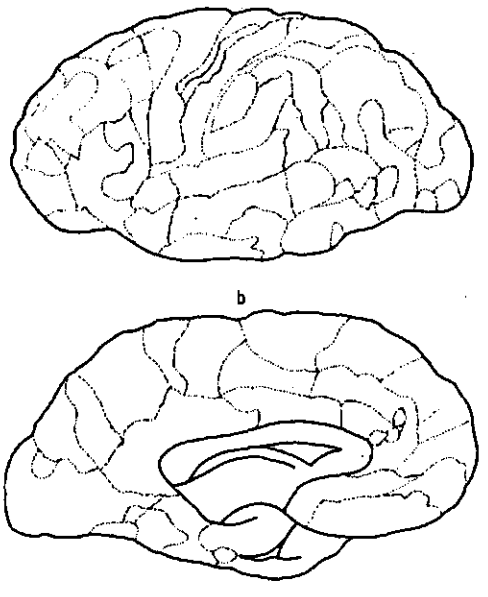
Iz fiziologije i neurologije se zna da najjednostavnije elemente ponasanja kao sto su segmentarni refleksi ostvaruju samo mehanizmi kicmene mozdine. Gledano tehnicki, refleksi kicmene mozdine predstavljaju zastitu pojedinih organa za koje se uspostavlja refleksni kruc. To je slicno kao kad se u krugu za regulaciju brzine tiristorski upravljanog motora postavi osigurac za prekostrujnu zastitu s tim da je, grubo gledano, ovde refleks izbacivanje osiguraca.

Takodje se zna da najslozenija urodjena forma ponasanja kao sto je regulisanje ravnoteze razmene materija ,koju obezbedjuje disenje, hranjenje i termoregulacija , ostvaruje posredstvom mehanizama koji se nalaze u gornjim delovima mozdanog aparata (produzenoj mozdini, hipotalamusu). Jos slozenije forme ponasanja koje zahtevaju obezbedjivanje tonusa i koordinacije tesno su povezane sa radom malog mozga i subkortikalnih motornih centara. Konacno, najslozenije forme delatnosti ne mogu biti obezbedjene bez ucesca mozdane kore, koja predstavlja organ najvisih formi ponasanja zivotinja i svesnog ponasanja coveka. Iz ovih cinjenica se vidi da uloga nizih nivoa mozdanog aparata ima odlucujucu ulogu u onim funkcijama koje su najznacajnije za celi organizam (odrzanje organizma). Ujedno te funkcije po svom karakteru i psihickim delatnostima su nize. Kad bi se to pokusalo preslikati na neki racunarski sistem onda bi nizi nivoi preuzimali funkcije koje su po svojoj kompleksnosti nize ali znacajnije za ukupnu organizaciju i funkciju sistema, dok bi visi nivoi vrsili detaljnu obradu informacija i preduzimali potrebne akcije koje ona iziskuje.

2.2 Strukturna i funkcionalna organizacija mozdane kore

Posmatranja koja su pokazala da mozak u celosti, a delimicno i njegova kora nisu jednorodne strukture datiraju iz pocetka prosloq veka.

Kako su pokazala kasnija istrazivanja nova mozdana kora se sastoji od sest slojeva celija. Samo donji od njih predstavljaju aparate koji neposredno povezuju mozdanu koru sa periferijom: organima cula (IV- aferentni sloj) i misicima (V - eferentni sloj). Ovi slojevi se nazivaju primarne zone kore.



Slika 6. Gruba topografska karta moždanih zona
a) spoljna površina
b) unutrašnja površina

Kako su pokazala istrazivanja nad svakom 'primarnom', zonom kore nadogradjuje se sistem 'sekundarnih', zona u kojima dominantno mesto zauzimaju slozeniji po svojoj strukturi II i III sloj. Ti slojevi sastoje se od celija sa kratkim aksonima od kojih najveci deo ili nema veze sa periferijom ili dobija svoje impulse iz subkortikalnih formacija, koje se nalaze u dubini mozga, a koje ostvaruju prvu preradu

informacija sto dolaze sa periferije. Struktura tih slojeva omogucuje da se svrstaju u slozeniji asosicajativni ili integralni aparat mozdane kore.

Hijerarhijska struktura mozdane kore iako se moze videti ako se pogleda topografska karta mozga (slika 6). Kao sto se na karti moze videti, u mozdanoj kori coveka se mogu izdvojiti oblasti koje leze na granicama izmedju kortikalnih predstavnistva pojedinih osecajnih zona mozdane kore, a koje su dobile naziv tercijerne zone kore (ili zone prepokrivanja kortikalnih delova pojedinih analizatora). Te oblasti kore se sastoje u celosti od gornjih slojeva celija i one nemaju direktne veze sa periferijom. Postoje sve osnove za predpostavku da tercijerne zone kore obezbedjuju zajednicki rad grupa analizatora, tj. najslozeniju funkciju mozga.

Sve do sada izneseno vodi do zakljucka da racunarski model koji bi koristio osnovne principe rada mozga mora biti hijerarhijska struktura u vise nivoa. O samoj hijerahijskoj strukturi bit ce detaljno govoreno kasnije ali ono sto je vazno napomenuti je to da nizi nivoi hijerahije sa stanovista ukupnog sistema moraju izvrsavati znacajnije funkcije. Ovo proizlazi iz saznanja da nizi nivoi mozga kontrolisu najznacajnije funkcije organizma, kao sto su disanje, puls, lucenje zlezda itd. Oni ne izvrsavaju vise psihicke delatnosti, ali igraju odlucujucu ulogu u funkcijama odrzavanja organizma sto je sa stanovista ukupnog sistema najznacajnije funkcija.

Kako su pokazala istrazivanja, dete dolazi na svet sa potpuno oformljenim aparatima subkortikalnih formacija i najjednostavnijim, projekcionim ili primarnim zonama kore i sa nedovoljno oformljenim aparatima slozenijih sekundarnih i tercijalnih zona kore. Najburnije sazrevanje sekundarnih delova kore zapaza se u uzrastu od 2-3 godine zivota dok najslozenije oblasti sazrevaju konacno u uzrastu od 6-7 godina. Tercijerni delovi kore sazrevaju u periodu do 3-3,5 godina zivota dok najslozenije oblasti produzavaju svoj razvitak do 12 godine zivota. Ova hijerahijska struktura mozdane organizacije i same kore govori da su najznacajnije funkcije po organizam (za njegovo preziviavanje u sredini) kod coveka ugradjene od samog rodjenja, dok sve vise forme psihicke delatnosti se razvijaju kroz razvoj sekundarnih i tercijalnih zona kore a samim tim i visih nivoa mozdane hijerarhije. To znaci da nizi nivoi hijerarhije pored svojih unapred definisanih funkcija imaju i dodatnu funkciju 'stimulisanja', i odredjivanja razvoja visih nivoa jer njihove funkcije se tek kasnije u ontogenezi razvijaju zajedno s razvojem sekundarnih zona kore. Ako se povuce paralela na model racunarskog sistema to znaci da nizi nivoi hijerarhije bi morali da pored svojih normalnih (nizih) funkcija i da aktivno ucestvuju u funkcijama visih pa cak i da te funkcije odredjuju.

James Albus u svojoj knjizi 'Brains, Behaviors & Robotics', je dao hijerahijski model mozga (slika 7.1 7.2) gde nizi nivoi ekstrahiraju informacije za vise nivoe. Taj model bi vazio kad bi se covek radjao s potpuno formiranim sekundarnim i tercijarnim zonama kore, jer onda bi nizi nivoi hijerarhije izlucivali informacije za vise. Ali kako to izlucivanje izvrsiti kad visi nivoi hijerarhije nisu formirani do kraja i kad nizi nivoi uticu na formiranje visih?

2.3 Tri osnovna funkcionalna bloka mozga

Mogu se izdvojiti tri osnovna funkcionalna bloka ili tri osnovna mozdana aparata cije je ucesce neophodno za ostvarivanje bilo kakvog oblika psihicke delatnosti. Ti osnovni blokovi su: 1. blok regulisanja tonusa ili stanja budnosti, 2. blok prijema, obrade i cuvanja informacija, 3. blok programiranja, regulisanja i kontrole psihicke delatnosti.

Svaki od tih osnovnih blokova poseduje hijerarhijsku strukturu i sastoji se od najmanje tri nadgradjene zone tipa: primarne, u koju stizu impulsi sa periferije ili odakle se upucuju impulsi prema periferiji; sekundarne, u kojoj se obavlja obrada primljene informacije ili se pripremaju odgovarajuci programi; tercijarne, koja predstavlja aparate velikih hemisfera koji obezbedjuju najslozenije forme psihicke delatnosti. Da bi racunarski model opstao morao bi zadovoljiti upravo predlozenu hijerarsijsku strukturu koja bi na neki nacin oponasala upravo pomenuta tri funkcionalna bloka mozga. Taj racunarsi sistem bi morao imati adekvatne komponente pojedinim funkcionalnim blokovima i to:

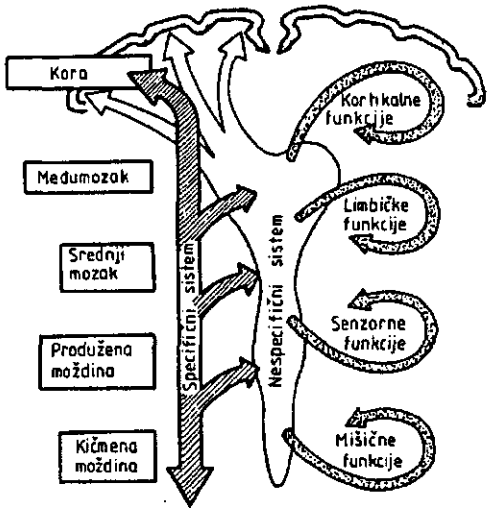
- Aktivirajucu (uzbudnu) komponentu
- Obradjujucu komponentu
- Programirajucu komponentu

2.3.1 Blok regulisanja tonusa i stanja budnosti

Jedno od najvaznijih otkrica za mesto bloka koji regulise tonus kore je otkrivanje cinjenice da se ti aparati ne moraju nalaziti u samoj kori, vec u delovima mozga koji se nalaze u delovima mozdanog stabla i subkortikalnim formacijama, i da se ti aparati nalaze u dvojnom odnosu prema kori podizuci njen tonus a istovremeno i osecaju na sebi njen regulacioni uticaj.

Godine 1949 dva istaknuta naucnika, Megun i Moruci, otkrili su da se u delovima mozdanog stabla nalazi posebna nervna formacija, koja je po svojoj strukturi i po svojim funkcionalnim svojstvima prilagodjena zadatku da ostvaruje ulogu mehanizama koji regulisu stanja mozdane kore tj. ona je u stanju da menja njen tonus i obezbedjuje stanje budnosti.

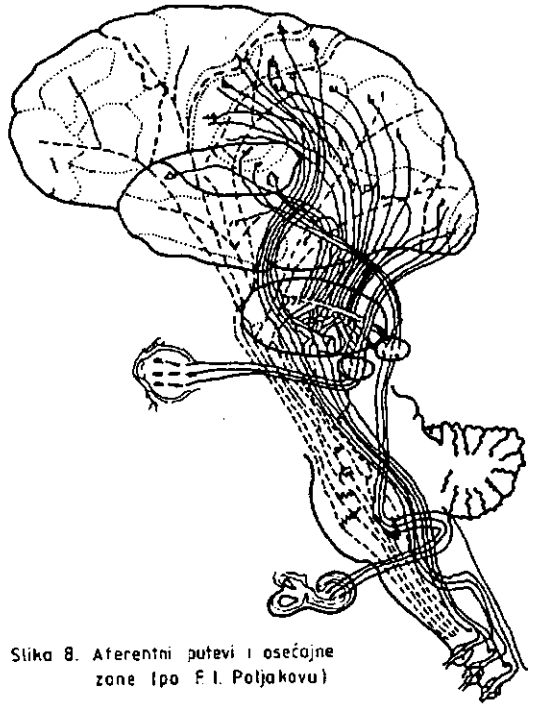
Ta formacija je sagradjena po tipu nervne mreze u koju su ubacena tela nervnih celija. Po mrezi te formacije koja je nazvana RETIKULARNA FORMACIJA, nadrazaj se siri ne pojedinim izolovanim impulsima, ne po zakonu sve ili nista, vec postepeno menjajuci svoj nivo i modulirajuci stanje celokupnog nervnog aparata (slika 7).



Slika 7. Shema aktivirajuće retikularne formacije

Jedan deo vlakana retikularne formacije ide nagore zavrsavajuci se u visim nervnim formacijama i na kraju u formacijama nove kore cija je uloga u organizaciji slozene psihicke delatnosti. Te formacije su dobile naziv uzlazni retikularni sistem. Samo otkrice da se u evolucione najstarijim delovima mozga nalazi takve formacija koja regulise ukupni tonus kore govori puno, jer su to istovremeno najnizi nivoi hijerarhije mozga. Racunarski gledano to znaci da nizi nivoi predlozenog hijerarhijskog modela deluju upravljacki na vise nivoe. Ovaj

model je suprotan svim dosadasnjim hijerahijskim modelima jer su uvek nizi nivoi samo ekstrahirali informacije za vise a odluke donosili visi nivoi. Mogucnost opstanka ovakvog modela lezi u cinjenica sto nizi nivoi ne obradjuju sve informacije vec samo one koje su najinteresantnije za odrzanje sistema. Ujedno vreme obrade takvih informacija je minimalno, jer reakcije moraju biti jako brze. Zbog toga nizi sistemi hijerahije moraju uticati na vise kako bi ukupna reakcija sistema bila skoro trenutna, a samim tim visi nivoi se moraju podciniti nizim. Ovo dovodi do zakljucka da najnizim nivoima moraju biti dostupne sve informacije koje dolaze u sistem. Priroda je to resila postavivsi retikularnu formaciju kao jednu mrezastu cilindricnu tvorevinu oko svih nervnih vlakana koji dolaze u mozdani sistem (slika 8).



Slika 8. Aferentni putevi i osećajne zone (po P.I. Poljakovu)

Druga vlakna retikularne formacije krecu se u suprotnom smeru. Ona polaze od visih nervnih formacija i idu ka nizim strukturama srednjeg mozga, hipotalamusa i mozdanog stabla. Te formacije su dobile naziv silazni retikularni sistem i one stavljaju ostale delove mozga pod kontrolom onih programa koji se stvaraju u kori velikog mozga i za cije je izvrsavanja potrebna modifikacija i modulacija stanja budnosti. Ovo govori o postojanju 'povratne veze', od visih nervnih formacija ka nizim ili od visih psihickih procesa ka nizim. Racunarski gledano, to znaci da visi nivoi hijerarhije imaju povratni uticaj na nize nivoe i to takav da oni mogu modulirati uticaj nizih nivoa ali sa ogranicenjem da on moze postojati samo do jedne granice. Visi nivoi ne mogu iskljuciti funkcije nizih jer u tom slucaju bi postojali radi sebe samih. Poznato je da se nervni sistem nalazi stalno u stanju odredjene aktivnosti i da je postojanje izvesnog tonusa neophodno za ispoljavanje bilo kakve aktivnosti.

Mogu se izdvojiti tri izvora aktivacije nervnog sistema. Prvi od izvora je proces razmene materija organizma koji leze u osnovi unutarnje ravnoteze organizma i instiktivnih procesa. Drugi izvor aktivacije je povezan sa dolaskom u organizam drazi iz okolnog sveta i on dovodi do nastanka potpuno drugih formi aktivacije koje se ispoljavaju o obliku orjentacionog refleksa. Svaka reakcija na 'novotariju', zahteva poredjenje nove sa sistemom starih drazi. Samo takvo poredjenje moze da odredi da li je data draz nova i zahteva li orjentacioni refleks ili je vec poznata sto znaci da za nju nije potrebna posebna mobilizacija organizma. Samo takav mehanizam moze da obezbedi proces privikavanja kada vise puta ponovljena draz gubi karakter novog. Ocigledno je da je u taj mehanizam ucenja ukljucen mehanizam pamcenja koji omogucuje takvu komparaciju. Znatan deo ljudske aktivnosti uslovljen je namerama i planovima, perspektivama i programima koji se formiraju u procesu svesnog zivota coveka i oni predstavljaju treci izvor aktivacije.

Tipovi izvoraaktivacije govore da mozdani sistem stalno vodi racuna o svom unutrasnjem stanju , okolini i tekucem izvrsenju programa psihicke delatnosti. Kod racunarskih sistema koji bi koristili principe mozdane organizacije bilo bi za ocekivati da sva tri izvora aktivacije budu zastupljena. Drugi i treci po samoj prirodi modela moraju biti prisutni ali se postavlja pitanje sta s prvim tipom izvora. Kod coveka postoji veliki broj redundantnih sistema ,koji preuzimaju funkcije ako glavni nosilac funkcije odkaze. Najbolji primer je disanje koje je jedna od kriticnih funkcija za zivot. Kad se presecanjem nerva koji regulise (upravlja) radom dijafragme ona iskljuci iz rada ne dolazi do gusenja vec njenu funkciju preuzimaju medjurebarni misici koji sire i skupljaju grudni kos i tako omogucuju razmenu vazduha u plucima. Ista stvar se dogadja i sa mozdanim centrima. Kad neki centar usled nekog ostecenja otkaze, drugi preuzimaju njegove funkcije u potpunom ili degradiranom obliku. To znaci ako se zeli bilo kakva redundancija u modelu (neki oblik sistema neosetljivog na greske) ovaj izvor aktivacije mora biti prisutan. U stavrnisti model bi stalno vodio racuna u kakvom se stanju (funkcionalnom) nalaza pojedine komponente hijerarhije i sami nivoi hijerarhije. Globalne odluke o stanju sistema i njegovo eventualno rekonfigurisanje vodilo bi se na najnizem nivou hijerarhije

2.3.2 Blok prijema, obrade i cuvanja informacija

Ovaj blok se nalazi u spoljnjim delovima nove kore i zauzima njene zadnje delove, ukljucujuci u sebe aparate vidne (potiljacne), slusne (slepoocne) i opsteosecajne (temene) oblasti. 'sastoji se od subkortikalnih neurona i mozdane kore. 'ti neuroni rade na principu sve ili nista primajuci odvojene impulse i prenose ih na druge grupe neurona. Drugim recima ovaj funkcionalni blok prima nadrazaje od perifernih receptora, rastavlja ih u najsitnije delove (vrsi analizu najsitnijih delova informacija), i vrsi kombinovanje u potrebne dinamicke funkcionalne strukture (vrsi sintezu u citave funkcionalne sisteme). U racunarskom modelu ovaj blok bi morao biti zastupije kao analizator i sintezator svih ulaznih informacija u sistem i sluzio bi kao podloga za buduce reakcije sistema. Organizacija ovog bloka analogno predhodnom bi morala da bude hijerahijska.

Osnovu ovog bloka sacinjavaju primarne ili projekcione zone mozdane kore, koje karakterise razvoj neurona IV aferentnog sloja, ciji znatan deo poseduje najvecu modalnu specificnost. Npr: pojedini neuroni vidnih aparata reaguju samo na nijanse boje, karakter linije, pravac kretanja. U primarnim zonama pojedinih oblasti mogu se sresti neuroni multimodalnog karaktera koje reaguju na nekoliko vrsta drazi ali njihov broj nije veci od 4 – 5 %.

Upravo ova cinjenica moze dovesti u sukob s predlozenim modelom jer najnizi nivo bi morao da vrsi najgrublju obradu informacija a nadredjeni su visem. Kako to da neki nivo koji vrsi analizu najsitnijih delova informacija moze biti na nizem nivou od nivoa koji vrsi sintezu? Odgovor lezi u nacinu analize informacija i reakcija na nju. Resetvljanje informacija na delove dato je njenom prirodom i ono samo ne znaci nista. Medjutim, reakcije sistema na neke elemente informacija (npr. bljesak, jak zvuk,

gorak ukus) moraju biti brze (trenutne) i taj nizi nivo mora tu brzu reakciju omoguciti. Tek kasnije se ustanovljava (*sintezom informacija*) sta se dogodilo i da li je reakcija sistema bila ispravna ili ne. To znaci da nizi nivoi hijerarhije modela vrse rastavljanje informacija na sastavne delove za vise nivoe, a reakcije koje oni preduzimaju na osnovu tih delova informacije su u svojoj sustini znacajnije za sistem od reakcija visih nivoa.

Nad primarnom nalaze se aparati sekundarnih zona kore u kojima vodece mesto zauzimaju neuroni II i III sloja, i oni nemaju izrazenu modalnu specificnost sto omogucuje da se kombinuju drazi koje dolaze u potrebne 'funkcionalne obilke' i tako ostvaruje sinteticka funkcija.

Saznajna delatnost coveka se nikad ne oslanja samo na jednu vrstu drazi vec predstavlja rezultat poli- modalne delatnosti. Zbog toga sasvim je prirodno sto ona mora da se oslanja na zajednicki rad celog sistema zona kore velikog mozga. Funkciju obezbedjivanja takvog zajednickog rada cele grupe analizatora obezbedjuju i nose tercijarne zone bloka III kako se one nazivaju zone prepokrivanja kortikalnih delova razlicitih analizatora.

Tercijerne zone zadnjih delova mozga se skoro u celosti sastoje od celija II i III sloja kore sto znaci da se njihova funkcija skoro u potpunosti svodi na integraciju stimulusa koji dolaze sa raznih analizatora. Smatra se da najveci broj neurona tercijarnog sloja je multimodalnog karaktera i da reaguju na kompleksne oznake sredine (oznake raspoerda u prostoriji, broj elemenata ...) na koje ne reaguju neuroni prva dva sloja.

Delatnost tercijarnih zona zadnjih delova kore je neophodna ne samo za uspesnu sintezu ocigledne informacije vec i za prelazak od nivoa neposredne ocigledne sinteze na nivo simbolickih procesa, za operisanje sa znacenjima reci, slozenim gramatickim i logickim strukturama, sa sistemima brojeva i apstraktnim odnosima. Drugim recima tercijarne zone predstavljaju aparate cije je ucesce neophodno za pretvaranje ociglednog zapazanja u apstraktno misljenje i za ocuvanje u pamcenju organizovanog iskustva. Organizacija sekundarnih i tercijerzonih analizatora govore da za detaljnu obradu prispelih informacija treba puno vise vremena nego za primarnu analizu. To znaci da reakcije sistema su puno sporije ali puno kvalitetnije i da nivoi hijerarhije koji vrsi sintezu informacija moraju biti u visim nivoima hijerarhije, jer za fine i visoke akcije po svojoj strukturi moraju pored nizih nivoa biti ukljuceni i visi. Ovo sto je izlozeno suprotno je svim dosadasnjim modelima analize i sinteze informacija. Kao potvrda ovome pristupu moze posluziti podatak da ne spomenute drazi tipa, bijeska, gorkog ukusa, jakih zvukova, reakcije mozga nisu na nivou nove i stare mozdane kore vec na nivou limbickog sistema koji se nalazi 'ispod', nove i stare kore. Za detaljnu analizu i sintezu tih drazi kad se ustanovljavaju razlozi njihove pojave, reakcije su na nivou nove i stare kore.

Aparati treceg funkcionalnog bloka nalaze se u prednjim delovima velikih hemisfera, *ispred prednje centralne vijuge*. Izlazna vrata, tog bloka predstavlja motorna zona kore velikog mozga, ciji V sloj sadrzi Becove gigantske piramidalne celije, cija vlakna vode ka motornim jedrima kicmene mozdine. U toj zoni su u najvecem stepenu zastupljeni organi koji imaju najfunkcionalniji znacaj i kojima je potrebna najfinija moguca regulacija. Prirodno je da motorna struktura impulsa koje salje ovaj deo kore mora biti dobro pripremljena tj. ukljucena u odredjene programe, jer bez takve pripreme se ne bi mogli obezbediti svrsishodni pokreti. Sa stanovista modela primarni sloj ovog bloka bit ce najvisi sloj hijerarhije predlozenog modela. Razlog za to je ocigledan jer on bi u racunarskom modelu upravljao aktuatorima koji upravljaju procesom ili su veza s okolinom.

Priprema motornih impulsa ne moze biti ostvarena samo piramidalnim celijama vec mora biti obezbedjena kako aparatom prednje centralne vijuge tako i visim aparatima 'sekundarnih', zona motorne kore. Prednja centralna vijuga predstavlja projekcionu zonu, izvrsni aparat mozdane kore. Odlucujuci znacaj u pripremi motornih impulsa imaju sekundarne i tercijarne zone, koje se potcinjavaju istim principima hijerarhijske strukture i silazne specificnosti kao i predhodna dva bloka ali u obrnutom redosledu. Ulogu sekundarnih zona igraju premotorni delovi ceone oblasti mozga. Drazenje tih delova kore izaziva citave kompleksne pokrete kao sto su pokreti ociju, glave i celog tela i pokreti ruke koja nesto uzima... Sa stanovista modela znaci da neposredno nizi nivo vrsi organizaciju (pripremu) reakcija sistema, ali ne samo sa stanovista proracuna (programa) reakcije sistema vec i sa stanovista buduce reakcije sistema.

Najznacajnij deo ovog funkcionalnog bloka predstavljaju prefrontalni delovi mozga koji su sastavljeni od velikog broja celija II i III sloja i predstavljaju tercijerne zone. Oni igraju najznacajniju ulogu u formiranju namera i programa i regulisanja i kontrole najslozenijih formi ponasanja coveka. Upravo zbog toga, prefrontalna oblast ima najbogatiji sistem veza sa drugim delovima mozga kao i sa retikularnom formacijom. Zahvaljujuci dvostrukom karakteru tih veza prefrontalni delovi kore se nalaze u posebno povoljnom polo-polozaju i u odnosu na ponovnu preradu najslozenijih uzbuda, koje dolaze od svih delova mozga i za organzaciju upravljackih impulsa koji omogucavaju da se obavlja regulaciono delovanje na sve te strukture. Ako najvisi nivo hijerarhije je neposredni izvrsilac radnje onda nizi nivoi moraju pripremati globalne instrukcije za njihovo izvrsenje. Ta priprema globalnih radnji je tesno povezana s predhodno dva opisana funkcionalna bloka modela i ne moze se vrsiti bez njih. Nizi nivoi hijerarhije donose globalne odluke o reakciji sistema dok visi nivoi samo izvrsavaju globalne odluke i eventualno vrse njihovo doterivanje. Ovde je vazno napomenuti znacaj povratnih informacija koje ovim nivoima govore o izvrsenju predvidjenih programa i reakcija. Na osnovu njih ovaj blok vrsi korekciju (doterivanje) svojih programa tj. globalnih akcija.

Od odlucujuceg znacaja je cinjenica da ceone regije mozga poseduju jake snopove uzlaznih i silaznih veza sa retikularnom formacijom. Te oblasti nove kore dobijaju impulse od sistema prvog funkcionalnog bloka dok istobvemeno one vrse intezivan modulacioni uticaj na formacije retikularne formacije, dajuci njenim aktivirajucim impulsima diferencirani karakter i usaglasavajuci ih sa onim dinamickim semama ponasanja koje se formiraju neposredno u ceonoj kori mozga. Ovo istice usku funkcionalnu saradnju prvog i treceg funkcionalnog bloka i vezu sa svim

2.3.3 Blok programiranja, regulisanja i kontrole slozenih formi delatnosti

Prijem, prerada i cuvanje informacija predstavlja samo jednu stranu psihickog zivota coveka. Njenu drugu stranu predstavlja organizacija aktivne, svesne psihicke delatnosti. Covek ne reaguje samo pasivno na signale koje prima. On formira planove i programe svojih akcija, prati njihovo ostvarivanje i regulise svoje ponasanje, usaglasavjuci ga sa tim planovima i programima. Na kraju, on kontrolise svoju svesnu delatnost, poredeci efekat svojh akcija sa polaznim namerama i korigujuci pocinjene greska. vka.

komponentama nervnog sistema. Sa stanovista modela to znaci da su uzbudni i programirajuci sistemi tesno povezani. Programirajuci moze delovati na uzbudni, ali konacne odluke za sistem uvek donosi uzbudni. Kod modela treba istaci da se sva tri funkcionalna bloka

- aktivirajuci
- obradjujuci
- programirajuci nalaze medjusobno izmesani na svim nivoima hijerarhije i to ne kao odvojene komponente vec integralno u svim komponentama hijerarhije.

3. Z A K L J U C A K

Iz svega do sada iznesenog sledi da racunarski sistem koji bi pokusao iskoristiti osnovne postulate rada mozga mora zadovoljiti sledece globalne zahteve:

- Da je to dinamicka hijerarhijska struktura sa vise nivoa
- Da je stanje takvog sistema odredjeno spoljnom okolinom, unutrasnjim stanjem i funkcijom koja se trenutno izvrsava
- Da su svi ulazi u sistem direktno na najnizem nivou a svim ostalim nivoima se prosledjuje samo onaj podskup ulaznih informacija koji mu je potreban za obavljanje njegove funkcije
- Da najnizi nivo na osnovu informacija iz spoljnjeg sveta, unutrasnjeg stanja i funkcije koja se obavlja u sistemu donosi odluke o buducoj reakciji sistema (funkciji) i na osnovu nje inicira dalju analizu i sintezu ulaznih informacija. Istovremeno vrsi distribuciju odabrane reakcije ostalim elementima sistema
- Da poseduje sledece tri komponente ukonponovane na svim nivoima hijerarhije
  - aktivirajucu
  - obradjujucu
  - programirajucu
- Da ima dvosmernu komunikaciju izmedju nivoa i unutar nivoa hijerarhije
- Da na osnovu odredjene buduce reakcije se izvrsi rekonfiguracija sistema kako bi se sistem najbolje prilagodio nastaloj situaciji
- Da najnizi nivo svoje odluke i reakcije donosi skoro trenutno, a svi ostali nivoi u zakasnjenju za njim
- Da postoje mogucnosti uticaja visih nivoa na nize u procesu odredjivanja buduce reakcije sistema
- Da svi nivoi imaju mogucnost uticaja na kontrolisani proces stim da sto je nivo visi to njegov uticaj mora biti viseg kvaliteta
- Da ako nanizi nivo odredi pogresnu reakciju sistema ne postoji niko ko bi tu reakciju promenio u drugu

4. L I T E R A T U R A

1. James S. Albus. "Brains, Behavior, and Robotics" BYTE Books, Subsidiary of McGraw-Hill, 1981.
2. A.R. Lurija. "Osnovi neuropsihologije" prevod sa ruskog, Nolit, Beograd, 1983.
3. Lav Vigotski "Misljenje i govor" prevod sa ruskog, Nolit, Beograd, 1983
4. George D. Watts. "Dynamic Neuroscience", Harper and Row Publisher, 1975.
5. John F. Jarvis. "Robotics", Computer, strana 283-292, oktobar 1984.
6. William T. Powers. "The nature of robots" Byte, jun, jul, avgust, septembar 1979
7. Andrew Filo. "Designing a robot from nature" Byte, februar, mart 1979.
8. Steven & Svetiana Kartashev. "Dynamic archictetures: Problems and Solving" Computer, juli 1978.

KOMUNIKACIJSKI IN APLIKACIJSKI PROCESI V VIŠJIH NIVOJIH REFERENČNEGA MODELA OSI

INFORMATICA 4/87

UDK 681.3

B. Jerman - Blažič in M. Kapus - Kolar
Institut Jožef Stefan, Ljubljana

abstract
The paper deals with the communication concepts and functions associated with the upper layers of the ISO/OSI Reference Model. The basic communication functions in the upper layers of the model are compared to the information processing functions of a computer system. The role and the task of both functions in a distributed system are explained in an understandable way.

povzetek
Prispevek obravnava komunikacijske koncepte in funkcije višjih nivojev referenčnega modela OSI. Primerjali smo naloge komunikacijskih funkcij v modelu z nalogami funkcij za obravnavo in obdelavo informacij v računalniškem sistemu. Vlogi obeh funkcij v distribuiranem sistemu sta razloženi na enostaven način.

1. Uvod

Vsaka komunikacija med dvema ali več partnerji vključuje komunikacijske funkcije, ki jih identificiramo zaradi preglednosti in načrtovanja sistemov kot tri razrede komunikacijskih funkcij. Ta razdelitev velja ne glede na to, ali komunikacijo izvajajo ljudje s prenašanjem pisem ali komunicirajo stroji z elektronskim izmenjavanjem podatkov. Standardi iz referenčnega modela OSI obravnavajo te funkcije za primer komuniciranja preko naprav za prenos podatkov, na primer preko mrež s preklapljanje paketov ali vodov, najetih linij, lokalnih mrež ali javnih mrež za prenos podatkov

Komunikacijske funkcije vsakega od razredov obravnavajo različne, dobro ločljive vidike komuniciranja. Ti so:

1. **prenos podatkov** - to so komunikacijske funkcije za prenos predstavitev informacij (podatkov) iz enega končnega sistema v drugega, s pogostostjo napak, sprejemljivo za aplikacijski proces.

2. **povezava** - to so komunikacijske funkcije, ki omogočajo aplikacijskim procesom, da vzpostavijo dialog. Ker v vsakem računalniškem sistemu pogosto obstaja več aplikacijskih procesov, ni dovolj, da se podatki dostavijo v izbrani končni sistem. Potrebne so še funkcije, ki ugotovijo, kateri aplikacijski proces pošilja oz. sprejema informacije.

3. **sodelovanje** - to so komunikacijske funkcije, ki omogočajo aplikacijskim procesom smiselno komunikacijo in izvajanje procedur za porazdeljeno procesiranje informacij.

Te funkcije so del sistemske opreme sodobnih računalniških sistemov, ker je le s pomočjo teh funkcij možna komunikacija med porazdeljenimi aplikacijskimi procesi z napravami za prenos podatkov. Aplikacijskim procesom morajo biti na razpolago v računalniških sistemih, v katerih so aplikacije locirane.

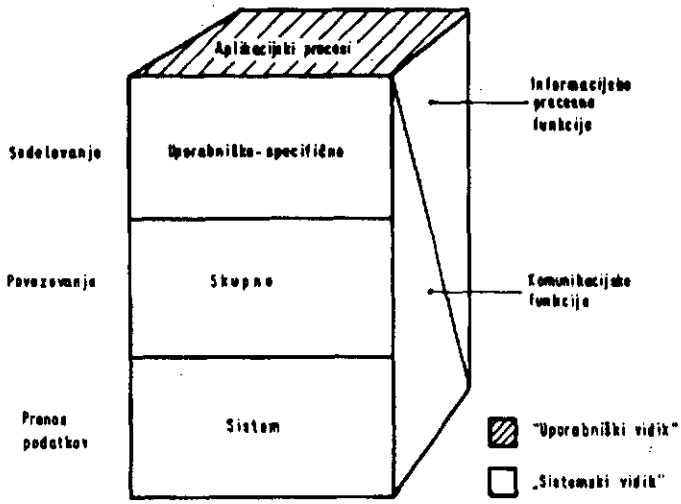
Slika 1. ilustrira odnos med informacijsko-procesnimi funkcijami enega računalniškega sistema in komunikacijskimi funkcijami aplikacijskega procesa. Za uporabnika se komunikacijske funkcije izvajajo kot del aplikacijskega procesa, z vidika sistema pa se te funkcije izvajajo kot funkcije ločenih nivojev. Nekatere od teh funkcij so tesno povezane s specifičnimi komunikacijskimi potrebami uporabnikove aplikacije, zato se te funkcije specificirajo za vsako aplikacijo posebej. Uporabniško specifične funkcije se vežejo na funkcije v sistemu, ki so splošnega pomena in so uporabne pri večjemu številu različnih aplikacij. Splošnih funkcij v računalniškem sistemu, ki ima vgrajene funkcije za komunikacijo, ni treba posebej specificirati, ker so uporabniku na razpolago kot del sistema. Najbolj elementarne komunikacijske funkcije so sistemsko specifične funkcije, ki delajo s konkretnimi napravami za prenos podatkov.

2. Opredelitev komunikacijskih funkcij v OSI RM

OSI RM identificira naštete tri nivoje komunikacijskih funkcij in jih dalje deli v sedem nivojev, ki ustrezajo sedmim nivojem tega modela. Vsak od nivojev nudi del množice vseh funkcij, ki so potrebne za sodelovanje oddaljenih aplikacijskih procesov. OSI RM podaja abstrakten pogled na te funkcije, ki ga prikazuje slika 2. Slika 2. kombinira različne poglede na aplikacijski proces in

prikazuje odnos med tremi nivoji komunikacijskih funkcij in nivoji OSI RM. Tudi v OSI RM so funkcije prenosa podatkov ločene od funkcij povezovanja in sodelovanja. S to ločitvijo funkcij na meji med transportnim nivojem in nivojem seje je naznačena razlika med funkcijami prenosa podatkov (spodnjimi nivoji) in funkcijami porazdeljenega procesiranja podatkov (zgornjimi nivoji).

V nadaljevanju privzemamo, da so funkcije spodnjih nivojev realizirane. Te funkcije



Slika 1 Različni vidiki komunikacijski funkcij

podpirajo funkcije zgornjih nivojev s prenašanjem predstavitev informacij med končnimi sistemi v obliki nizov binarnih oktetov in nudijo servis s kvaliteto, ki je sprejemljiva za dane aplikacijske procese. Funkcije vseh spodnjih nivojev skupaj nudijo **servis transporta podatkov**, ki podpira izvajanje funkcij zgornjih nivojev.

Kot ilustrira slika 3., je rezultat izvajanja funkcij spodnjih nivojev prenos nizov binarnih oktetov iz oddajnega v sprejemni končni sistem (lahko preko poljubnega števila sistemov - posrednikov). To je analogno prenosu pisem ali magnetnih trakov. Servis, ki ga nudijo spodnji nivoji, je kot servis dostave pošiljke "od vrat do vrat", ki ga nudi poštni sistem. Slovenski del JUPAK-a na sedanji stopnji razvoja nudi le približek tega servisa (prve tri, namesto štirih nivojev). Navpične puščice na sliki 3 predstavljajo tok podatkov iz enega končnega sistema v drugega. Vodoravne puščice predstavljajo interakcije ali protokole, potrebne v vsakem od nivojev za izvajanje funkcij transporta podatkov. Po terminologiji OSI so to funkcije in protokoli, definirani z OSI standardi za fizični, linijski, mrežni in transportni nivo, ki nudijo zgornjim nivojem servis transporta podatkov.

Preden aplikacijski procesi lahko uporabljajo podatke, jih morajo sprejeti in pravilno interpretirati. V vsakem računalniškem sistemu lahko obstaja veliko aplikacijskih procesov. Zato se morata aplikacijska procesa, ki želita izmenjavati podatke, najprej dogovoriti za komunikacijo in določiti pravila, po katerih bo potekala. Ker vsak proces po svoje **predstavlja** informacije, se morata dogovoriti tudi za enotno predstavitev informacij pri izmenjavi.

Za to potrebne funkcije in protokoli omogočajo aplikacijskim procesom v različnih računalniških sistemih **povezavo in sodelovanje**. V OSI RM so definirane kot funkcije nivoja seje, predstavitvenega in aplikacijskega nivoja.

3. Povezovanje - nivo seje

Nivo seje nudi funkcije za povezavo dveh aplikacijskih procesov v odnos logične komunikacije in funkcije za organizacijo in sinhronizacijo njunega dialoga. Pri tem se uporabljajo mehanizmi za vzpostavitev, vzdrževanje in razpustitev **povezav na nivoju seje**. Povezava na nivoju seje je dogovor med dvema aplikacijskima procesoma o vključitvi v kontroliran dialog za izmenjavo podatkov. Aplikacijski procesi torej izmenjujejo podatke s pomočjo povezav na nivoju seje. S pomočjo mehanizmov povezav na nivoju seje lahko aplikacijski procesi pošiljajo podatke, ki jih sprejemni sistem dostavi naslovljenemu aplikacijskemu procesu.

Preden začneta dva aplikacijska procesa izmenjavati podatke, morata vzpostaviti povezavo na nivoju seje. Vsak aplikacijski proces lahko sam sproži pobudo za vzpostavitev povezave na nivoju seje, lahko pa tudi sprejme zahtevo za vzpostavitev povezave od drugega aplikacijskega procesa. Ko je to opravljeno, je odnos med aplikacijskima procesoma vzpostavljen in procesa lahko začneta izmenjavati podatke.

Povezavo na nivoju seje lahko obravnavamo kot povezavo med dvema aplikacijskima procesoma preko nivojev seje njunih končnih sistemov. Vendar ne smemo pozabiti, da je povezava na nivoju seje odvisna od povezav, vzpostavljenih na nižjih nivojih za izvajanje funkcij nivoja seje. Te povezave so potrebne za **prenos podatkov in protokolnih informacij** med končnima sistemoma. Privzeli bomo, da te povezave obstajajo, to je da jih spodnji nivoji vzpostavljajo in razpuščajo v skladu s potrebami povezav in funkcij na nivoju seje.

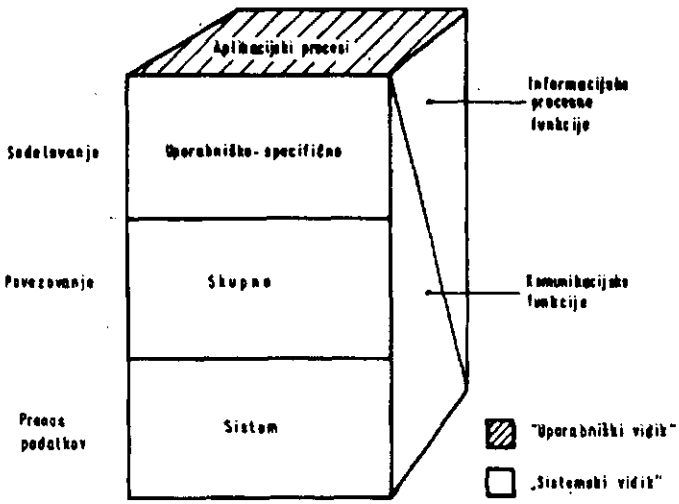
Povezava na nivoju seje se lahko raztegne tudi čez več povezav na spodnjih nivojih. To lahko primerjamo z dvema šahistoma, ki igrata partijo preko telefona. Podobno kot povezava na nivoju seje, partija teče, dokler se eden od šahistov ne odloči za konec. Dejanska povezava preko telefona pa se lahko medtem večkrat vzpostavi in spet razpusti, saj je potrebna samo, kadar eden od šahistov želi drugemu sporočiti naslednjo potezo.

Med vzpostavljanjem povezave se dva aplikacijska procesa dogovorita za pravila **dialoga**, ki se bodo uporabljala pri komunikaciji med njima. Pomen pojma **dialoga** na nivoju seje je analogen kot pri komunikaciji med ljudmi. Prvi tip dialoga označuje pretok informacije samo v eni smeri, na primer od predavatelja k poslušalcem. Pri drugem tipu dialoga teče informacija v obeh smereh, vendar hrati samo v eni smeri, ko na primer predavatelj prekinja predavanje, da lahko poslučalci postavljajo vprašanja. V tem primeru si predavatelj in poslušalci izmenjujejo pravico govora. Primer tretjega tipa je komunikacija dveh ljudi, ki kontrolirata časovno kritičen proces (na primer pristajanje letala), zato morata nujno imeti oba ves čas pravico do govorjenja. Bistveno je, da potreben tip dialoga izhaja iz narave aktivnosti.

Nivo seje nudi številne tipe dialoga za različne tipe aplikacijskih procesov. Nekateri aplikacijski procesi (na primer tisti s področja kontrole časovno kritičnih procesov)

prikazuje odnos med tremi nivoji komunikacijskih funkcij in nivoji OSI RM. Tudi v OSI RM so funkcije **prenosa podatkov** ločene od funkcij povezovanja in sodelovanja. S to ločitvijo funkcij na meji med transportnim nivojem in nivojem seje je naznačena razlika med funkcijami prenosa podatkov (spodnjimi nivoji) in funkcijami porazdeljenega procesiranja podatkov (zgornjimi nivoji).

V nadaljevanju privzemamo, da so funkcije spodnjih nivojev realizirane. Te funkcije



Slika 1 **Različni vidiki komunikacijski funkcij**

podpirajo funkcije zgornjih nivojev s prenašanjem predstavitev informacij med končnimi sistemi v obliki nizov binarnih oktetov in nudijo servis s kvaliteto, ki je sprejemljiva za dane aplikacijske procese. Funkcije vseh spodnjih nivojev skupaj nudijo **servis transporta podatkov**, ki podpira izvajanje funkcij zgornjih nivojev.

Kot ilustrira slika 3., je rezultat izvajanja funkcij spodnjih nivojev prenos nizov binarnih oktetov iz oddajnega v sprejemni končni sistem (lahko preko poljubnega števila sistemov - posrednikov). To je analogno prenosu pisem ali magnetnih trakov. Servis, ki ga nudijo spodnji nivoji, je kot servis dostave pošiljke "od vrat do vrat", ki ga nudi poštni sistem. Slovenski del JUPAK-a na sedanji stopnji razvoja nudi le približek tega servisa (prve tri, namesto štirih nivojev). Navpične puščice na sliki 3 predstavljajo tok podatkov iz enega končnega sistema v drugega. Vodoravne puščice predstavljajo interakcije ali protokole, potrebne v vsakem od nivojev za izvajanje funkcij transporta podatkov. Po terminologiji OSI so to funkcije in protokoli, definirani z OSI standardi za fizični, linijski, mrežni in transportni nivo, ki nudijo zgornjim nivojem servis transporta podatkov.

Preden aplikacijski procesi lahko uporabljajo podatke, jih morajo sprejeti in pravilno interpretirati. V vsakem računalniškem sistemu lahko obstaja veliko aplikacijskih procesov. Zato se morata aplikacijska procesa, ki želita izmenjavati podatke, najprej dogovoriti za komunikacijo in določiti pravila, po katerih bo potekala. Ker vsak proces po svoje **predstavlja** informacije, se morata dogovoriti tudi za enotno predstavitev informacij pri izmenjavi.

Za to potrebne funkcije in protokoli omogočajo aplikacijskim procesom v različnih računalniških sistemih **povezavo in sodelovanje**. V OSI RM so definirane kot funkcije nivoja seje, predstavitvenega in aplikacijskega nivoja.

3. Povezovanje - nivo seje

Nivo seje nudi funkcije za povezavo dveh aplikacijskih procesov v odnos logične komunikacije in funkcije za organizacijo in sinhronizacijo njunega dialoga. Pri tem se uporabljajo mehanizmi za vzpostavitev, vzdrževanje in razpustitev **povezav na nivoju seje**. Povezava na nivoju seje je dogovor med dvema aplikacijskima procesoma o vključitvi v kontroliran dialog za izmenjavo podatkov. Aplikacijski procesi torej izmenjujejo podatke s pomočjo povezav na nivoju seje. S pomočjo mehanizmov povezav na nivoju seje lahko aplikacijski procesi pošiljajo podatke, ki jih sprejemni sistem dostavi naslovljenemu aplikacijskemu procesu.

Preden začneta dva aplikacijska procesa izmenjavati podatke, morata vzpostaviti povezavo na nivoju seje. Vsak aplikacijski proces lahko sam sproži pobudo za vzpostavitev povezave na nivoju seje, lahko pa tudi sprejme zahtevo za vzpostavitev povezave od drugega aplikacijskega procesa. Ko je to opravljeno, je odnos med aplikacijskima procesoma vzpostavljen in procesa lahko začneta izmenjavati podatke.

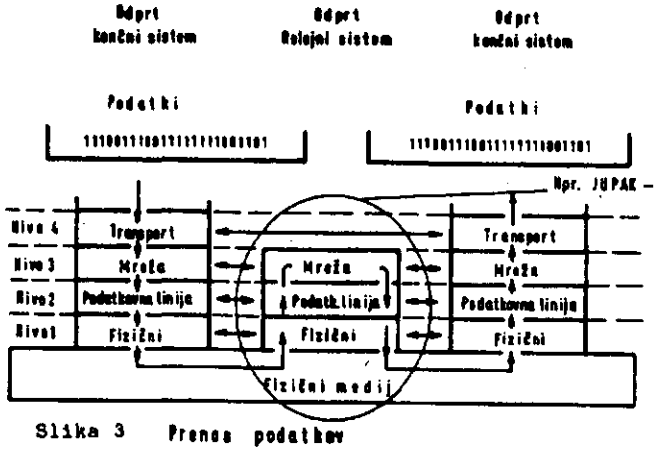
Povezavo na nivoju seje lahko obravnavamo kot povezavo med dvema aplikacijskima procesoma preko nivojev seje njunih končnih sistemov. Vendar ne smemo pozabiti, da je povezava na nivoju seje odvisna od povezav, vzpostavljenih na nižjih nivojih za izvajanje funkcij nivoja seje. Te povezave so potrebne za prenos **podatkov** in **protokolnih informacij** med končnima sistemoma. Privzeli bomo, da te povezave obstajajo, to je da jih spodnji nivoji vzpostavljajo in razpuščajo v skladu s potrebami povezav in funkcij na nivoju seje.

Povezava na nivoju seje se lahko raztegne tudi čez več povezav na spodnjih nivojih. To lahko primerjamo z dvema šahistoma, ki igrata partijo preko telefona. Podobno kot povezava na nivoju seje, partija teče, dokler se eden od šahistov ne odloči za konec. Dejanska povezava preko telefona pa se lahko medtem večkrat vzpostavi in spet razpusti, saj je potrebna samo, kadar eden od šahistov želi drugemu sporočiti naslednjo potezo.

Med vzpostavljanjem povezave se dva aplikacijska procesa dogovorita za pravila **dialoga**, ki se bodo uporabljala pri komunikaciji med njima. Pomen pojma **dialoga** na nivoju seje je analogen kot pri komunikaciji med ljudmi. Prvi tip dialoga označuje pretok informacije samo v eni smeri, na primer od predavatelja k poslušalcem. Pri drugem tipu dialoga teče informacija v obeh smereh, vendar hrati samo v eni smeri, ko na primer predavatelj prekinja predavanje, da lahko poslučalci postavljajo vprašanja. V tem primeru si predavatelj in poslušalci izmenjujejo pravico govora. Primer tretjega tipa je komunikacija dveh ljudi, ki kontrolirata časovno kritičen proces (na primer pristajanje letala), zato morata nujno imeti oba ves čas pravico do govorjenja. Bistveno je, da potreben tip dialoga izhaja iz narave aktivnosti.

Nivo seje nudi številne tipe dialoga za različne tipe aplikacijskih procesov. Nekateri aplikacijski procesi (na primer tisti s področja kontrole časovno kritičnih procesov)

odprtimi sistemi in jih ISO razvija kot skupna orodja za katerikoli tip aplikacij. Nekatere standarde aplikacijskega nivoja razvijajo druga standardizacijska telesa za tipe aplikacij splošnega pomena (npr. za delo s sporočili in izmenjavo dokumentov), medtem ko se ostali razvijajo za uporabo v posebnih dejavnostih (npr. v bančništvu, trgovini, letalskem prometu, knjižnicah itd.)



Slika 3 Prenos podatkov

Nekateri standardi aplikacijskega nivoja podpirajo splošne tipe aplikacij, ki so zgodovinsko povezane z uporabo računalniških sistemov. To so standardi za prenos datotek med porazdeljenimi sistemi in delo z njihovo vsebino (prenos datotek, dostop do datotek in upravljanje z datotekami), dostop do računalniških sistemov preko terminalov (servis virtualnega terminala) in za izvajanje poslov v porazdeljenih sistemih (prenos in manipulacija poslov). Te standarde razvija ISO za podporo množice raznih tipov aplikacij, pri katerih so potrebne te splošne procesne funkcije.

5. Predstavitveni nivo sodelovanja

Standardi aplikacijskega nivoja definirajo procedure in tipe informacij, potrebne za sodelovanje med porazdeljenimi aplikacijskimi procesi. Definicija tipov informacij, ki se uporabljajo v danem aplikacijskem procesu, je **abstraktna sintaksa** podatkov. Za sodelovanje aplikacijskih procesov je potrebno, da uporabljajo isto abstraktno sintakso.

Podatkovni elementi, definirani s standardi aplikacijskega nivoja, so abstraktne definicije informacij, ki se bodo prenašale in morajo biti nekako zakodirane. Način kodiranja v posameznem računalniškem sistemu je odvisen od konvencij, ki se uporabljajo v tem sistemu, ter od prevajalnikov, zbirnikov itd. , ki so bili uporabljeni za kreiranje aplikacijskega procesa. Vse konvencije za predstavitev informacij v računalniškem sistemu skupaj imenujemo **konkretna sintaksa** informacij. Pri sodelovanju heterogenih sistemov je običajno, da so informacije lokalno predstavljene v vsakem sistemu po drugačnih konvencijah. Konkretno sintakso za predstavitev informacij v posameznem sistemu imenujemo njegova **lokalna sintaksa**.

Pri sodelovanju morajo aplikacijski procesi kodirati informacije po istih pravilih. Čeprav ni nujno, da sodelujoča procesa uporabljata enako lokalno sintakso, se morata dogovoriti za konkretno sintakso za kodiranje informacij, ki se bodo prenašale med njima. Konkretno sintakso, ki se uporablja pri prenosu, imenujemo **prenosna sintaksa**. Prenosna sintaksa je lahko enaka lokalni sintaksi enega ali obeh sistemov, kar pa ni nujno. Bistveno je, da se sistema dogovorita za določeno prenosno sintakso in sta sposobna prevajati med svojo lokalno sintakso in izbrano prenosno sintakso.

Mehanizmi predstavitvenega nivoja omogočajo aplikacijskim procesom definicijo in izbor prenosne sintakse za njihovo komunikacijo. Prenosno sintakso izberejo z dogovarjanjem med sistemoma funkcije predstavitvenega nivoja v imenu aplikacijskih procesov iz množice prenosnih sintaks, ki jih podpirata oba sistema. Med komunikacijo se funkcije predstavitvenega nivoja lahko dogovorijo za spremembo prenosne sintakse in izberejo novo v skladu s standardi aplikacijskega nivoja. To se na primer zgodi v primeru, ko dva aplikacijska procesa komunicirata po standardih za delo z datotekami in se dogovorita za določeno prenosno sintakso za izmenjavo znakovnih nizov. V nekem trenutku se pojavi potreba po prenosu datoteke, ki vsebuje cela števila. V tej točki lahko funkcije predstavitvenega nivoja izberejo novo prenosno sintakso - za predstavitev celih števil.

Prenosna sintaksa je rezultat uporabe množice pravil kodiranja na abstraktni sintaksi, ki jo uporabljata sodelujoča aplikacijska procesa. Kot rezultat standardov aplikacijskega nivoja se definirajo vedno nove kategorije informacij, ki predstavljajo informacije v raznih tipih aplikacij. Vsak od teh standardov definira za predstavitev informacij podatkovne tipe različne kompleksnosti. Če naj uporabljena prenosna sintaksa odraža semantiko komunikacije, mora biti sposobna predstaviti vse potrebne podatkovne tipe in pripadajoče podatkovne vrednosti. Bilo bi na primer nemogoče predstaviti vse informacije (t.j. simbole) 8-bitno kodirane znakovne množice, če je bilo za prenos izbrano 5-bitno kodiranje.

Vsak aplikacijski proces mora nujno imeti pripravljeno eno ali več prenosnih sintaks za komunikacijo z drugimi sistemi. Zato mora biti vsakemu standardu aplikacijskega nivoja pridružena prenosna sintaksa, ki je dovolj močna za izražanje kompletne semantike standarda. Ena od možnih rešitev bi bila seveda definicija ene same prenosne sintakse za podatkovne elemente določenega standarda aplikacijskega nivoja, morda po kakšnem obstoječem standardu za kodiranje podatkovnih elementov. Tako bi lahko nek standard predpisoval, da se morajo znakovni nizi vedno prenašati kodirani po ISO 646. To bi pomenilo, da bi dva sistema, ki uporabljata ta standard, nujno morala prevajati svojo lokalno sintakso v ISO 646, tudi če bi bili njuni lokalni sintaksi enaki, kar ne bi bilo gospodarno.

Zato je bil pri zgornjih nivojih izbran pristop specifikacije potreb po prenosu informacij na aplikacijskem nivoju v obliki splošne abstraktne sintakse, brez specifikacije specifične prenosne sintakse. Primerno prenosno sintakso mora nuditi predstavitveni nivo.

Definicija in registracija prenosnih sintaks je posebno področje standardizacije. Vsaka prenosna sintaksa se definira in registrira glede na abstraktno sintakso, ki jo podpira. Tako lahko predstavitveni nivo izbere prenosne sintakse, ki so najprimernejše za določen standard aplikacijskega nivoja. Britanski nacionalni računski center (National Computing Center) je prevzel dolžnosti telesa za

registracijo kod oziroma kodne sintakse za specifične aplikacije.

Standardi predstavitvenega nivoja definirajo mehanizme za dogovarjanje o množici sprejemljivih predstavitvenih sintaks ter izbor in spremembo izbora ene izmed njih. Pri tem predpostavljajo, da so obravnavane sintakse uradno registrirane, vendar ne izključujejo možnosti dinamične definicije posebnih prenosnih sintaks s pomočjo jezika za definicijo podatkov.

6. Sklepne misli

ISO in druge standardizacijske organizacije intenzivno delajo na standardizaciji zgornjih nivojev. Z napredovanjem tega dela postaja vedno bolj viden njegov odnos do drugih vidikov procesiranja informacij. Ceprav področje OSI standardov pokriva samo komunikacijske vidike povezovanja odprtih sistemov, morajo ti standardi upoštevati tudi širše zahteve.

Zato je nujno, da načrtovalci in strokovnjaki, ki delajo na sorodnih področjih standardizacije, priznajo uporabnost komunikacijskih servisov, ki se standardizirajo v okviru OSI, strokovnjaki za OSI pa morajo pri razvoju novih servisov upoštevati potrebe aplikacij, ki jih bodo uporabljale.

7. Reference

Uporabljeni so bili delovni dokumenti in standardi ISO za področje OSI.